1-1-2003

# The mediated data integration (MeDInt) : An approach to the integration of database and legacy systems

Suvimol Mukviboonchai
*Edith Cowan University*

# USE OF THESIS

# The Mediated Data Integration (MeDInt): An Approach to the Integration of Database and Legacy Systems

by

Suvimol Mukviboonchai

B.S., M.S.

A Dissertation Submitted in Partial Fulfilment of
the Requirements for the Award of

Doctor of Philosophy

At the School of Computer and Information Science
Faculty of Communication, Health and Science

EDITH COWAN
UNIVERSITY
WESTERN AUSTRALIA

29 October 2003

# ABSTRACT

The information required for decision making by executives in organizations is normally scattered across disparate data sources including databases and legacy systems. To gain a competitive advantage, it is extremely important for executives to be able to obtain one unique view of information in an accurate and timely manner. To do this, it is necessary to interoperate multiple data sources, which differ structurally and semantically. Particular problems occur when applying traditional integration approaches, for example, the global schema needs to be recreated when the component schema has been modified. This research investigates the following heterogeneities between heterogeneous data sources: Data Model Heterogeneities, Schematic Heterogeneities and Semantic Heterogeneities. The problems of existing integration approaches are reviewed and solved by introducing and designing a new integration approach to logically interoperate heterogeneous data sources and to resolve three previously classified heterogeneities. The research attempts to reduce the complexity of the integration process by maximising the degree of automation.

Mediation and wrapping techniques are employed in this research. The Mediated Data Integration (MeDInt) architecture has been introduced to integrate heterogeneous data sources. Three major elements, the MeDInt Mediator, wrappers, and the Mediated Data Model (MDM) play important roles in the integration of heterogeneous data sources. The MeDInt Mediator acts as an intermediate layer transforming queries to sub-queries, resolving conflicts, and consolidating conflict-resolved results. Wrappers serve as translators between the MeDInt Mediator and data sources. Both the mediator and wrappers are well-supported by MDM, a semantically-rich data model which can describe or represent heterogeneous data schematically and semantically.

Some organisational information systems have been tested and evaluated using the MeDInt architecture. The results have addressed all the research questions regarding the interoperability of heterogeneous data sources. In addition, the results also confirm that the MeDInt architecture is able to provide integration that is transparent to users and that the schema evolution does not affect the integration.

# DECLARATION

*I certify that this thesis does not, to the best of my knowledge and belief:*

*i.* *incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education;*

*ii.* *contain any material previously published or written by another person except where due reference is made in the text; or*

*iii.* *contain any defamatory material.*

Signature [redacted]

(Suvimol Mukviboonchai)

Date 1 October 2003

# ACKNOWLEDGEMENTS

Foremost, the financial support provided by the Royal Thai Government and the Office of Rajabhut Institutes Council throughout the research period is greatly acknowledged. Without them, I would not have had this great opportunity.

I express my gratitude to my supervisor, Dr Chaiyaporn Chirathamjaree, for his guidance, advice and support throughout my research period. He always provided me his valuable time despite of his busy schedule of management, lecturing and research. I sincerely thank him for his time and patience. His support is greatly acknowledged.

I would also like to thank my associate supervisor, Associate Professor Vichit Lorchirachoonkul from National Institute of Development Administration (NIDA), Thailand, for his valuable time and generous recommendations. I am really appreciated.

I acknowledge with thanks those who contributed articles. Their names are listed in the References. I am also thankful to the anonymous referees from the four conferences. Their suggestions and comments have improved the content of my research. My thanks are also due to Dr. Helen Armstrong and Dr. Val Hobbs for their valuable suggestions on reviewing my thesis proposal three years ago. I am grateful.

No word can express my thanks to Dr. Judith Rochecouste for her precious time and a great deal of support on my proposal and thesis editing, correcting grammar, and English academic writing. I know she faced difficulty with my non-native English language.

I am thankful the library support provided by the Library Account Manager, Ms. Lyn Leslie, and the technical support provided by the SCIS support team for their great care and helpful throughout my research period.

My thanks also go to my friends who studied with me at NIDA and Thammasat University, and who worked together at UNHCR and Bangchak Green Net, who have always kept in contact via e-mail over a long distance and have not let me down.

A warm word of thanks to my four-year close friends in Perth, especially Surasithe Khwanboonbumpen and Kacha Chansilp, for all the wonderful memories and their encouragement when I was disheartened. Their compassion and warmth made me feel like I was not away from home.

I am also want to thank other teachers and friends whom I could talk, discuss, and learn from.

Thanks to my brother, Porrames Mukviboonchai, for taking good care of our mother alone during this research period. I understand that he has been busy and stressed from his hard work, but he looked after our mother well.

My special thanks go to my mother, Sulai Mukviboonchai, for all her unsparing love and inspiration. I remember that she taught me mathematics when I was in secondary school, even then she did not finish with primary school. I have learnt from her about calmness, patience, and self-content. Thanks for being a great mum.

