Technical Report
RAL-TR-97-063

Interoperating Database Systems: Issues and Architectures

S Hamill M Dixon B J Read and J R Kalmus

November 1997

COUNCIL FOR THE CENTRAL LABORATORY OF THE RESEARCH COUNCILS

The Central Laboratory of the Research Councils
Library and Information Services
Rutherford Appleton Laboratory
Chilton
Didcot
Oxfordshire
OX11 0QX
Tel: 01235 445384 Fax: 01235 446403
E-mail library@rl.ac.uk

ISSN 1358-6254

INTEROPERATING DATABASE SYSTEMS: ISSUES AND ARCHITECTURES

Sean Hamill

School of Computing,
Leeds Metropolitan University, Beckett Park, LEEDS LS6 3QS, UK.

Maurice Dixon

Department of Computing, Information Systems and Mathematics,
London Guildhall University, 100 Minories, LONDON EC3N 1JY, UK.

Brian J Read & John R Kalmus

Department for Computation and Information,
Rutherford Appleton Laboratory, Chilton, DIDCOT, Oxon, OX11 0QX, UK.

CONTENTS

PREFACE

This report is based on a working paper produced in partial fulfilment of the Degree of Doctor of Philosophy of Leeds Metropolitan University. The purpose of the survey was "To Understand the Issues in Architectures for Database Federations" and to highlight areas where further work is required.

ACKNOWLEDGEMENTS

We are grateful to Professor Keith Jeffery of the RAL Information Systems Engineering Division for encouraging us to undertake research in the field of heterogeneity in database systems. Leeds Metropolitan University provided a research bursary under which this work was undertaken. Rajesh Kaurhal of Bradford University provided access to a range of reprints.

ABSTRACT

Information systems have been developed in recent years to meet particular local business needs. The information residing in those local systems represents a major business asset. However business needs change during the lifetime of an information system. The changes can arise either through the discovery of new business needs or through the restructuring of business departments. In many cases this leads to the need for several information systems to be used in a collaborative way.

The outcome is that business has to contend with information systems which are *distributed, autonomous* and *heterogeneous*. The systems are distributed in the sense that they are physically residing on different computers, possibly separated by substantial distances. The systems are autonomous because they have been acquired and operated independently; indeed even when they are required to co-operate operational independence may still be required. The systems are heterogeneous because they have been designed separately, implemented on different hardware and software platforms, and used by operations staff with different world views.

This report considers the architectures proposed to allow these distributed, autonomous, and heterogeneous systems to interoperate. Specifically we consider:

1. **Distributed Database Systems**: These systems are tightly coupled and assume a top-down design and implementation. The distributed database management systems are available as proprietary products. These systems are not appropriate where there is design autonomy and are not readily applicable to platform heterogeneity due to restrictions in the available gateways to different DBMSs.

2. **Global Schema Multidatabases:** These systems attempt to provide an overarching global schema for the set of databases. They are tightly coupled. The global schema itself may represent only a subset of the local schema definitions. A particular problem is lack of resilience to local changes to local schema. More generally the heterogeneity makes this architecture unsuitable for all but very simple cases.

3. **Federated Database Systems:** These systems attempt to provide interoperation by tailoring different shared elements of the schema to specific classes of user. The sharing can be either tightly coupled or loosely coupled.

We then discuss the tightly and loosely coupled approaches to federation by considering in detail two important research projects, viz:

1. The IRO-DB project [BFHK94] as an example of tightly coupled federal architecture applied to the interoperation of object oriented and relational databases.

2. The MDSL project [LMR90] as an example of a loosely coupled federal architecture created using a multidatabase language.

The specific problems associated with heterogeneities in schema and semantics are considered in a parallel report [HDR97].

1 INTRODUCTION

In the early 1960s, database systems were proposed as a solution to the problem of shared access to heterogeneous data files created by multiple applications in a centralised environment. These data files were difficult to manage; they frequently contained duplications, inconsistencies, redundancies, and various types of heterogeneity at both structural and data levels. To overcome these problems, the autonomous files were replaced with a centrally defined database which was under the centralised control of a *database management system* (DBMS). The DBMS acted as a layer of abstraction between the centrally defined model of the relevant organisation's data requirements, and the applications which used or manipulated the data.

Today, many independent databases exist, particularly in large organisations and/or those which may have undergone substantial commercial or structural changes such as mergers and take-overs. It is often the case that different business units within the same organisational context capture and store the same data, related data, or even the same data viewed from different perspectives. These databases often serve critical functions and embody significant resource investments within their business units. Thus, in many cases the preservation of the environments of these databases is essential, whilst on the other hand, as the information requirements and sophistication of the users evolve, a clear need to share or integrate data at a global level can be identified. The above scenario demonstrates the need to facilitate database interoperation within the context of an organisation. There is also a need to provide interoperation on an inter-organisational basis, particularly in the domain of public information systems. An example is databases specifically intended to market an organisation's products or services to the public.

Consider, for example, the case where airline companies provide public information systems documenting their schedules, ticket costs, flight departure airports and so on. In the absence of interoperation a prospective ticket buyer looking for the best deal would have to query each airline's database separately, and then manually compare the individual results. What this situation requires is a facility to execute database queries which involve the access to data from more than one independent database system, or in short, queries which involve database interoperation.

In this paper we discuss the major issues in the field of multidatabase systems. A taxonomy of interoperable database systems is presented along with a brief discussion of each of the solutions identified. Federated database systems, which offer the most promising solution to the problem of providing interoperation without sacrificing component database autonomy, are examined in some detail along with appropriate examples. Finally, we identify some of the problems which will form the basis of future research in this area.

2 ISSUES OF MULTIDATABASE SYSTEMS

In this section we consider the three major issues which raise themselves in multidatabase systems, namely: distribution, autonomy and heterogeneity [SL90]. We present these issues in

general terms here and then present their specific impact on the various categories of multidatabase systems in the subsequent sections of this paper.

2.1 Distribution

The most obvious issue concerning multidatabase systems is that the data are physically distributed over a number of component databases. The reason for this distribution may be due to the component databases being legacy (or pre-existing) systems, or because the data have been deliberately distributed, fragmented or replicated over a number of component databases. The reason for deliberately distributing data may be to increase system performance and/or model naturally the decentralised nature of the organisation concerned. Whatever the reason, designers of systems which facilitate database interoperation attempt to hide from the global user the physical location of data to some extent. This is referred to as *distribution transparency.*

A second issue worth considering when dealing with distributed data is that of how to specify and enforce inter-database constraints. At the local DBMS level, the ideal is that constraints are specified as part of the database schema, but more often than not they are specified as part of the application code. When used correctly, these constraints provide the local user with a consistent view of data. In environments which consist of a number of independent databases which manage related data, the extension of this facility to provide constraints across the boundaries of database systems is often desirable, or in some cases even imperative. The problem is how to extend the technique from the local to the global level. Leaving the task of specifying inter-database constraints to the developers of global applications is unacceptable in many cases for two reasons: (i) the task of specifying constraints as part of the application code at the local level is a burden on the developer. Expecting the developer to be aware of, and maintain all the constraints at the global level is liable to introduce mistakes and omissions, and (ii) each independent database should have control over its own data, and therefore the inter-database constraints associated with that data. Relinquishing the enforcement of constraints to third party application developers at the global level is not likely to be acceptable. Specifying the inter-database constraints as part of the database schema at the local level directly is also not an option due to the requirement that global activities should not interfere with local ones. However, it is possible to augment the local schema with a separate schema for inter-database constraints which is maintained at the global level, and is therefore invisible to local users and applications. This is the approach advocated by Rusinkiewicz *et al* [RSK91] where dependencies are specified in a declarative fashion and are treated as separate to the local schemas. They employ a five-tuple 'data dependency descriptor' which, in addition to specifying the source and target data objects in the relationship, also includes a predicate for testing consistency, the time boundaries by which the constraint must be satisfied, and a procedure for restoring the consistency of the relevant data objects.

2.2 Autonomy

As indicated above, not only is there a need to interoperate independent database systems, but there is also a primary need in many circumstances to achieve interoperation without sacrificing the independence of the component database systems. This need for independence

is termed *component autonomy*. The autonomy property allows each component database system to control the access to its data by foreign systems, and allows the *Database Administrators* (DBAs), users and applications at the local level to proceed in a manner unaffected by any activities which are taking place at the global level. Local database systems may exhibit four different types of autonomy (Sheth & Larson [SL90]). These are:

- ***Design Autonomy***. This is often regarded as the most critical category of component autonomy because it refers to the ability of the developers and administrators of component database systems to make their own design choice in any situation. In particular:

  (i) how the real world is perceived and what real world entities are to be modelled in the database,

  (ii)the choice of data model, query language, transaction management policies and concurrency control policies,

  (iii)the semantic interpretation of the data objects, i.e. what assumptions are made by the users and applications of the database when 'casting' a model world data object to the concept it is intended to represent in the real world,

  (iv)what database constraints are specified (e.g. uniqueness, inclusion and semantic integrity constraints),

  (v)what operations the database system supports, and

  (vi)what low level implementation strategies are adopted (e.g. file structures and indexing mechanisms).

- ***Communication Autonomy***. This refers to the ability of a component DBMS to decide whether or not, and to what extent, to respond to a request from a foreign DBMS.

- ***Execution Autonomy***. This refers to the ability of a component DBMS to execute local commands or transactions without any interference from operations initiated at the global level. This means that a component DBMS with execution autonomy can abort any operation which fails to meet its constraints criteria. With respect to the ordering of database operations, execution autonomy implies that the DBMS may not have an ordering imposed on it by the global level and also, need not inform the global level of any execution ordering it decides upon.

- ***Association Autonomy***. This refers to the ability of a component DBMS to decide whether, and to what extent, to share its operations and data with foreign DBMSs. This category of autonomy also includes the ability of a DBMS to disassociate itself from the multidatabase system at any time, and the ability to participate in any number of other interoperable systems.

Design and association autonomy are static properties of a component database system (Schaller et al [SBEL93]); they concern static issues which are considered and decided upon prior to interoperation. On the other hand, communication and execution autonomy are dynamic properties which enable the component to exercise run-time decisions. In the general case, the static aspects of component autonomy must be fully preserved; although it may be the case that in specific situations the members of the interoperable system may negotiate agreements which relax some requirements to facilitate better interoperation. In contrast, due to the dynamic nature of communication and execution autonomy, there are run-time situations in which it is desirable that they are compromised. For instance, relaxing execution autonomy in the sense that components are required to inform the global system of their operation ordering would ease somewhat the difficult problem of global transaction management.