Lastly, I dedicate this work to my father, Prayong Mukviboonchai, who passed away just before I started this research.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1 - INTRODUCTION

An adequate information system is one of the critical competitive components in running a successful business in terms of transaction recording at the operation level, reporting at the management level, or decision making at the executive level. In a large or medium sized organisation, it is certainly possible to have more than one information system serving the organisation's operations. New business activities and the evolution of database technology all result in the adoption of many different database systems within an organisation, for example, legacy file processing systems, relational database systems, and/or object-oriented database systems. A single database supporting all applications within an organisation is ideal. The situation of island of information leads management and executives to become frustrated when they want to get a unique view of information from multiple systems. Therefore, data interoperability or database integration becomes necessary to obtain meaningful information from multiple and incompatible data sources.

Furthermore, many Internet and intranet technologies which play a significant role in business today increase the demand for data integration techniques. It is becoming more and more necessary to be able to integrate numerous information sources within an organisation or across organisations to serve customers and to link to suppliers via the Internet. Both legacy systems and modern databases need to be logically integrated to allow users to access information.

For the pragmatic reasons stated above, the data of an organisation or across organisations need to be interoperable to service customers, management, executives or new business projects. Therefore, this research focuses on developing a data integration architecture to interoperate multiple databases and legacy systems transparently and effectively.

## 1.1 The Significance of the Research

The question why we have to make heterogeneous data sources interoperable rather than transform them and import them into a single data source may be raised. Two major problems of transforming all different kinds of data sources into only one main data source is data latency and data integrity (CrossAccess Corporation, 2001). The integration system requires synchronisation in every transaction made to the system which is redundant and unnecessarily costs money. On the other hand, if this main data source is designed to be updated at every specified certain period of time, data inconsistency problems will happen as a result of the changes which do not propagate consistently to all related data sources.

In the process of interoperating any two or more database systems, heterogeneity is the most critical problem that needs to be solved, for instance, some databases are designed from different models, and the same real world entities may be represented by different names or measured by different units in multiple data sources. Although several researchers have been studying the conflicts and integration of heterogeneous database systems (Abdalla, 1998; Miller, 1998; Neild, 1999; Phijaisanit, 1997; Srinivasan, 1997; Yu, 1997), there is still no common methodology. Few theses have focused specifically on the integration of databases and legacy systems. In fact in legacy systems, the semantics are hidden and hard to determine.

Another significant issue is that the traditional approach integration is pair-wise or point-to-point interface. This then developed to the pre-integration approach using the global schema technique which requires complete pre-integration and is extremely expensive in both manpower and time. All local views are mapped by one global view which must be created before query processing. This raises a problem especially in a dynamic system. As a result when only the object of a local data source is modified or an operation function is evolved, this affects a number of changes on the global schema (Holowczak & Li, 1996). The global view must be recreated. It is also difficult to track overall changes either in pair-wise interfaces or in the global schema approach. Furthermore, conflicts must be solved in the process of the global schema creation. The more data sources are involved, the more difficult it is to solve such conflicts. This has led this research to focus on a solution that avoids integrating with the pre-integration approach.

## 1.2 The Purpose of The Research

This thesis focuses on investigating an approach to integrating heterogeneous data sources by:

- Addressing conflicts among heterogeneous database systems.
- Providing conflict resolution.
- Providing the appropriate architecture for achieving the interoperability or logically integrating of multiple data sources by which schema evolution will not affect the integration.
- This research covers legacy file processing systems, relational data models and object-oriented data models.

## 1.3 Research Questions

**Research question number one:**

What are the possible conflicts occurring with the integration of heterogeneous database systems? How can such conflicts be resolved? These conflicts would result from various systems using different data models. Before integrating any systems, conflicts or in correspondences between systems need to be solved to make the relevant data in those systems meaningful.

**Research question number two:**

What approaches will provide solutions, and how, to logically integrate heterogeneous database systems in the bounds of the following criteria?

- Transparency: the integration process should be transparent from users.
- Validity: the quality of the query result from the integration and conflict resolution processes.

- Scalability: requiring minimised modifications when the addition or removal of data sources are needed.
- Flexibility: component schema evolution should not affect the integration.
- Simplicity: minimising human interaction and maximising automation.

The objective of this question includes reducing the complexities of the integration process to get information from such systems so that users are not responsible for seeking where data sources are, what the conflicts are and how to resolve them. This thesis also seeks to provide a method by which the global schema is not created before issuing queries, thus the problem of schema changing can be avoided.

During the integration, there are a number of integration problems that need to be solved. The major ones are:

- The requested query may need information from multiple data sources.
  - How to define data sources relevant to the query?
  - Because object identifiers are defined independently in each source, what is the identifier used in the query?
  - How to split the requested query to each data source?
  - How will data sources, which are in different data models, understand the requested query?
- The sets of results from the query need to be integrated. They might be represented differently.
  - How to homogenise them?

## *1.4 Research Methodology*

The methodology used in the thesis is based on Formulative approach including Conceptual analysis, Conceptual implementation and Experimentation.

**Conceptual Analysis**

Firstly, the problems of integrating database and legacy systems were investigated. The topics below were surveyed and the research questions were drawn from these.

- database management systems and data models.
- conflicts and conflict resolutions.
- tools, techniques, and the pros and cons of integration approaches.
- information systems which require integration.

The research questions were formulated into the architecture requirements as the framework to construct an abstraction model based on the functional divide and conquer top-down approach... The model takes into account the relevant features according to the architecture requirements.

**Conceptual Implementation**

As a consequence of the model, the concept details were implemented to support the model constructed by developing the symbolic language and algorithms.

**Experimentation**

To prove the validity and the purpose of the model, some information systems which require logical integration were chosen as samples to evaluate and test the integration process. The result of the integration was reviewed and the integration model and algorithms were then refined.

## 1.5 The Organisation of the Thesis

The thesis is organised into nine chapters. This chapter begins with the significance and the goals of this study, followed by the research questions and methodology. The remainder of the thesis is organised as follows.

Chapter 2 and 3 present a review of literature relevant to this research including file and database characteristics, data models, definition languages and manipulation languages, heterogeneities, and resolutions. The major integration approaches of the previous research are surveyed. The strengths and weaknesses of each integration approach are emphasised. Related tools and techniques, which are useful for the integration, are reviewed.

Chapter 4 describes the framework and the development of a data integration model called the Mediated Data Integration architecture (MeDInt).

Chapter 5 introduces the Mediated Data Model (MDM), a data model used in MeDInt and appropriate for describing heterogeneous data schematically and semantically.

Chapter 6 and 7 provide the detail components, the functions, and the algorithms of the MeDInt Mediator and wrappers.

In chapter 8, the procedures and the results of the integration are presented and the model is evaluated and discussed.

Lastly, Chapter 9 presents discussion, contributions from this research, suggestions for future work, limitations and conclusion.

# CHAPTER 2 – DATABASES AND HETEROGENEITIES

To interoperate multiple data sources, the main difficulties come from heterogeneities which can be classified into three levels. Firstly, platform heterogeneity includes different hardware, communication systems, and operating systems. Secondly, database management system heterogeneity includes different data models and query languages. Lastly, data heterogeneity includes both the heterogeneities in structure of data collected and also the data itself. For example, different representations might be used to refer to the same object. This research focuses on the last two heterogeneities because the first heterogeneity perspective, hardware, communication system, and operating system heterogeneity can be overcome by middleware technologies, for example CORBA, Microsoft .NET etc. Conversely, the database management and data heterogeneities are quite complex, involve more human work, and require a precise methodology. Therefore, the heterogeneities referred to in this research are only database management and data heterogeneities.

Heterogeneities from multiple data sources resulting from the interoperability of databases and legacy systems are considered in this research. Basically, these issues arise not only from heterogeneous data sources, but also homogeneous data sources, because of design autonomy. However, heterogeneities which occur in homogeneous data sources are a subset of those in heterogeneous data sources. Consequently, this thesis focuses on the generalised heterogeneous ones.

To integrate data from heterogeneous sources, one critical point is that their data structures need to be interchangeable. This dictates that a common data model is needed to represent different data structures semantically. In this chapter, traditional and semantic data models are investigated to determine the useful characteristics for developing the appropriate data model to be a common data model for the integration. Also data definition languages and data manipulation languages are investigated to gain a basic understanding of heterogeneities. Existing heterogeneities and resolutions are classified and explored.

## 2.1 File and Database Characteristics

File processing systems are the record-keeping and retrieving systems which come before database systems. Even though these are traditional data recording systems, it cannot be denied that they are still being used in most organisations which have multiple information systems. File systems have a number of limitations, for example, separated and isolated data, data duplication, application program dependency, and the difficulty of representing data in the users' perspective (Date, 1990; Kroenke, 2002).

The database approach was introduced in the 1970s to overcome the problems arising from legacy file-processing systems. The limitations of file recording systems mentioned above were then overcome (Codd, 1970; Date, 1990; Kroenke, 2002). Data from different purposes that were separated and isolated into different files in different systems without any related information could be integrated into a database system. This makes it easier for users to create a view or inquiry from several entities. A well-designed database especially in terms of data integrity aspect can reduce data duplication. In terms of program independence, data in a database can be accessed by its database management system, and not by an application program, thus, any changes made to the database will not affect application programs.

In terms of heterogeneous data integration, the characteristics of legacy file processing and database management to be considered are as follows.