2.3 Heterogeneity

The problem of heterogeneity in interoperable database systems is a consequence of design autonomy due to the diverse design choices available to system designers at all levels. We classify four different categories of heterogeneity which occur during database interoperation, namely:

(i) systems heterogeneity,

(ii) syntactic heterogeneity,

(iii) schematic heterogeneity,

(iv) semantic heterogeneity.

• ***Systems Heterogeneity*** concerns the differences in the low level architectural platforms component database that systems employ, such as hardware configuration, operating systems and communications facilities. For example, one database system may be implemented upon a Unix platform, whilst a second system may be implemented on top of the VMS operating system. Also at this level are heterogeneities which are a consequence of the different systems techniques used by different DBMSs such as query processing strategies, concurrency control mechanisms and transaction management facilities. Past research in the area of systems heterogeneity has dealt with the problem successfully by providing an adapter for each type of component configuration which enables the component to converse with the multidatabase system software.

• ***Syntactic Heterogeneity*** is due to the syntactical differences between data models (or in other words, differences in the facilities available to express and manipulate data) which are provided by the component DBMSs. Examples of heterogeneities which exist at the syntactic level are:

(i) differences in the data structure primitives provided by the data models (e.g. relations versus record types versus classes),

(ii) differences in the mechanisms used to express database constraints,

(iii) differences in the query languages provided by the DBMSs.

The issues of syntactical heterogeneity have been resolved by providing a canonical (or common) data model at the global level. Each component is then augmented with a translating process (Sheth & Larson [SL90]) which serves the purpose of translating between the data model at the local level and the canonical data model at the global level.

• ***Schematic Heterogeneity*** is the next level of heterogeneity in interoperable database systems. Each database system wishing to participate in a multidatabase environment must translate its schema into an equivalent schema expressed in the elected canonical data model (as discussed above). The canonical data model must be rich enough to be able to model all possible local database schemas. Recent research has advocated the adoption of semantically richer, object oriented data models for this purpose (Saltor *et al* [SCGS91]). Because the canonical data model must be rich enough to model all eventualities, there will often be a number of ways to model some single concept. As an example consider two schemas which model a many-to-many relationship between two types using an object data model. The relationship may be captured directly by the two types in the first schema (i.e. each type explicitly models a set containing a pointer to each related instance of the other type). Alternatively, the relationship may be modelled in a second schema by explicitly using a third type (in a similar manner that an explicit relation is used to model a many-to-many relationship in the relational data model). Both these alternatives model the same real world concept in a structurally different manner. Schematic heterogeneity concerns this situation where equivalent or related data concepts are present in different database systems and have conflicting structural representations.

Resolving schematic heterogeneities does not present too much of a problem: integrated schemas which model every data concept once and once only may be formed by restructuring and merging component schemas (Batini *et al* [BLN86]). The major problem here is one of detecting which structurally conflicting data concepts actually represent the same real world concept. What is required in this situation is an ability explicitly to capture and reason with data semantics independently of the data's structural properties. Data semantics, in this context, refers to the meaning, interpretation and usage of data concepts, not their schematic properties. Unfortunately, current database schemas model mainly the structural attributes of data, the semantics for a significant part being embedded in the database applications and the user's perception of the database. It should also be noted that it will often be the case that equivalent or related concepts will be modelled structurally equivalently, or as is more likely to be the case, structurally similar enough, to be recognised. There has been much research to date in this area of recognising data concepts based on their structural resemblance (Larson *et al* [LNE89]), and there has also been work on providing some degree of automated support for this process (Sheth *et al* [SLCN88]; Sheth *et al* [SGN93]).

• ***Semantic Heterogeneity*** is the final level of heterogeneity in interoperable database systems. Semantic conflicts arise between related data concepts due to differences in their meaning, interpretation or use. For example, consider two stock market databases, each of which records the price of shares, the first of which is situated in London, whilst the second is operated by a company sited in New York. It is reasonable to expect that the London database records its share prices in Sterling and the New York database in US Dollars. But these facts are not represented as part of the database schemas: they are merely how the users interpret the data. The structural properties of both data items may be similar, but clearly any query which compares the prices of both databases will be erroneous. This example is not particularly difficult to resolve - a conversion function may be employed to translate between currencies. The difficult problem lies in actually detecting the conflict.

Semantic heterogeneity in interoperable database systems presents a difficult problem. The reason for this is the implicit nature of data semantics, i.e. the meanings, interpretations and uses reside in implicit assumptions made by the users and applications of database concerned. An example of a resolvable semantic conflict was presented above. Unfortunately many semantic conflicts exist whose nature does not permit such a simple resolution strategy.

Consider two travel agency database systems which record the prices of holiday packages. One system may update the holiday prices as changes occur in the real world whilst the other updates all prices at the close of the day. A query involving both of these database systems to discover the cheaper price offered for the same holiday package will return an answer consistent with the current database states. However, due to the different semantics of the two price attributes this answer may not be what the user thinks s/he has obtained. There is no simple conversion function that may be applied here, but it may be argued that at least some attempt should be made to explain the conflicting semantics. Thus, the system would still provide the same query answer but with a caveat that the price attributes of the two databases have different semantics.

We argue that the general solution to the problem of semantic heterogeneity lies in the explicit capture and representation of data semantics as part of database schemas. This itself presents a problem because, to date, there is no model which sufficiently captures the extent and nature of the issues concerning semantic heterogeneity. Thus, we further argue that the most promising path to solving this problem lies in understanding and modelling the real world problems first.

The categories of heterogeneity we presented above explicitly separated schematic heterogeneity and semantic heterogeneity. Much of the research work in this area to date has used either of the two terms to include both concepts (Sheth & Larson [SL90]; Kim & Seo [KS91]). We believe that the way forward in this field lies in this separation. The main advantage of this separation lies in the ability to reason about data semantics regardless of possibly conflicting implementation structures [HDR97].

However, the distinction between what constitutes structure and what constitutes semantics is often not clear. One reason for this is that current data modelling tools have an ability to capture a degree of data semantics as part of database schemas. One way of dealing with this

is to adopt a policy where all conflicts between concepts which arise due to a diversity in the schema modelling primitives provided are considered schematic conflicts, and anything which is not covered by schema primitives is considered a semantic conflict. Clearly, this leads to a situation where there is not a clear separation between structure and semantics, and therefore a reduction in the advantage of being able to reason with data semantics independently of structure.

A second method of dealing with this problem is to represent all the semantic issues covered by the database schema again at some semantic level. This will result in a redundancy of a portion of the database schema, but will avoid the problem of an unclear separation between structure and semantics and will also retain the full advantage of being able to reason about semantics independently of structure. A full discussion of the issues of schematic and semantic heterogeneity may be found in our report that was prepared in parallel with this [HDR97].

3 A TAXONOMY OF INTEROPERABLE DATABASE SYSTEMS

In this section we present a taxonomy of interoperable database systems and discuss each category in terms of the issues presented previously. The taxonomy is presented below in Figure1.

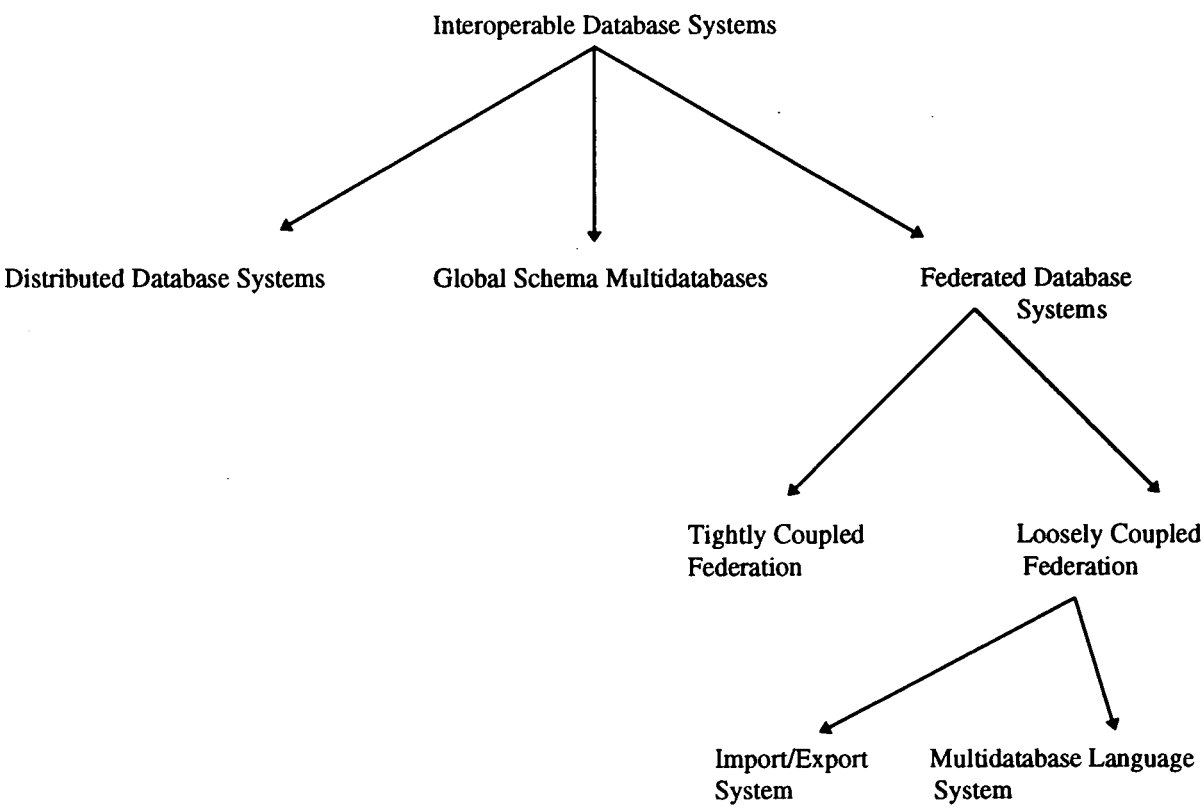

Figure 1: A Taxonomy of Interoperable Database Systems

Past research has introduced a wide range of solutions for sharing information in database systems at a global level. All of these solutions include a global software component which maintains global information and provides some means of access to multiple, physically distributed, local components. The distinguishing element between these systems is in the structure of the global component and how it interacts with the local database systems.