TABLE 2.1 COMPARISONS OF FILE PROCESSING AND DATABASE MANAGEMENT

| | Files | Databases |
|---|---|---|
| Data | Isolated | Integrated |
| | Duplication | Duplication reduced |
| Metadata | No | Data Definition Language |
| Data Retrieval | Application | Query Language |

Table 2.1 illustrates that, firstly data stored in file processing systems are isolated and duplicated because the relationship information cannot be defined. Secondly, no schema information is identified in file processing systems because there is no metadata. Finally, the query languages provided in database management systems can be used to retrieve data, while data retrieval in file processing systems depends on the application.

## 2.2 Data Models

There are two meanings of data models which always cause confusion (Hirschheim, Klein, & Lyytinen, 1995). The first is the graphical, conceptual, notational or textual information which perceptively represents the data of a system. Data models are used to represent the organization information logically by data structures. The other meaning of data model is "the outcome of using a data modelling language in some specific situation" (Hirschheim et al., 1995). Data models are generally related to a data definition language (DDL) and a data manipulation language (DML) to define data structures or schemas to represent objects or entities. This research uses the term data models in the second sense.

Data models provide the structuring of database systems. Several kinds of data models have been developed, for example, the hierarchical model, the network model, the relational model, the nested relational model, and the object model. The network, hierarchical and relational data models can be defined as classical data models (Gray, Kulkarni, & Paton, 1992; Hirschheim et al., 1995). To overcome weaknesses in the classical data models, a variety of data models have been developed, for example, the semantic data model, the object-oriented model, and so on.

### 2.2.1 The Relational Data Model

Database systems mostly are based on the relational data model. Codd (1970) presents the relational model applied from a mathematical concept. A database is perceived as a collection of tables. A relation or a table is a collection of tuples or records. The ordering of tuples is unimportance. Relations describe entities or relationships between entities. Properties or attributes make differences of relations. A primary key is the unique identifier for a table. Tables or views (virtual tables) can be created, altered or deleted by using a data definition language. Users inquire to a database using a data manipulation language. In this part, the relational algebra including a number of operators is provided to operate one or more relations to create a new relation. These operators can be classified into two groups: traditional set operations and special relational operations. The traditional set operations are union, intersections, difference and Cartesian product. The special operations are restrict, project, join and divide (Date, 1990; Kroenke, 2002).

### 2.2.2 The Semantic Data Model

Codd (1979) extended the relational model to capture more meaning from the data to provide more intelligent databases and more systematic database design. This activity is so called Semantic Data Modelling. The attempts were searching for meaningful units of information that larger than n-ary relation called atomic semantics.

The Semantic Data Model (SDM) is designed to clearly and precisely describe databases to be closer to the human perception more than the relational data model (Bertino, Catania, & Zarri, 2001; Hammer & McLeod, 1981). Entities are grouped into classes represented by an SDM schema. Each class or semantic object includes a class name, a collection of members, a textual class description, and a collection of attributes which represent object characteristics.

The Semantic Model provides perception or conceptual representation of real world objects. Abstraction is one of the features that serve this representation. There are four main abstractions: generalisation, aggregation, classification, and association (Bertino et al., 2001). Semantic data models have been introduced to overcoming the semantic limitations of the relational model. Semantic Models represent some important types of constraints more easily: key dependencies and inclusion dependency. Languages used for semantic models are able to query abstract data types.

Semantic models can be categorised into three main classifications (Hammer & McLeod, 1981). The first class covers the abstraction mechanism or aggregation such as the Entity Relationship Model (ERM). In the second class, the use of attributes to interrelate objects is added, for example, the Functional Data Model (FDM) and DAPLEX (Shipman, 1981). An example of the third class is the Semantic Database Language (SDM) (Hammer & McLeod, 1981). An SDM database is a collection of entities organised into classes, or types. Moreover, there are a number of semantic models: TAXIS, SAM, IFO, RM/T, GEM, etc.

### 2.2.3 The Hyper Semantic Data Model

Hyper Semantic data models combine the concept of semantic data models and artificial intelligence by focusing on object, operations, relationships and associated knowledge (Potter, Trueblood, & Eastman, 1989). The characteristics of this model are:

- generalisation, classification and aggregation derived from semantic data models,
- membership ( 'is-a-member-of'),
- constraint, ('is-a-constraint-on'),
- heuristic (inference mechanism),
- temporal (representation of synchronous or asynchronous relationships).

### 2.2.4 The Object Data Model

The Object Modelling Technique (OMT) methodology uses three kinds of models to describe a system: the object model, the dynamic model and the functional model (Blaha & Premerlani, 1998; Rumbaugh, Blaha, Premerlani, Eddy, & Lorensen, 1991). An object model, presented by an object diagram, describes the static structure of a system covering objects, relationships, attributes and operations. A dynamic model, presented by a state diagram, describes the interactions among objects, which are changed overtime. A functional model, presented by a data flow diagram, describes how data values are transformed and computed within a system.