The term **tightly coupled system** is used to indicate a system where global functions have access to the low level internal functions of the local DBMS, whilst on the other hand, the term **loosely coupled system** indicates that the global functions access local functionality only through the local DBMS's external interface (Bright *et al* [BHP92]). However, these are two extremes and it is often more practical to classify systems as being more/less tightly coupled than other (classes of) systems. Systems oriented towards the tightly coupled approach facilitate increased efficiency in global processing with the disadvantage that the local component may not have full control over its resources, i.e. the local system may not be fully autonomous.

3.1 Distributed Database Systems

The distributed database system approach is usually implemented in a top-down manner, with the choices for distribution being based on software engineering criteria. Local and global functionality are implemented simultaneously leading to a system which is very tightly coupled. The local nodes are often homogeneous and present the same interfaces at all levels. Data may be deliberately fragmented or replicated over local nodes in the design stage to give optimal global performance along with the other benefits of decentralisation. With regard to the distribution aspect of interoperable database systems, these types of system provide the global user with distribution transparency by means of a global schema. Global queries are submitted over this schema and the global query processing system maps the appropriate sub-queries to the local levels, and then subsequently combines the local responses to form the global query result. The global system has full control over local functionality. This results in very little autonomy at the local nodes, and in systems that are often optimised towards the needs of the global user. Heterogeneity does not present a problem in distributed database systems for two simple reasons: (i) the local nodes are not autonomous, and (ii) the global system is designed as a whole entity rather than being comprised of a collection of pre-existing systems.

So far in this paper the terms 'multidatabase systems' and 'interoperable database systems' have been used interchangeably. We will now be more precise and exclude distributed database systems from the definition of multidatabase systems due to the non-autonomy of local nodes and the fact that both local and global functionality are so tightly coupled it is often difficult to distinguish between the two. Multidatabase systems are now defined as distributed systems which act as interfaces to, or are structured as global layers on top of, multiple **autonomous** database systems. Although the local layers must perform some limited support for global functionality, the interface between the two levels is at the external interface provided by the local DBMS. With the exception of distributed database systems, the systems presented in the taxonomy (Figure 1) are classified under the term multidatabase system.

3.2 Global Schema Multidatabases

This class of systems attempts to integrate pre-existing database systems whilst largely maintaining their autonomy. As indicated above, this is achieved by interfacing the local and global systems through the external interface of the local level. The distribution aspect is handled by providing a global database schema. However in this case the global schema is much more difficult to produce and maintain due to the pre-existing, autonomous nature of the components, especially when the number of local databases becomes large. The local databases may exhibit all of the four classes autonomy discussed in the previous chapter, although there may be certain situations where it may be desirable to relax some of these requirements. Each of the four levels of heterogeneity is likely to raise a problem in global schema multidatabases, i.e. the systems may be implemented on different hardware and software platforms, they may employ different data models, there may be structural conflicts between equivalent or similar concepts, and there also may be semantic conflicts between related data concepts. Research has found global schema multidatabases lacking as a solution to the problem of providing global access to multiple database systems because the autonomous, heterogeneous nature of the component systems makes the task of producing and maintaining a single, all-encompassing global schema too difficult in all but trivial cases. Examples of global schema multidatabase projects are Preci* (Deen *et al* [DAOT85]) and Multibase (Landers and Rosenberg [LR82]).

3.3 Federated Database Systems

More recent research has focused on the federal database systems approach (Sheth & Larson [SL90]), which attempts to provide controlled and co-ordinated manipulation of a collection (the federation) of co-operating heterogeneous database systems. These co-operating database systems are autonomous and the global system has only a small knowledge of their operations. There is a clear distinction between users and transactions at the local level and users and transactions at the global level. Local users interact directly with the component system and are therefore unaware of the existence of the federation. On the other hand, global users issue global transactions at the federal level and these transactions are then mapped onto possibly multiple components by the global query processing system. However, the component database system itself does not distinguish between local and global operations; the issue is that there are two ways these operations may be generated - either directly from a local user or application or by the federal system. Federal database systems can be broadly divided into two architectural categories: tightly coupled and loosely coupled architectures.

A federation represents database co-operation with a purpose: a tightly coupled architecture attempts to integrate the data concepts present in the component database systems which are pertinent to the identified co-operation. These integrated units are termed federal schemas (Sheth & Larson [SL90]), and most systems of this type are likely to support multiple federations, each of which can be applicable to a particular set of users. Some authors (Sheth & Larson [SL90]; Schaller *et al* [SBEL93]) make a distinction between multiple federation and single federation architectures. In this case the idea is a single federation architecture is equivalent to global schema multidatabases in the above taxonomy. We believe distinguishing between federal and global schema systems is helpful because of the different philosophies surrounding the two approaches: global schema systems attempted to provide sharing by

providing a single all-encompassing schema, whilst federal systems recognised that this is not likely to be possible and instead concentrated on providing multiple integrated units specifically tailored to the needs of a class of user/application. Federal systems therefore overcome the schema complexity problems associated with the global schema multidatabase approach simply because a federal schema is intended to integrate narrow domains of data rather than being an all-encompassing model of the data accessible from the global level. The appropriate database administrators co-operate to generate the federal schemas which are then considered static entities in the sense that the individual users are not allowed to customise their content. Users are again provided with distribution transparency due to the presence of possibly multiple integrated schemas. An important issue concerning this class of system is the schema integration process because this is where the problems of schematic and semantic heterogeneity require addressing; this will be examined in more detail in a later section of this paper.

In contrast to tightly coupled architectures, the distinguishing characteristic of loosely coupled architectures is that it is the individual user's responsibility to create and maintain a federation and no control is enforced by the administrators of the federal system. Thus, by default a loosely coupled architecture supports multiple federations. Two well-known loosely coupled approaches are documented in the literature: Multidatabase language systems (Litwin *et al* [LMR90]) and Import/Export schema systems (Heimbignor and McLeod [85]). Both of these approaches are discussed below.

Multidatabase language systems view a federation as a named collection of autonomous databases. A central feature to this approach is a data language that supports the declaration and manipulation of the constituent databases (Litwin & Abdellatif [LA87]). Consequently, this approach has a somewhat different mechanism for dealing with distribution from the one described above. The user is directly aware of the existence of a number of databases, but their exact physical locations remain transparent. (This is sometimes referred to as location transparency.) The autonomous nature of the component databases again raises the problems of heterogeneity. Furthermore, this category of architecture provides the user with no shielding from schematic and semantic conflicts whatsoever, the onus is entirely on the user to recognise these conflicts and then take some action accordingly. However, the multidatabase language is specifically augmented with extensions designed to help the user resolve these conflicts, although no support is provided by the system to actually detect conflicts in the first instance. Again these issues will be discussed in the following section.

An import/export schema system allows a user to produce integrated schemas by importing a foreign data concept which some other component has exported. The system provides some in-built functionality to facilitate the import and export activities of a user. In addition, a set of derivation operations is provided to aid the user in performing the task of unifying an imported data concept with the existing local schema. There are two levels of support for the distribution aspect. Firstly, the user is provided with location transparency similar to the situation above during the importation process, i.e. the user is aware of foreign data objects but need not be concerned with their physical locations. Secondly, once a data concept has been unified into the local schema full distribution transparency is provided as any attempts to access instances of the data concepts are automatically mapped back to the concept's source (the source being the component that originally exported the data concept) by the system. Again, the onus is on the user to recognise and resolve schematic and semantic conflict, in this case during the unification stage.

4 FEDERAL DATABASE SYSTEMS FOR AUTONOMOUS DATA SHARING

The federal database system approach to global data sharing amongst pre-existing systems is widely regarded as the most promising approach for two reasons:

(i) the autonomy of the component systems is supported.

(ii)it avoids the problems associated with a single, all-encompassing global schema.

We now present the two federal approaches in more detail, followed by a comparison.

4.1 The Tightly Coupled Approach

4.1.1 Architecture

Figure 2 depicts a general architecture for tightly coupled federal database systems (Sheth & Larson [SL90]). The following schema components are identified on the five level schema architecture:

- *Local Schema.* A local schema is the conceptual schema of the component DBMS. It is expressed in the data model native to the local system. Thus, syntactic heterogeneity exists at the local level. The federal system need not make any special requirements of, or need not have to make any changes to, the content of the local schema. This means that design autonomy is preserved in the component systems.

- *Component Schema.* A component schema represents the local schema of the component DBMS translated into the chosen canonical data model of the federal system. This level eliminates the problem of syntactic heterogeneity amongst the component DBMSs. Also, any augmentations needed at the local level to better facilitate interoperation can be included in the component schema, thus enriching the local level without sacrificing design autonomy. As part of the schema translation process, mappings from the component schema to the local schema are generated by the federal system. This is essential because any sub-queries submitted to the component schema must be translated to equivalent queries expressed in the local data language, and then mapped onto the local schemas.

- *Export Schema.* The association autonomy property of the local database systems requires that the component should be able to decide to what extent to share its data resources with the other members of a federation. The contents of an export schema represent the subset of the contents of the component schema which the component wishes to share. As the federal system can support multiple federations, a component system can make available multiple export schemas. Each export schema will in this case contain the subset of data which the local database is willing to share with a particular federation. This facility provides the local level with powerful control and management of association autonomy.

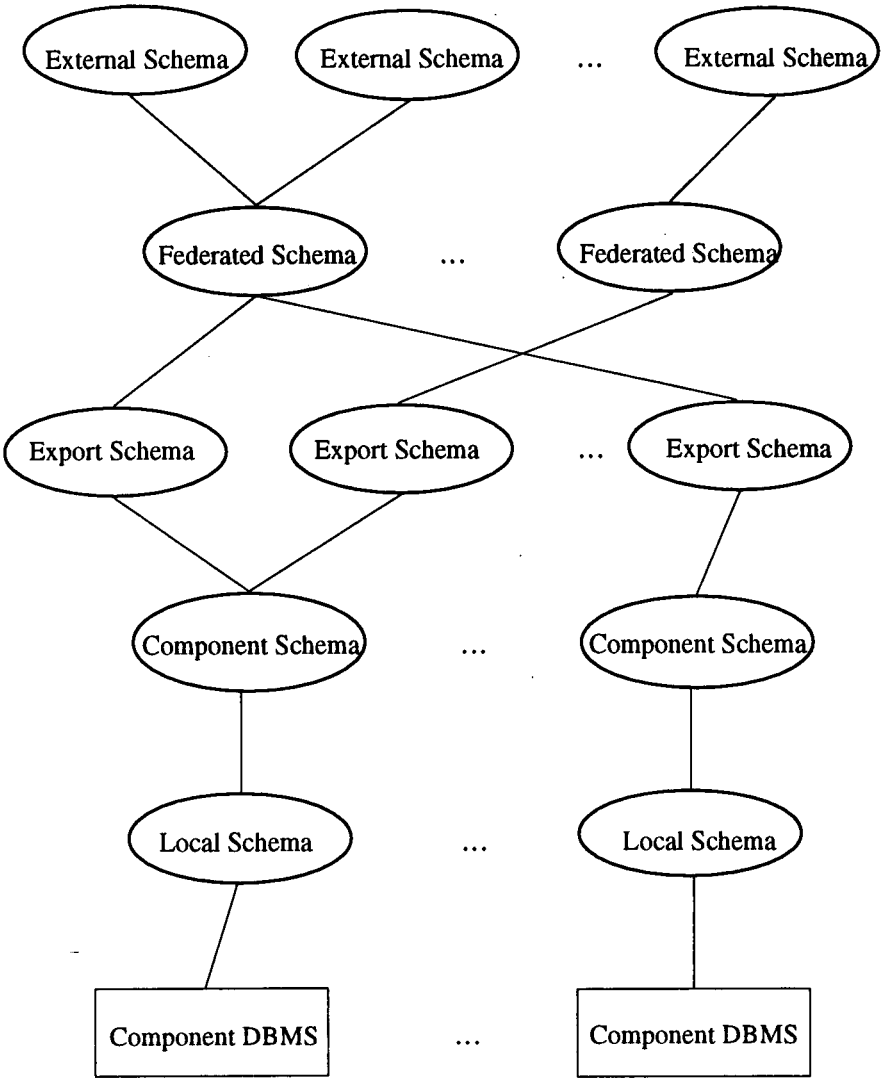

Figure 2: A Generalised Tightly Coupled Architecture (Sheth & Larson [SL90])

• *Federated Schema.* A federated schema represents an integration of multiple export schemas. It is also possible to include information managed at the global level in federated schemas. The integration process performed at this level is a key issue within this architecture because schematic and semantic conflicts must be detected and resolved here to permit efficient interoperation. The users/applications of a particular federation define their queries in terms of the associated federal schema. It is then the responsibility of the federal system to map these queries to the local level and merge any sub-query results accordingly.

• *External Schema.* External schemas in a tightly coupled federal system perform the same sort of function as external views in a centralised database system. The reason for this is that the federal schemas can be relatively complex and cannot be customised by individual users. An external schema can be used to specify a subset of a federated schema which is pertinent to, and is customised towards, the needs of a particular user and/or application. Furthermore, the external schema can be translated into the native data model of the user/application, thereby preventing the user from having to learn the

canonical data model of the federal system. An external schema can also be used to include additional global information, additional integrity constraints, and to enforce additional security.

4.1.2 Schema Integration

As indicated above, the task of integrating export schemas to generate federal schemas is an issue of crucial importance in tightly coupled federal systems. Many methodologies for schema integration have been proposed by researchers, some of which have been specifically developed for integrating schemas of independent database systems, whilst others were developed for the task of view integration in centralised database design. Both of these types of methodology can be applied to database integration with some success. In their survey paper, Batini *et al* [BLN86] compare 12 of these methodologies. They identify five distinct phases that may be present in a schema integration methodology. Each of the methodologies they review perform some, but not necessarily all, of the five identified stages. These stages are:

(i) *Pre-integration.* The component schemas are analysed to decide upon some policy which will govern the integration process. This may include an assignment of preferences to schemas, an order of integration, and adoption of an integration strategy. (For example, a binary strategy integrates two schemas at a time, a one-shot strategy integrates all the schemas in one step, and an iterative strategy attempts to identify closely related sub-schemas and integrate these in a single step.) Additional decisions which may be made at this stage include choosing naming conventions and specifying global constraints. In the context of federal systems, pre-integration also involves the choice of the federal canonical data model.

(ii) *Schema Comparison.* This step involves two stages: (a) the schemas are analysed and compared to detect possible schematic and semantic conflicts, and (b) specifying interrelationships among the concepts in the schemas. The first stage is where problems exist in federal systems because of the autonomous, heterogeneous nature of the component databases. Performing a manual schema comparison stage becomes fraught with difficulties because firstly, the number of input export schemas could be relatively large and of a complex nature, and secondly, as discussed previously the full semantics of a database are unlikely to be captured in the database schemas - additional semantics may be contained in implicit solutions made by the users/applications of the component. The person carrying out the comparison stage is unlikely to have access to all of these semantics; mistakes and misunderstandings may therefore be present in a resulting federal schema.

Because of these reasons, attempts have been made to provide automated tool support for schema comparison or develop theories which may be used as a basis for such tools. The first generation of tools and theories for detecting relationships and conflicts between data concepts defined in independent schemas were based on the structural properties of the data. The main idea being that if structural relationships can be established between the attributes of two entities, then some relationship may hold between the entities themselves. The structural properties of attributes which are analysed are their uniqueness constraints, cardinality, domain, integrity constraints,

security constraints, allowable operations and scale factors (Larson *et al* [LNE89]). The methodologies for establishing relationships between attributes are very formal in nature and typically lead to assertions of the form *x is equivalent to y*, *x includes/is included in y*, and *x is disjoint from y*. Attribute equivalence theories are only subject to semi-automation because it is possible that two unrelated concepts may have equivalent structural properties. In this case the person carrying out the integration process has to reject the relationship proposed by the integration tool and perhaps declare some different relationship to hold.

The major drawback of attribute/entity equivalence is that only structural properties are considered. It is often the case that semantic relationships hold between structurally incompatible data concepts. Thus, more recent research (Sheth & Kashyap [SK92]; Liu & Bukhres [LB93]) has attempted to relate data concepts according to their 'semantic proximity' to one another (or how close they are in some semantic space). This work relies on capturing and representing the *real world semantics* (RWS) of data objects. The primary vehicle for defining RWS is context. An interesting aspect of this work is the recognition that two data concepts can have a closer semantic proximity in one context than another. For instance, the data concepts CARGO_SHIP and SPEEDBOAT are likely to have a closer semantic proximity in the context of an application wishing to discover the average number of engines fitted on boats, than in the context of an application wishing to discover the average size of cargo holds. This work is still very much in the early research stages and a key question yet to be addressed is how to represent and reason with the RWS of data concepts.

(iii) *Schema Conforming.* This step involves attempting to resolve any conflicts detected in the previous stage. Schematic heterogeneities may be resolved by restructuring one or all of the concepts involved. Semantic heterogeneities often have to be resolved by negotiation between the component DBAs and federal system designers.

(iv) *Schema Merging.* The component schemas are now in a form which allows relatively easy merging to form an integrated schema.

(v) *Schema Restructuring.* It may be necessary to restructure the schema derived in the previous stage until it meets certain criteria satisfactorily. These criteria are: (a) the integration should be complete in the sense that it models all of the concepts present in the export schemas correctly, (b) if the same concept is represented in multiple export schemas it should be represented only once in the integrated schema, and (c) the integrated schema should be sufficiently easy for the users to understand.

4.1.3 Canonical Data Models

An important issue to be considered during the design of a tightly coupled federal database system is the choice of canonical data model. Associated with any data model is a measure of its 'representational ability' (Saltor*et al* [SCG91]), this being composed of two factors:

(i) the *expressiveness* of the data model and

(ii) the *semantic relativism* of the data model.

Briefly, the expressiveness factor of a data model relates to the degree to which the model can naturally capture any real world conceptualisation. The semantic relativism relates to the extent to which the model can accommodate different conceptualisations of the same real world. (For example, the relational model supports the derivation of external views to accommodate different perceptions of the same schema.)

Saltor *et al* [SCG91] classify the requirements a model needs to demonstrate in its representational ability for it to be an appropriate canonical data model for a federal system. With regard to the expressiveness factor, the model needs to be sufficiently rich enough to represent naturally all of the existing versions and any future anticipated, enriched versions of component database schemas. It is important to note that the local schemas may undergo an enrichment process prior to being integrated into the federation to capture additional semantics, and this knowledge acquisition process prior to data model translation must be taken into account. With these considerations in mind, Saltor *et al* [SCG91] recommend that a sufficiently expressive canonical data model should support the following abstractions: classification, metaclasses, generalisation/specialisation, multiple inheritance, different classes of specialisation - e.g. disjoint, aggregation, association, and the definition of new behaviour.

With regard to semantic relativism, a candidate data model must support the implementation of integration operators such as *meet*, *fold* and *gen*. These operators can be used to construct super-views of component schemas during the integration process. Also to facilitate the integration process, the model should offer only a single basic structural primitive. (For example, an object model will support one single structuring primitive, the 'object', whilst the entity-relationship model supports two, entities and relationships.) The model must also include facilities for defining views with multiple semantics. This is important in the context of federal systems because users need to be able to define their own interpretations of federal schemas (external schemas in the five-level architecture shown in Figure 2).

Saltor *et al* [SCG91] review current data models against their requirements and find that object models offer the best alternative due to their rich range of modelling abstractions, extensibility and support of a single modelling structure. The major drawback with the relational model is its poor natural support for the more powerful modelling abstractions such as generalisation. Extensions to the basic entity-relationship model offer a wide range of modelling abstractions, however a disadvantage is the fact that the model supports more than one structuring primitive. Also, extended ER models offer no support for modelling behavioural aspects. Functional data models perform well in the semantic relativism aspect, and offer all of the necessary expressiveness characteristics.

4.1.4 The Problem of Schema Evolution

Schema evolution in interoperable databases refers to the ability of the local database schemas to evolve over time. This is a major problem in federal architectures due to the high degree of design autonomy exhibited by the component database systems. The schema integration process in tightly coupled architectures produces integrated federal schemas along with the appropriate mappings from the integrated schema elements to the local schema elements. The

problem is this: as local schemas evolve the mappings between the integrated schemas and local schemas are likely to become inconsistent. Also it may not be possible in some situations to reflect any changes to local schema elements in the integrated schemas. Schema evolution in multidatabase systems is one of the prominent present day and future research issues.

4.1.5 IRO-DB - An Example Tightly Coupled Federal Architecture

The IRO-DB project (Busse *et al* [BFHK94]; Gardarin *et al* [GGF&al95]) is concerned with developing a federation of relational and object oriented DBMSs. IRO-DB supports both loosely coupled and tightly coupled federated architectures, although here we are interested in the tightly coupled case. The IRO-DB system architecture is presented in Figure 3 (Busse *et al* [BFHK94]). As can be seen, the architecture comprises three layers: the *interoperable*, *communication* and *local* layer. The key features and functionalities of these layers are presented below with a more extended summary in Appendix 1.