An object is a boundary concept. An object class is a group of similar objects. The classification concept allows objects with the same attributes and behaviour to be grouped into a class. A class can be defined as a specialisation of one or more classes. A class defined as a specialisation is called a subclass and inherits attributes, messages and methods from its superclass. The subclass can specialise another class by additions and substitutions. An object is an instance of its class. Generalisation and inheritance are abstractions for sharing similarities among classes. A link is an instance of an association. An association describes a group of links connecting objects from the same class. Associations may be one-to-one, many-to-many, or ternary.

An operation is a function or transformation applied to objects. Polymorphism allows an operation to have more than one method on several classes, but such methods must have the same signature. The same operation may behave differently when applied to different classes. Encapsulation is the concept of separating the internal and external implementation details of an object.

### 2.2.5 The Object-relational Data Model

The object-relational data model was developed to be compatible with the relational data model and to provide extended object capabilities such as primitive type extensions, complex types, inheritance and so on (Bertino et al., 2001). Examples of object-relational DBMS are Oracle, DB2, Sybase, UniSQL etc.

### 2.2.6 The OMG Object Model

The Object Management Group (OMG) Object Model can be described by objects, requests, types, interfaces and operations (OMG, 2001). *Objects* are real-world entities with their unique identities. An object is an encapsulated entity which can be requested for some services from clients. Objects are instances of types. Clients request services by issuing *requests*. A request consists of an operation, a target object, optional parameters, and an optional request context. *Types* are classes of objects that are grouped together, and can be related through the subtype/supertype relationships. A type defines the state and behaviour of objects. A type is an identifiable entity with an associated predicate defined over entities. An associate predicate consists of a mathematical function with a Boolean result. An entity satisfies a type if the predicate is true for that entity. An entity that satisfies a type is called a member of the type. An object can have only one type. The extension of a type is the set of entities that satisfy the type at any particular time. A type can inherit from other types and multiple inheritance is supported. *Interfaces* are descriptions that a client may request of an object through that interface. *Operations* are entities defining the behaviour of objects. They have their own identifiers which can be requested for services from clients. Operations have signatures such as name, argument types, and returned types. Operations cause method invocation in the object implementation (OMG, 2001).

### 2.2.7 The ODMG Object Model

The ODMG-93, initiated by the Object Database Management Group (ODMG) - a working group within the OMG, is an object-oriented database management system (ODBMS) standard supporting portability across database systems. The ODMG 3.0 (Cattel & Barry, 2000) currently consists of:

- a data model (ODMG/OM) which is based on OMG object model,
- object specification languages which are the Object Definition Language (ODL) used to define object types, and Object Interchange Format Language (OIF) used to load the instance of an ODMS to or from files,
- a declarative language which is the Object Query language (OQL) used for querying and updating objects, and
- C++, Smalltalk and JAVA language binding.

ODMG/OM is compatible with OMG/OM, because ODMG/OM has been developed specially for database management system concepts. Therefore, ODMG/OM is an extension and superset of OMG/OM (Ben-Natan, 1995). ODMG supports the ISO

STandard for the Exchange of Product data - STEP (Schönhoff, Strässler, & Dittrich, 1997; Strässler & Schönhoff, 1998).

The ODMG object model supports objects and literals (values). Objects have a state and a behaviour. The object state consists of a number of properties, which can be either attributes or relationships. An attribute is related to a class, but a relationship is related between two classes. Literals can be:

- atomic types: long, short, float, double, Boolean, char, and string,
- types defined through the set, bag, list, and array constructors,
- enumeration types defined by the enum constructor, and
- the predefined, structured types date, interval time, and timestamp.

*Type* has an interface and implementations. The type definition, properties and operations, are supported by an instance of this type. Each implementation consists of data structures supporting the properties of the type and methods that implement the operations defined by that type. Types define the dynamic database schema; that means the model supports schema evolution. Types can be objects themselves and can have attributes. Types have two importance properties: the extent to which they are the set of all instances of type, and a set of keys which can define a set of properties that uniquely identify an object in an extent. It is also extended to support instance model such as a relationship between objects (Ben-Natan, 1995).

*Properties* defined for a type are an instance of a type. They can be queried or manipulated. Properties are represented as attributes or relationships. Attributes are part of the type definition which maps a named value with an instance of a type. Relationships are defined between two types to maintain referential integrity (Ben-Natan, 1995).

*Operations* are part of the type definition. They model the behaviour of instances of the type. An operation is composed of its name which is unique for each type, argument names and their types, returned types, and exceptions (Ben-Natan, 1995).

*Objects* are encapsulations of state, identity, and behaviour. Objects can be mutable or immutable. Mutable objects have an identifier and they may change their state throughout their lifetime. The state of an immutable objects is its identity. An object is the root of a hierarchy for mutable objects, and a literal is the root of hierarchy for immutable objects.

The ODMG standard does not support views which are provided in RDBMS. It provides meta data management at the object level. It also allows operations, updates, insertions, etc to be performed on individual objects or collections of objects.