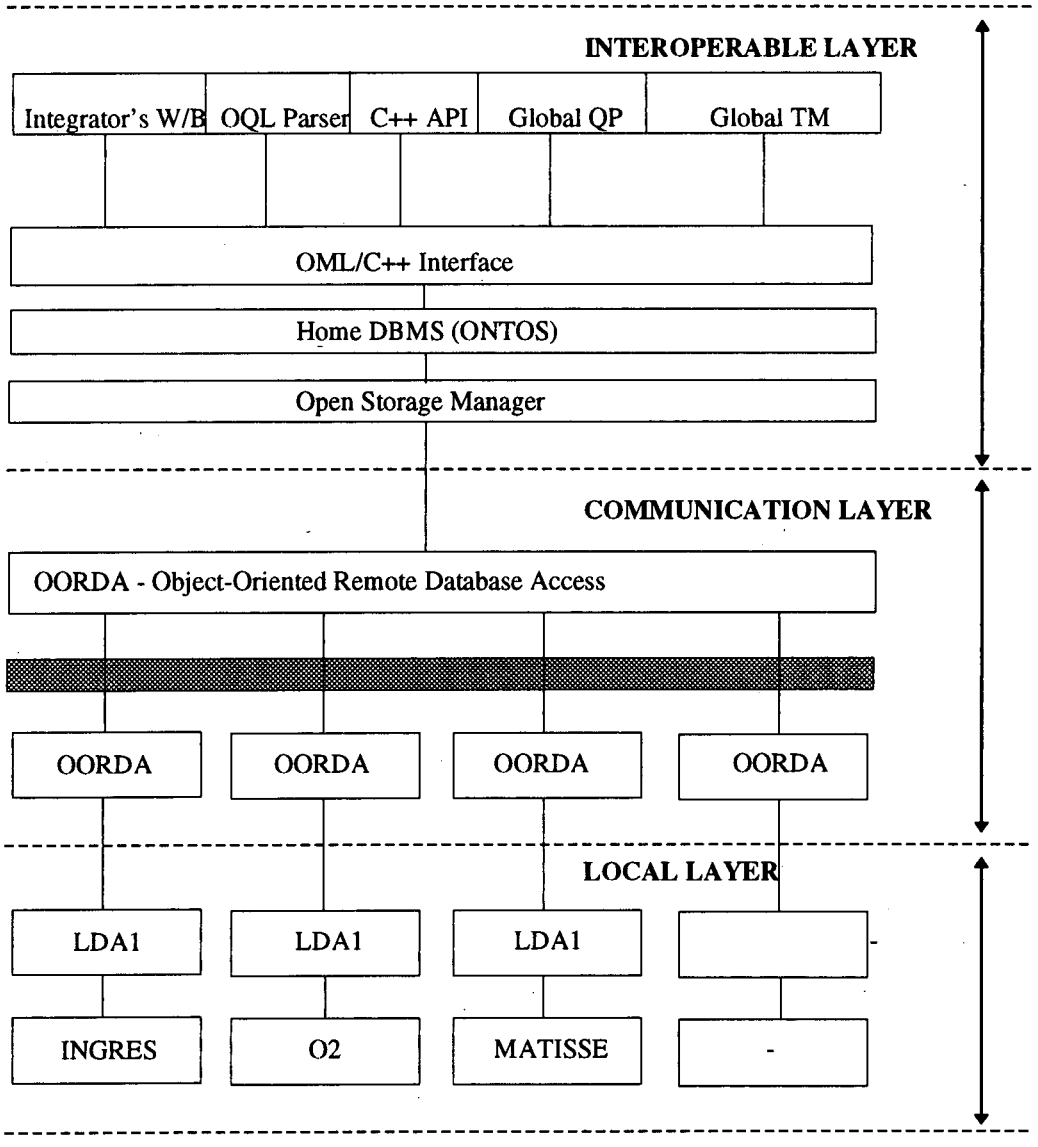

Figure 3: IRO-DB Architecture

The local layer is intended to provide homogeneous access to the component database systems, thus alleviating the problems of syntactic and systems heterogeneity. The local database schema is (partially) translated into a schema expressed in the canonical data model, which in this case is the ODMG-93 (Cattell [C93]) object model for databases (see later). This export schema, together with the appropriate mappings, is used as the basis for translating OQL queries into the appropriate local operations. The communication layer realises the transfer of objects, OQL queries, and query results to and from client and server sites. The communication layer is also responsible for the transfer of export schemas between the local and interoperable layers. The interoperable layer provides the facilities needed to create and manage federated schema, and therefore support for tightly coupled federations.

4.2 The Loosely Coupled Approach

4.2.1 Architecture

Figure 4 presents a reference architecture for loosely coupled federal database systems (Litwin *et al* [LMN90]).

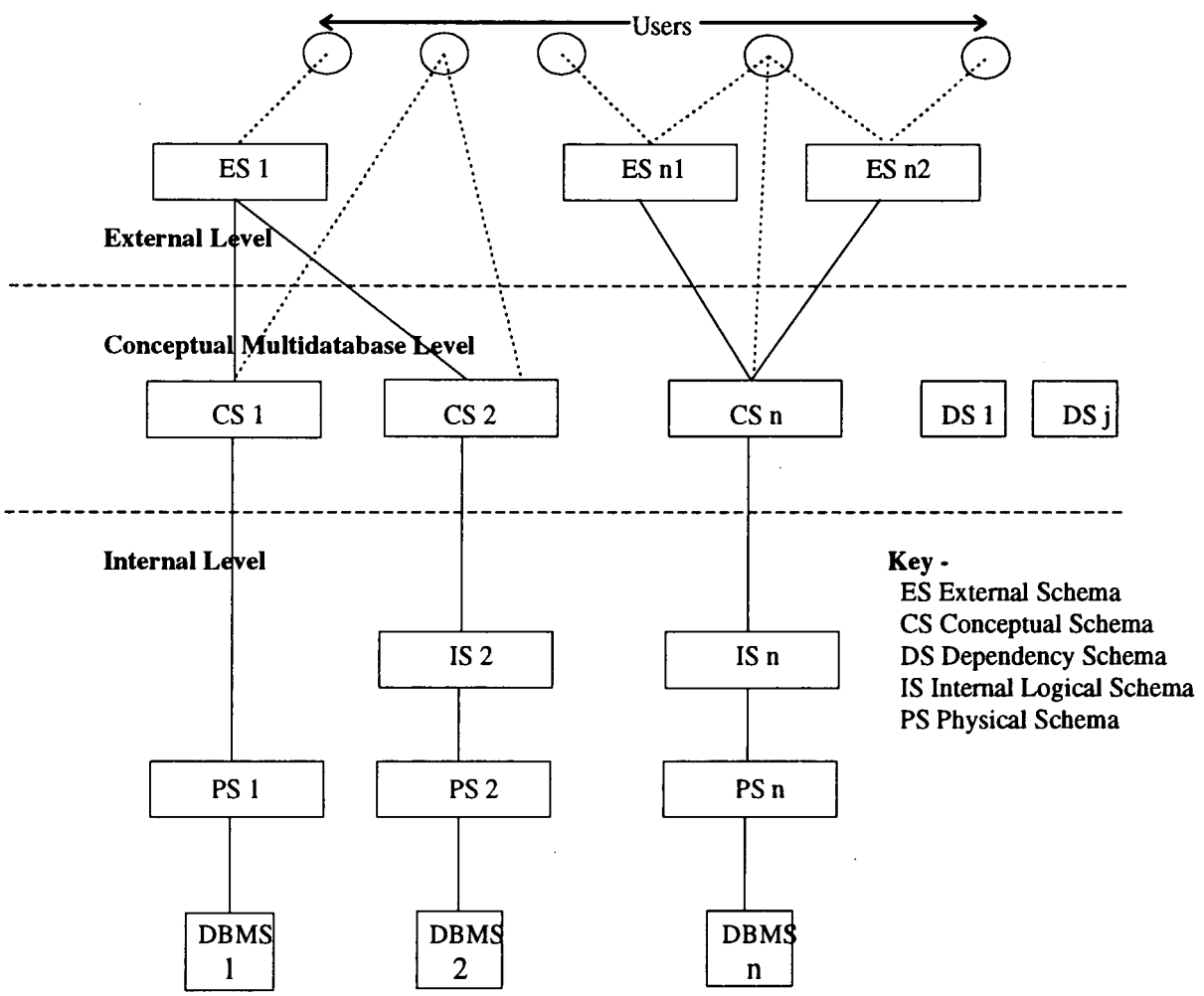

Figure 4: A Loosely Coupled Federation Reference Architecture (Litwin et al[LMN90])

The following layers are shown on the loosely coupled reference architecture:

- *Internal Level.* The internal level consists of the component DBMS and its physical schema.

- *Conceptual Multidatabase Level.* The internal level presents a conceptual schema to the conceptual multidatabase level. As can be seen from Figure 4, DBMS 1 presents this conceptual schema directly. In this case the conceptual schema at the multidatabase level is the actual conceptual schema of the component database, i.e. the component shares its entire data resources with the other users of the federation. To handle the situation where a component only wishes to share a subset of its schema, this subset becomes the content of the conceptual schema at the multidatabase level and the actual conceptual schema of the component is represented by an internal schema at the internal level. (See DBMS 2 and DBMS n in Figure 4.) If a canonical data model is required at the conceptual multidatabase level, as is usually the case, it is the responsibility of the component DBMS to ensure that the conceptual schema it presents to the multidatabase level respects this requirement. In this case, the component must also maintain the translation mappings from the conceptual schema to its local schema.

  The multidatabase level also includes a facility to define dependencies between collections of databases. These are expressed in dependency schemas (Figure 4), and their purpose is to allow the federation administrators to specify inter-database constraints which provide the ability to enforce some element of consistency in the absence of integrated schemas. Typical inter-database dependencies which may be specified in the dependency schema are equivalence of attributes or domains (Litwin *et al* [LMN90]).

- *External Level.* A user may construct external schemas at the external level, external schemas being views of collections of conceptual schemas. Collections of databases may be presented as a single integrated database in an external schema, but this should not be confused with integrated schemas in the tightly coupled federal database approach. In the loosely coupled approach, external schemas are the user's responsibility to create and maintain. No consistency can be guaranteed if conflicts are not resolved or the proper inter-database constraints are not enforced.