## 2.3 Query Languages

A query language is separated into two parts: data definition and data manipulation. Data definition languages are used to define the structures of information including creation, modification, and deletion operations. Data Definition Language (DDL) is the term that is used in relational database management systems (RDBMSs). Data manipulation languages refer to data retrieval operations. Data manipulation languages for the relational data model are non-procedural languages based on mathematics – relational calculus and relational algebra (Codd, 1970). Query languages allow access to the information in a declarative, value-based manner. Using query languages is the only way to access a relational database management system. SQL is the standard query language for relational databases. C-SQL (Sciore, Siegal, & Rosenthal, 1994) is an extended SQL used to deal with semantic values.

In object-oriented database management systems, there are two ways to access data: navigating on object identifications (OIDs) and using query languages. Manipulation languages provide constructs to access and use the information in a programmatic manner. ODMG defines object manipulation language (OML) to support both C++ and Smalltalk. Object Query Language (OQL) is a declarative language for querying object-oriented databases. It provides an SQL-like query language. The Object Definition Language (ODL) is a programming language-independent specification language based on Interface Definition Language (IDL) syntax to define ODBMS schemas and semantics (Ben-Natan, 1995). ODL provides a way to define object types and structures.

Some other examples of query languages are SQL-92, an SQL extension concerned with object-oriented aspects (Cattel & Barry, 2000), VQL (View Query Language) (Abdalla, 1998), the derived version of OQL to support semantic context, XQuery, an XML query language, designed by the World Wide Web Consortium expressing queries across the structure of XML (*XQuery 1.0: an XML query language*, 2002). Bolloju (1996) presents a semantic approach to achieve semantic interoperability based on semantic query transformation by providing the Structure Object Query Language (SOQL), an object-oriented model which is rich in semantics itself. It interoperates two autonomous information system contexts by the transformation of SOQL to SQL. The mappings of structures, names, and attributes are used in the process of the transformation with an assistance of domain knowledge.

## 2.4 Heterogeneities

Information from different data sources cannot be integrated or interoperable because of heterogeneities of data models, schema designs, or semantic contexts. Morgenstern (1997) states that there are four levels at which differences may arise, including differences at the data level, data schema level, data model level, and the metadata model level. Kim and Seo (1991) classify conflicts in multidatabase systems into schematic and data conflicts regarding to the relational data model. Heterogeneities in this thesis are classified into three levels: Data Model Heterogeneities, Schematic Heterogeneities, and Semantic Heterogeneities.

### 2.4.1 Data Model Heterogeneities

Database management systems serving the application systems in an organisation may be different because of a change of technology. This causes the use of different data models which is one of the major problems in integrating of heterogeneous database systems (Reddy, Prasad, & Reddy, 1989). In addition, Data Model Heterogeneities lead to differences in structure, constraints and query languages (Sheth & Larson, 1990). Further than the differences in characteristics of data models themselves, in this study, Data Model Heterogeneities cover two differences, those of data definition languages and data manipulation languages. The consequence of different data definition languages is that the data integration system cannot get the schema or data definitions of component data sources. Conversely, different data manipulation languages lead to the problem of how to inquire data from heterogeneous data sources.

### 2.4.2 Schematic Heterogeneities

Schematic Heterogeneities are discrepancies in the structure of component data sources. In other words, the same concept is structured or modelled differently. Data Model Heterogeneities and design autonomy cause the differences in the structures. Schematic Heterogeneities can be categorised into three types: Naming conflicts, Structural conflicts, and Classification conflicts.

In terms of design autonomy, data source components are designed using its own terminologies in each independently-designed data source. This causes ***Naming conflicts*** (Goh, Madnick, & Siegal, 1994) or *inconsistencies in naming objects* (Reddy et al., 1989). In some cases, different names are assigned to the same concept, called synonyms. For example, the object representing the course information for students to enrol was named *unit* in one data source, but *course* in another source. On the other hand, when the same name is assigned to different concepts, these are called homonym (Batini, Lenzerini, & Navathe, 1986), for example, *name* of the entity *Book* (*Book.name*) is an attribute referred to the names of the books, while *name* of the entity *Author* is an attribute referred to the names of the authors.

Naming conflicts can occur in both object and attribute levels. Kim, Choi, Gala & Scheevel (1993) classify these conflicts into *Table versus table* and *Attribute versus attribute* conflicts. The former occurs when tables having the same name are used to represent different objects in different systems, or tables having different names are used to represent the same real world object in different systems. The Attribute versus attribute conflict occurs when attributes having different names are used to represent the same object in different systems, or attributes having the same name are used to represent different objects.

***Structural conflicts***, a further set of conflicts, sometimes called *Table versus attribute conflicts* (Kim et al., 1993; Kim & Seo, 1991), Schematic Heterogeneity (Miller, 1998), or Type conflicts (Batini et al., 1986) occur when different structures are used to refer to the same concept. The same information can be represented as an attribute in one system, but as an entity in another system or an attribute is represented by multiple attributes in another systems. For example, in library systems, authors can be represented by only an author's name as an attribute in an information system, but represented by an entity including author biography in another data source.

This conflict includes the combination of many-to-many table conflicts and many-to ,many attribute conflicts (Kim & Seo, 1991). Critchlow (1997) classifies Structural conflicts into simple and complex structural conflicts. *Simple structural conflicts* occur when the same concept entities in different data sources can be mapped directly one-to-one. *Complex structural conflicts* occur when an entity is represented by several entities in another data source.

This research also defines a third type of Schematic Heterogeneities resulting from either a specialisation or generalisation called ***Classification conflicts.*** For example, in a university information system, staff and students are defined as different entities in a relational database, but both of them are a subtype of a person object type in an object database. The object type includes the shared characteristics of students and staff such as id, name, address and date of birth. The unshared properties are defined further in staff and student objects.

### 2.4.3 Semantic Heterogeneities

In order to exchange information among disparate sources, the meaning of data represented in each source has to be considered in addition to the differences in the structure of data. This means that semantic interoperability is required. Semantic Heterogeneities are discrepancies in the meaning of related data among heterogeneous systems, in another words, different ways of representing the same or overlapping data. Such discrepancies may be due to differences in system design, missing data, and other issues. They can exist even when data has come from the same kind of database management system, but are designed differently by database administrators. This category is the major consequence of design autonomy. Semantic conflicts are classified in this research as followed.

Firstly, ***Naming conflicts*** (Goh et al., 1994) or *Different expressions* (Kim & Seo, 1991) which can occur in the semantic level as well as in the schema level are the synonym or homonym of values of data. For example, ***month*** could be represented differently by *'Jan', '1', '01, or 'January'.*

***Representation conflicts*** (Goh et al., 1994), which Holowczak & Li (1996) call *Format heterogeneity*, occur when different formats or data types are used to represent the same object such as a student identification number which is represented by characters in one system, but by numbers in another system.

*Different units* (Kim & Seo, 1991), *Measurement conflicts* (Goh et al., 1994) , or ***Scaling conflicts*** occur when different units are used to measure an object in different systems. This leads to data which cannot be integrated with different units. Normally, this type of conflict is hidden and not easily solved because general data models cannot represent the context of data. For example, employee's *salary* in one system is coded on monthly basis, but on a yearly basis in another system.

***Level of Abstraction Conflicts*** or *Granularity conflicts* (Goh et al., 1994) are inconsistencies of data in disparate sources. This type of conflict occurs from data collected in different levels of composed data or abstraction. For example, the number of students in a system is classified by year in one system, but by faculty in another system.

*Different precisions* (Kim & Seo, 1991) or ***Precision conflicts*** (Abdalla, 1998) occur with different cardinalities, for example, a score is represented by A, B, C, D and F in one system, but by a percentage in another system.

***Missing data*** is that data which is gathered in one system, but does not exist in another system. Kim and Seo (1991) call this *Wrong Data* and may be caused by incorrect-entry data or obsolete data.

***Scope conflicts*** are discrepancies in the scope of the data stored in different systems. For example, a faculty system has only student information of students in the faculty, but the student information system collects information on all the students in the university.

There are further types of conflicts, for example, ***Computational conflicts*** (Goh et al., 1994) occurring when the values of the same object are computed in dissimilar ways, and ***Behaviour conflicts***, identified by Abdalla (1998), occurring when using object-oriented models which are different in operations, parameters and return types.

## 2.5 Conflict Resolutions

Schemas and the sets of result from multiple data sources may be represented differently. During the integration process, these heterogeneities or conflicts need to be resolved. A number of conflict resolution methods have been surveyed. They have been classified into schematic conflict resolution and semantic conflict resolutions.

### 2.5.1 Schematic Conflict Resolutions

Schematic Heterogeneities make the difficulties of integrating the same concept which is modelled differently. These are the first thing that needs to be resolved to obtain the unique concept of the heterogeneous data sources. The followings are some attempts to resolve Schematic Heterogeneities.

*Schema Translation* (Batini et al., 1986) is the technique mostly used in the global schema approach to merge or restructure different schemas to provide users with a unique schema. It is very convenient to users, but the process of creating the global schema is very complicated in large database systems. Abdalla (1998) similarly resolves Schematic Heterogeneities in the global schema integration by using *mapping techniques* for both naming and structural conflicts. Naming conflicts can be resolved by mapping a global name to local names. Structural conflicts can be resolved by generating global entities mapping to local entities. Critchlow (1997) also defined the mapping between databases which so called Schema coercion. The Entity-relationship data model are used as a canonical data model to represent the corresponding schemas. These correspondences then are used to generate a program to transfer data between databases.

There are four techniques of *object matching* classified in (Zhou, Hull, & King, 1996). *Key-based matching* is that objects from different databases should use the same key, called a universal key. *Lookup-table-based matching* holds pairs of object ids or keys for the corresponding objects. *Comparison-based matching* compares attributes of two objects, based on arithmetic or logical comparisons or user-defined functions and then returns a Boolean value. Lastly, *historical-based matching* is two objects that match each other can remain matched even if they cease to satisfy other conditions. These object matching techniques are used in Squirrel prototype (Zhou, Hull, King, & Franchitte, 1995).