It should be noted that architecture presented above is one of a specific multidatabase language system (Litwin *et al* [LMN90]) rather than an explicit attempt to capture the general case. However, we feel that this architecture does capture the general case for loosely coupled federations with some degree of success. For instance, the above architecture and the import/export schema architecture proposed by Heimbigner and McLeod [HM85] are very similar, the major differences being in the tools available to the global user for creating and managing federations, rather than in the schema architecture. Also, in the previous section it was stated that the local layer and the communication layer of the IRO-DB architecture (Figure 3) realised a loosely couple federation. Here the local layers present ODMG export schemas to the communication layer in a similar manner to the way local databases present conceptual schemas to the conceptual multidatabase level in the reference architecture above. A user placed above the communication layer can, with appropriate tools, name and query

collections of databases and even construct integrated views which serve the same purpose as the external schemas in Figure 4.

In Section 3 of this paper we presented a taxonomy of interoperable database systems in which we classified two categories of loosely coupled federal database systems: import/export schema systems and multidatabase language systems. Henceforth we consider multidatabase language systems only, as this is by far the more prominent paradigm.

To return to Figure 4, a user at the global level has two methods of access to the multidatabase system: (i) directly at the conceptual multidatabase level using the multidatabase language, and (ii) through an external view using either the multidatabase language or a usual database query language if the external schema defines a single integrated database (Litwin *et al* [LMN90]). It should also be noted that the tools which can be used to define external schema are embedded in the multidatabase language themselves (in a similar manner to how SQL provides facilities for creating views). Thus, the key element in these systems is the multidatabase language tool. The features of these languages are presented in the following section.

4.2.2 Multidatabase Language Features

The basic function of a multidatabase language is to allow users to define and manipulate a collection of autonomous databases. Traditionally, these languages have been relationally oriented due to the non-procedural nature of relational query languages, the predominance of relational database systems, and the formal origins of the relational model. However, recent research has focused on the use of object oriented query languages for this purpose (e.g. IRO-DB presents users with an OQL interface to remote databases). Whatever paradigm is adopted, there are a number of essential functional requirements a multidatabase language must provide to be suitable for the task at hand. These are (Litwin *et al* [LMN90]):

- The ability explicitly to name collections of databases as multidatabases. These multidatabase names can then be used as the scope for data manipulations which range over the collection of databases.

- The ability to execute, within the context of a single statement, data definition statements at the multidatabase level such as the creation, alteration and removal of data types (relations or objects) in different databases. Also, the language should allow the import of data definitions from remote databases.

- The definition of units, scales and precisions of data values in local schemas. This feature enables the system automatically to make conversions between heterogeneous representations of related data.

- The execution of data retrieval and update operations which access data types in different databases. These are termed elementary queries and are similar in nature to SQL queries, the difference being that elementary queries are qualified with database names to permit the query to access multiple databases. The use of these designators

to qualify data type names with database names is important because it will often be the case that data types defined in different databases will share the same name.

• The execution of multidatabase queries. Multidatabase queries provide the user with the ability to perform in a single statement, an operation which acts on possibly heterogeneous data types present in a named collection of separate databases. The difference between elementary queries and multidatabase queries is as follows: elementary queries are designed to be used in situations where a single query involves access to data defined in different databases; multidatabase queries, on the other hand, are intended for situations where separate databases model the same real world concepts. The user can then use a multidatabase query to 'broadcast' the same intention to several databases. The ability to define a multidatabase as a named collection of databases was described above. One of the primary functions of these multidatabases is to act as the scope for multidatabase queries. Examples of both elementary and multidatabase queries are presented in Appendix 2.

• In relational systems, the provision of implicit joins for manipulations of different databases with similar data which is expressed in relations with different vertical decompositions. This facility allows the same query formulation to be 'broadcast' to databases in which the same information is modelled by different relational structures, by implicitly constructing a common relational structure from each of the different decompositions.

• The provision of *dynamic attributes* which allow a user to transform heterogeneous data values within the scope of a query. They are dynamic in the sense that they are defined within a query, and are therefore attributes that are not defined in any schema. With the exception of updates, they may be manipulated in the scope of the query like any ordinary attribute. Dynamic attributes are likely to be used often in a multidatabase system because users often wish to place their own interpretation upon data, or transform related data items to a common format/unit for the purpose of comparing or joining. This feature will be illustrated in Appendix 2.

• The language must be extensible in the sense that new in-built functions can be defined as needed. Again, this feature will be illustrated in Appendix 2.

• The provision of multidatabase view definition and multidatabase external schema definition facilities. Multidatabase views are virtual data types which are derived from multiple databases and may be employed by users to place their own semantics on data which are of interest to them. A multidatabase external schema defines a virtual database as a collection of virtual relations. These can be oriented towards the needs of a particular class of user/application and, as discussed above, can be used to provide the abstraction of a single integrated database. This can sometimes be of great benefit because they can allow less experienced users to query multiple databases using standard query tools (i.e. query tools designed for single databases).

• Support for 'inter-database queries' which are used to import and export data between multiple databases in a single statement. This facility needs to provide a means of dealing with key conflicts between incoming and existing data instances.

- The language also needs to provide support for defining auxiliary objects such as inter-database constraints and dependencies which exist between data types in different databases.

4.2.3 The Problem of Schema Evolution

In general, users can use loosely coupled federal architectures from two perspectives: (i) as an occasional user wishing to discover some information which is relevant only at the time of the query session, e.g. as in the case of the user trying to find the cheapest price for the same holiday, and (ii) a frequent user who will wish to submit the same multidatabase queries at different times, e.g. in the case of a federation of databases containing information about stocks and shares where a user may wish to submit a query which retrieves the latest prices of his/her various shares every week. In the first situation, schema evolution does not present a problem because the user will typically find out what information is available in the local databases and build the appropriate multidatabase query within the session. However, instead of constructing the same query time after time, users querying the multidatabase frequently will store their queries in compiled form and run them when required. Schema evolution in the local database systems is then likely to make these compiled queries inconsistent. It may be the case that in some circumstances the global system can monitor simple changes to local schemas such as the renaming of attributes. Any compiled queries which reference data objects which have been subject to these simple changes can then be recompiled prior to execution, and the global system can then amend inconsistencies during the compilation phase. However, in the case of more drastic changes to local schemas the compiled queries will simply fail and the user will have to first discover what local changes have been made, and then either amend or re-formulate the query.

4.2.4 MDSL - An Example Multidatabase Language

In Appendix 2 we present a brief overview of the features of the MDSL multidatabase language [LA87] which is the language developed in conjunction with the MRDSM (Multics Relational Data Store Multidatabase) prototype. It must be stressed that this system does not exhibit some of the language characteristics presented in the previous section. In particular, the language does not support the specification of explicitly named multidatabases, does not facilitate the derivation of views or external schemas, and does not provide support for defining inter-database constraints or dependencies. However there are many useful features in the language such as the ability to:

i) open several databases with one command

ii) provide aliases for databases and tables

iii) introduce domains that range over semantically equivalent types in different databases

iv) identify the name of the database of a successfully retrieved row

v) define attributes dynamically using lists, formulae, and code

4.3 A Comparison of Loosely Coupled and Tightly Coupled Architectures

In tightly coupled architectures, the federations are created and managed by the administrators of the system according to some pre-determined policy. For this reason tightly coupled systems are referred to as static federal database systems. On the other hand, loosely coupled systems are referred to as dynamic federal database systems because the individual users are provided with the tools dynamically to create and manage their own federations in the absence of any pre-determined policies. Although the two approaches presented above may appear to have quite different strategies for realising interoperation, they actually perform similar functions. The difference is in when these functions are performed rather than what they are.

In the tightly coupled approach a local database schema is converted to an equivalent representation in the canonical data model, and a subset of this is exported for the use of the federation. Sets of export schemas are then statically integrated to form the federal schemas. These federal schemas are then made available for the use of the multidatabase users. In the loosely coupled approach export schemas contain the subset of a local schema that a component database system is willing to share. (The local schemas may also be converted into a canonical model if necessary.) These export schemas are then made available to the federation users who can dynamically create their own federations using the tools provided which in most cases are multidatabase languages. The difference is that the tightly coupled approach statically integrates *schemas* prior to interoperation, whilst the loosely coupled approach dynamically integrates *data* during interoperation, in the form of a multidatabase manipulation expressed in the multidatabase language.

Sheth & Larson [SL90] report that the loosely coupled approach is better suited for interoperation between large numbers of very autonomous read-only databases such as public information systems, whilst the tightly coupled approach is better suited for corporate interoperation where the extra control provided by the tightly coupled nature of the approach is desirable.

5 CONCLUDING REMARKS

Research has proposed a number of solutions to the problem of providing interoperation amongst independent database systems, and most of these solutions have met with some degree of success. Federal database system architectures are the most promising solutions to date because they cater for high degrees of component database system autonomy. However, there are still many key problems in this area which have failed to be successfully addressed.

Neither federal architecture deals successfully with the problem of schematic heterogeneity. In the case of loosely coupled architectures the tools provided to the user for resolving schematic conflicts are complex. It is unreasonable to expect that a non-expert user who has some familiarity with a single database query language will easily come to terms with the MDSL language discussed in section 4.2. Also, the user of a loosely coupled system is offered poor support for detecting data of interest, particularly if the number of databases participating in the federation is large. Tools may be used to detect data items based on structural resemblance, but related concepts which have diverse schematic representations will be difficult to detect. Tightly coupled architectures rely on a schema integration stage which is

largely manual and dependent on the administrators' knowledge of the schemas. Again, tools are available which detect structural relationships among data items only.

Current federal database architectures offer little support for resolving semantic heterogeneities. The onus is on the user or administrator to detect and resolve semantic conflicts when creating federations. As indicated in section 2.3, a degree of the semantics of a database reside in the implicit assumptions made by the users and administrators. These assumptions are unlikely to extend to the multidatabase level. It is therefore possible that subtle semantic conflicts will be undetected because data items have been interpreted incorrectly during the federation creation stage.

We argue that the solution to both these issues lies in the explicit capture and representation of database semantics. The problem associated with schematic heterogeneity is not one of resolving schematic discrepancies, but rather one of detecting them. The ability to reason about data semantics independently of structure will aid in detecting semantically related data items which have conflicting structural representations. Similarly, the ability to reason with explicitly represented data semantics will facilitate the detection, and resolution in some cases, of semantic heterogeneities. We discuss the issues of schematic and semantic heterogeneity in a subsequent report [HDR97].

APPENDIX 1

INTEROPERABLE RELATIONAL AND OBJECT DATABASES

This appendix contains an extended discussion of that in section 4.1.5 of IRO-DB, an example of a tightly coupled federal architecture based on the object database model.

The IRO-DB project (Busse *et al* [BFHK94]; Gardarin *et al* [GGF&al95]) is concerned with developing a federation of relational and object oriented DBMSs. IRO-DB supports both loosely coupled and tightly coupled federated architectures, although here we are interested in the tightly coupled case. The IRO-DB system architecture was presented in Figure 3 (Busse *et al* [BFHK94]). As can be seen, the architecture comprises three layers: the *interoperable*, *communication* and *local* layer. The key features and functionalities of these layers are presented below in more detail.

The **local layer** is intended to provide homogeneous access to the component database systems, thus alleviating the problems of syntactic and systems heterogeneity. The local database schema is (partially) translated into a schema expressed in the canonical data model, which in this case is the ODMG-93 (Cattell [C93]) object model for databases (see later). This export schema, together with the appropriate mappings, is used as the basis for translating OQL queries into the appropriate local operations. This functionality is supported through providing the local level with a *local database adapter* (LDA). These adapters consist of two major levels: a generic part which is common to all adapters and a specific part which is designed for each particular DBMS. The key advantage of this approach is that the local layer provides a 'standard local access point' (Busse *et al* [BFHK94]) where each local database can be accessed in an ODMG compliant manner. This means that the users at the local level are able to issue their queries in the OQL data language as an alternative to the data language of the local DBMS. The generic functionality of each LDA consists of the following:

- The provision of an 'export schema repository' which serves as a data dictionary for the ODMG representation of the shared portion of the local database schema.

- The provision of a 'generic mapping repository' which contains the information required to convert between the local schema and the ODMG compliant export schema. Note that this repository contains only a generic description of the mappings, the actual implementations being provided by the specific part of the LDA.

- An 'export schema manager' is provided which is responsible for the management of both of the above repositories.

- An OQL parser which provides the external interface to the LDA and, with the aid of the above repositories, translates OQL into a syntactic query tree.

The specific functionality provided by each LDA is as follows:

- A 'mapping toolbox' is used to implement the routines for converting between local data types and ODMG data types.

- A 'local mapping adapter' is used to transform the syntactic query trees provided by the OQL parser into local queries expressed in the query language of the local database.

Notice that the elements of the local layer are covered by three schema levels in the generic tightly coupled architecture (Sheth & Larson [SL90]) presented above, whilst only two schema levels are used by IRO-DB. The reason for this is that the IRO-DB architecture does not convert the whole of the local schema into the canonical data model, but instead converts only the portion it wishes to share into the export schema in a single step. The Sheth & Larson [SL90] architecture uses two steps to perform this: the local schema is fully translated into a representation in the canonical data model (a component schema), and then the sharable portion of this is then expressed in export schemas. A second difference between the two architectures is that the Sheth & Larson [SL90] architecture includes the ability to express multiple different export schema, each of which contains the information the component system wishes to share to a particular class of user/application. This facility is not present in the IRO-DB architecture. Of course, different views may be expressed over the export schema by different users/applications, but the difference is that what these different users/applications have access to is no longer under the control of the component system.

The **communication layer** realises the transfer of objects, OQL queries, and query results to and from client and server sites. The communication layer is also responsible for the transfer of export schemas between the local and interoperable layers. For further details of the services provided by the communication layer, along with its architecture and details of the standards embraced during its design, the reader is referred to Busse *et al* [BFHK94] and Gardarin *et al* [GGF&al95]. It is worth noting that the local layer and communication layer provide the ability to realise loosely coupled federations, where the data has to be accessed and integrated explicitly by the individual users.

The **interoperable layer** provides the facilities needed to create and manage federated schema, and therefore support for tightly coupled federations. As discussed above, the designers of IRO-DB have adopted the ODMG-93 object model for the canonical data model. The primary reason for this is that the designers see a benefit in using this upcoming standard because there is a foreseeable availability of ODMG compliant interfaces for many commercially available databases (Busse *et al* [BFN94]). The suite of tools provided by the interoperable layer (Figure 3) consists of two categories. One component is named the integrator's workbench. Its purpose is intended to assist in the design and maintenance of interoperable schemas. These interoperable or federated schemas are formed by the integration of numerous (partial) export schemas. The second category of tool spans the rest of the components shown on the top-level of the interoperable layer (Figure 3). Their intended purpose is to support the users of the federated system as outlined below:

- *OQL Parser.* This parses a global query expressed in OQL against an interoperable schema and generates an *object expression tree* (OET).

- *Global QP.* The global query processor uses the mapping information in the global data repository (the home DBMS) to break the query expressions in the OET, which refer to an interoperable schema, into sub-expressions which refer to the export

schemas at the local sites. This transformed OET is optimised and passed to an execution component that sends the OQL sub-queries through the communication layer to the appropriate LDAs and merges the relevant query results to form a global result.

• *Global TM.* The global transaction manager implements a nested transaction control protocol as proposed by the ODMG model. Information provided by global repositories and the transaction management services provided by the communication layer are utilised to assist in the management of global transactions.

• *C++ API.* This provides an ODMG compliant C++ application programming interface which can be used by applications to access the interoperable schemas and their data instances.

The integrator's workbench is an interoperable layer tool which is used to create and maintain integrated schema. It is able automatically to generate a federated schema using path correspondence assertions declared by the user and algorithms which automatically detect correspondences between schema elements. The home DBMS, which should be ODMG compliant to guarantee portability of the interoperable layer, stores and provides: (i) access to the copies of local database's export schemas, (ii) access to the interoperable schemas and associated mapping information, (iii) access to the information concerning where export schemas originate from, and (iv) any other information which needs to be maintained at the global level. In addition, the home DBMS is used to store intermediate query and sub-query results which are waiting to be composed into a global result.

Busse *et al* [BFN94] classify the different types of object classes which may appear in a federated schema. These are:

(i) *external classes* which are the existing classes at the local node which are made available to the federal system via the export schema. They are physically bound to the local node and have no instances on the global node;

(ii) *imported classes* which are (partial) one-to-one copies of external classes at the global node (i.e. they represent the copies of the export schemas at the interoperable level). They exist on the global node, but their instances reflect only the external instances of the corresponding external class;

(iii) *derived classes* that exist in integrated schema and are derived from imported classes in one or more steps to facilitate interoperation. A single derived class may be derived from one or more imported classes. As indicated above, the home DBMS stores the imported class specifications, the derived class specifications (or interoperable schemas) and the mappings between the two; and

(iv) *standard classes* that represent the information maintained at the global level that is not imported from local databases, i.e. classes which are created and have their instances at the global node.

Classes of type (ii) and (iii) are termed *virtual classes.* They behave like normal classes and consist of an interface and an implementation, but unlike ordinary classes, the implementation

derives its instances from the instances of other classes. It is the responsibility of the remaining component of the IRO-DB architecture, the Open Storage Manager, to define a virtual class on the home DBMS for every imported class passed through the communication layer. These virtual classes are identical to the corresponding classes defined in the relevant export schemas with the exception that the access methods are overloaded with methods which retrieve the instances from the remote sites. Thus, the open storage manager ensures that the users/applications of the home DBMS can access imported classes in a distribution transparent manner.

APPENDIX 2

MDSL - AN EXAMPLE MULTIDATABASE LANGUAGE

In this appendix we present a brief overview of the features of the MDSL multidatabase language (Litwin & Abdellatif [LA87]) which is the language developed in conjunction with the MRDSM (Multics Relational Data Store Multidatabase) prototype. It must be stressed that this system does not exhibit some of the language characteristics presented in Section 4.2.2. In particular, the language does not support the specification of explicitly named multidatabases, does not facilitate the derivation of views or external schemas, and does not provide support for defining inter-database constraints or dependencies. The purpose behind its presentation here is not to introduce a language which successfully meets the above criteria, but rather to demonstrate some of the facilities such a language would need to provide to support multidatabase queries. Figure 5 presents the export schemas of the four databases we will use in the examples presented in this section.

```
Manchester:
      Book(ISBN, Title, Author, Cost, Category, Rating)

New_York:
      Books(Title, Author, ISBN, Price, Pub_Date, P#)
      Publishers(P#, Pname, Paddress)

London:
      Book(ISBN, Author, Title, Price, Type, Date)

Glasgow:
      BookItems(ISBN, Title, Price, Author, Category, Rating)
```

Figure 5: Export Schemas of Book Shop Databases

The schemas shown in Figure 5 represent the database export schemas of four independent book shops. For simplicity, the actual databases are named after the cities where the book shops are based. Thus, there is a database named 'Manchester' with a 'Book' relation, a database named 'New_York', and so on. The databases store information concerning the same universe of discourse, this in fact being the books the shops stock. Notice that some of the relations concerning books have attributes that are missing in others. For example, the Manchester database has an attribute *Rating*, which corresponds to a rating assigned to books in accordance with their reviews. This attribute is not present in the London database. Also, the attributes *Category* in the Manchester and Glasgow databases, and the attribute *Type* in the London database represent the same concept, this being the subject a book may be classified such as History, Computing or Cinema. The more subtle discrepancies between these schemas will be presented below along with the examples that introduce the language features which are intended to resolve them.

Example 1: what are the book categories common to both the Manchester and Glasgow databases ?

The example serves two purposes: (i) to present the basic structure of a MDSL query, and (ii) to demonstrate the concept of an 'elementary query' (discussed above). The MDSL query formulation of this example is:

```
open Manchester r Glasgow r
-db (M Manchester) (G Glasgow)
-range (x M.Book) (y G.BookItems)
-select x.Category
-where x.Category = y.Category
retrieve
close M G
```

The *open* command opens the named databases for processing, the *r* option specifying read-only mode. The *close* command is only necessary if the databases opened are not needed for further queries. The *-db* clause serves two purposes: (i) it can be used to specify aliases for databases, thus making the rest of the query easier to express and understand, and (ii) it makes it possible to define the set of databases the query should refer to (some databases may have been left open by previous queries), the default being all of the databases open at the time in question. The *-range (tuple_variable relation(s))* clause is used to define explicitly semantic variables whose range is the named relation(s). (A key element in expressing multidatabase queries is declaring variables which range over more than one relation - see later.) The *-select* and *-where* clauses have the same semantics have as the corresponding clauses in SQL. Finally, the query command specifies the action to be taken; the options are *retrieve, modify, store, delete, copy, move* and *replace*. This query is relatively simple: it opens both of the required databases, declares an alias and a tuple variable for each, and specifies the selection clause. The interesting aspect is the use of designators to distinguish between *Book.Category* and *BookItems.Category* in the context of the query. This ability to uniquely identify similarly named data items defined in different databases is the defining characteristic of elementary queries.

Example 2: from the London and Manchester databases, retrieve the books whose subject is Modern Languages.

The fundamental purpose of this example is to demonstrate the technique employed in MDSL to formulate multidatabase queries. In addition to this, as the attribute *Category* in the Manchester database models the same aspect of the real world as the attribute *Type* in the London database, this query introduces a *Semantic Variable*. A semantic variable is a variable whose domain is data type names, and they are typically used to resolve naming heterogeneities such as the one above. The use of a semantic variable indicates that the query concerns each of the data types in its domain. Consider the query formulation for example 2:

```
open Manchester r London r
-range (x Book)
-range_s (cat Category Type)
-select x
-where (x.cat = "Modern Languages")
retrieve
```

Firstly, the relation name *Book* in the second line of the query is known as a *Multiple Identifier* because both of the databases involved in the query contain a relation with this name. Multiple identifiers in MDSL are used to express multidatabase queries; in this case the tuple variable *x* ranges over both *Manchester.Book* and *London.Book*. Secondly, the third line

in this query defines the semantic variable *cat*, whose domain is the data types *Manchester.Book.Category* and *London.Book.Type*. The *where* clause in the query then refers to *x.cat*, thus resolving the heterogeneous attribute names present in the two relations. Presented below is a second multidatabase query, in this case a semantic variable is used to resolve naming heterogeneities between the relations the query is concerned with.

Example 3: from any of the databases, find the details of the book "The Stand" written by the author Stephen King.

```
open Manchester r Glasgow r London r New_York r
-range_s (B Book Books BookItems)
-range (x B)
-select x.Name(.B) x
-where x.Author = "Stephen King"
 & x.Title = "The Stand"
retrieve
```

The only aspect of this query that has not been introduced before is the use of the in-built function *Name* in the select clause. This tags each tuple with the name of the database from which it originated. This is useful in contexts like the above query because the user may decide to order the book from the shop offering the most reasonable price, and in the absence of the name function there will be no direct means of distinguishing which database a particular tuple originates from. This demonstrates the value of being able to extend a multidatabase language with new in-built functions.

Example 4: from the New York database, retrieve the publishers of any books written by the author Stephen King.

This query demonstrates how join clauses may be omitted from query formulations. These joins are termed *implicit* joins and serve two purposes: (i) they facilitate simpler formulation of queries, and (ii) they allow multidatabase queries to be expressed to databases whose relations model the same real world, but have different vertical decompositions. These joins are then deduced by the system from the database schemas. Consider the query formulation of example 4:

```
open New_York r
-select Pname
-where (Author = "Stephen King")
retrieve
```

Compare this with the equivalent but more complicated query below:

```
open New_York r
-range (B Books) (P Publisher)
-select Pname
-where (Author = "Stephen King")
       & B.P# = P.P#)
retrieve
```

We now introduce the concept of *Dynamic Attributes* in MDSL, which are transforms of actual schema attributes. Their use makes it possible for a user dynamically to transform data values into other forms. The syntax of dynamic attributes in MDSL (Litwin & Abdellatif [LA87]) is as follows:

```
-attr_d [hold] a : C/R
-define by MT(s) = m
```

The *a* represents the name of the dynamic attribute, the *hold* argument is optional and if it is specified further queries may refer to *a*, otherwise the scope of *a* is limited to the defining query. The *define by* clause defines the mapping, *m*, of the actual schema attribute(s) *s* to *a*. The *MT* represents the mapping type, which may be one of *D* for a dynamically defined dictionary, *F* for a formula, or *P* for a program. The forms of the corresponding clauses are as follows:

```
-define by D(s) = (a1, s1),...,(an, sn)
-define by F(s) = formula
-define by P(s) = Multics_Segment_Name
```

where s_i are the actual data values and a_i are the corresponding dynamic values.

Example 5: from the Manchester and Glasgow databases, list the details of books that have been assigned a *** rating.

Assume that the Manchester database uses '*', '**' and '***' as ratings to assign to books, whilst the Glasgow database assigns 1, 2 ,3 or 4. Further assume that a Manchester '***' rating is equivalent to a Glasgow 3 or 4, a '**' is equivalent to a 2, and a '*' is equivalent to a 1. The query formulation below uses a dynamic attribute with a dictionary mapping type to resolve these differences:

```
open Glasgow r Manchester r
-range_s (B Book BookItems)
-range (x B)
-attr_d Rating : C
-define by D(BookItems.Rating) = (***, 4), (***, 3)
                                 (**, 2), (*, 1)
-select x
-where (x.Rating = "***")
retrieve
```

Notice that *x* ranges over the tuples of both *Manchester.Book* and *Glasgow.BookItem*. The *where* clause contains the predicate *x.Rating* = "***". When the tuple variable *x* is ranging over the *Book* relation the actual schema attribute *Book.Rating* is used by the query. However, when *x* is ranging over the *BookItems* relation the schema attribute *BookItems.Rating* is not referred to, instead the query uses the dynamic attribute *Rating* in the selection clause (which has previously been derived from *BookItems.Rating*).

Example 6: from the Manchester and New York databases, list the prices of books written by the author Stephen King.

Assume that the New York database represents its book prices in US Dollars and the Manchester database in Sterling. The query formulation, which employs a dynamic attribute mapped by an arithmetic formula, is presented below:

```
open Manchester r New_York r
-range_s (B Book Books)
-range (x B)
-attr_d Cost : R
```

```
-define by F(Price) = Price * Exchange_rate
-select x
-where (x.author = Stephen King")
retrieve
```

Assume that *Exchange_rate* refers to the current Dollars/Sterling exchange rate and is a schema element maintained by one of the databases or a constant fed into the query in some manner.

REFERENCES

[BFHK94] R. Busse, P. Fankhauser, G. Huck and W. Klas, "IRO-DB An Object Oriented Approach Towards Federated and Interoperable DBMS", *GMD-IPSI*, 1994.

[BFN94] R. Busse, P. Fankhauser and E. Neuhold, "Federated Schemata in ODMG", *Proceedings of the 2nd International East/West Database Workshop*, 1994.

[BHP92] M. Bright, A. Hurson and S. Pakzad, "A Taxonomy and Current Issues in Multidatabase Systems", *IEEE Computer*, March 1992, pp 50 - 59.

[BLN86] C. Batini, M. Lenzerini and S. Navathe, "A Comparative Ana lysis of Methodologies for Database Schema Integration", *ACM Computing Surveys*, 18(4), December 1986, pp 323 - 363.

[C93] R. Cattell, editor, *The Object Database Standard: ODMG-93*, Morgan Kaufmann, 1993.

[DAOT85] S. Deen, R. Amin, G. Ofori-Dwumfuo and M. Taylor, "The Architecture of a Generalised Distributed Database System - PRECI*", *The Computer Journal*, 28(3), 1985, pp 282 - 290.

[GGF&al95] G. Gardarin, S. Gannouni, B. Fianance, P. Fanhauser, W. Klas, D. Pastre, R. Legoff and A. Ramfos, "IRO-DB A Distributed System Federating Object and Relational Databases", *Object Oriented Multidatabase Systems - A Solution for Advanced Applications*, edited by O. Bukhres and A. Elmagarmid, Prentice Hall, 1995.

[HDR97] S Hamill, M Dixon, and B J Read, "Classifying Schematic and Semantic Heterogeneities in Interoperating Database Systems", RAL Report RAL-97-xxx, 1997.

[HM85] D. Heimbigner and D. McLeod, "A Federated Architecture for Information Management", *ACM Transactions on Office Information Systems*, 3(3), July 1985, pp 253 - 278.

[KS91] W. Kim and J. Seo, "Classifying Schematic and Data Heterogeneity in Multidatabase Systems", IEEE Computer, December 1991, pp 12 - 17.

[LA87] W. Litwin and A. Abdellatif, "An Overview of the Multi-Database Manipulation Language MDSL", Proceedings of the IEEE, 75(5), May 1987, pp 621 - 631.

[LB93] X. Liu and O. Bukhres, "On Object Similarity in Heterogeneous Database Integration", *CSD-TR-93-045*, Department of Computer Sciences, Purdue University, July 1993.

[LMR90] W. Litwin, L. Mark and N. Roussopoulos, "Interoperability of Multiple Autonomous Databases", *ACM Computing Surveys*, 22(3), September 1990, pp 267 - 293.

[LNE89] J. Larson, S. Navathe and R. Elmasri, "A Theory of Attribute Equivalence in Databases with Application to Schema Integration", *IEEE Transactions on Software Engineering*, 15(4), April 1989, pp 449 - 463.

[LR82] T. Landers and R. Rosenberg, "An Overview of MULTIBASE", *Distributed Databases*, edited by H. Schneider, North-Holland Publishing Company, 1982, pp 153 - 183.

[SBE93] T. Schaller, O. Bukhres, A. Elmagarmid and X. Liu, "The Integration of Database Systems", *CSD-TR-93-046*, Department of Computer Sciences, Purdue University, July 1993.

[SCG91] F. Saltor, M. Castellanos and M. Garcio-Solaco, "On Canonical Models for Federated DBs", *SIGMOD Record*, 20(4), December 1991, pp 44 - 48.

[SGN93] A. Sheth, S. Gala and S. Navathe, "On Automatic Reasoning for Schema Integration",

International Journal on Intelligent & Co-operative Information Systems, 2(1), March 1993.

[SK92] A. Sheth and V. Kashyap, "So Far (Schematically) yet So Near (Semantically), *Proceedings of the DS-5 Semantics of Interoperable Database Systems*, Lorne, Australia, November 1992, Elsevier.

[SL90] A. Sheth and J. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases", *ACM Computing Surveys*, 22(3), September 1990, pp 183 - 236.

[SLCN88] A. Sheth, J. Larson, A. Cornelio and S. Navathe, "A Tool for Integrating Conceptual Schemas and User Views", *Proc. of the 4th International Conference on Data Engineering*, February 1988.

[RSK91] M. Rusinkiewicz, A. Sheth and G. Karabatis, "Specifying Inter-database Dependencies in a Multidatabase Environment", *IEEE Computer*, December 1991, pp 46 - 53.