In the case of different names of equivalent entities or the same name for different entities, and different names for equivalent attributes or the same name for different attributes, *a catalog* (Kim, 1995), *tables* (Holowczak & Li, 1996), or *meta-data repository* (Abdalla, 1998) can be used for maintaining these correspondences of attributes in disparate data management systems. However, it is not appropriate to maintain higher attribute correspondences such as one to many relationship attributes.

Kim (1995) suggests three *join* methods to integrate relevant data in heterogeneous systems. *Horizontal Joins* involve using union to unite entities. A union compatible join can be used if and only if each attribute of two local databases has its corresponding attribute after the transformation process. The extended union compatible join is used when there are inheritance hierarchy conflicts. *Vertical Joins* are used for integrating either entities or attributes among heterogeneous databases to one entity. *Mixed Joins* are the combination of horizontal joins and vertical joins.

Yan, Ozsu, & Liu (1997) presents a homogenisation methodology in the AURORA mediator system. An import schema is constructed. Then, schema mismatches are resolved by transformation operators in the relational data model environment (AURORA-RH). A group of related relations or related attributes are materialised to create a derived relation.

### 2.5.2 Semantic Conflict Resolutions

Kim (1995) suggests three ways of *homogenizing representations* to resolve different representations of equivalent data. Firstly, *different expressions*, which involve using separate codes or values to represent the same data, can be solved by defining the same object with different representations. A static lookup table can be created for defining equivalents, or operators can be defined using a multidatabase query language. Secondly, *different units* can be solved by defining arithmetic expressions (Kim, 1995). A formulae has been defined by Holowczak & Li (1996) for converting values in one system to correspond with units in another system. However, this resolution is not precisely accurate, that is, in some cases it operates accurately in only one direction, because of the decimal from the truncation of the reversed conversion. Lastly, *different precision* involves the domains of attributes, which are defined by different cardinalities, resulting in different scales of precision for similar data. A mapping among domains of equivalent attributes must be constructed by using a many-to-one mapping to convert a number of more precise domains to a less precise domain. If it is converted in an opposite way, this resolution is not precisely accurate (Kim, 1995).

Kim (1995) also suggests two ways to resolve data mismatches in heterogeneous systems by ***homogenizing attributes.*** Firstly, *type coercion* or data type mismatches are conflicts in which data types of equivalent attributes have different domains. A resolution is needed to change the data type of one attribute into another data type. There is no problem with changing an integer number to a real number, but there is a truncation problem for changing a real number to an integer number. Secondly, *attribute concatenations* are resolutions involving a character-type attribute in one system which is represented by more than one character-type attribute in another system. An operator can be defined for concatenating these attributes.

The *Object Exchange Model (OEM)* transforms objects into schema-less objects in which object id, object label, type and value are included. Meaningful tags or labels are used for describing meanings of objects instead of schemas (Papakonstantinou, Garcia-Molina, & Widom, 1995).

Abdalla (1998) defines *semantic specifications* to represent models semantically. There are two types of specifications which are enumerated domains and semantic contexts. *Enumerated domains* are for resolving conflicts from different expressions. An enumerated domain is an ordered set of defined value. For example: An attribute 'month' can have domain (Jan, Feb, …, Dec). A similar attribute can have domain (1, 2, …, 12). An enumerated domain can be multivalues ((Jan,1), (Feb,2), …, (Dec,12)). *Semantic contexts* are a set of elements, each of which is a pair of a property and an assigned value (LengthUnit=cm).

*Articulation axioms* are bi-directional (Holowczak & Li, 1996). These axioms will return a true value if the logical expression is true in a given context. The benefit of bi-directionality is that it can be reversed accurately. (Holowczak & Li, 1996) also suggests that Naming conflicts can be solved by *Aliases* and Representation Conflicts can be solved by *Superclasses,* a characteristic of the object model to represent related component entities.

*Tables, operators or functions* can be defined in class definitions for solving heterogeneity. Using the benefits of functions, a data mining approach was suggested to discover data value conversion rules from the data (Lu, 1998; Lu, Fan, Goh, Madnick, & Cheung, 1997). This resolution can also be used in the case of the complex heterogeneity. Domain structural mismatches can be solved by using functions and mapping tables.

To resolve the conflict that was defined in the previous section as Table versus attribute, an *independent view* can be constructed to access data. This view neither depends on any specific names nor changes when schemas are modified (Miller, 1998). Also conflicts have been solved in the Multibase project using a *generalisation concept* by inheriting the common characteristics (both attributes and functions) and defining them as a supertype definition.

Sciore et al. (1994) describes values semantically by composing a simple value and its context information to be a *semantic value* which can be exchanged between systems via converting from the source context to the receiving context with the assistance of conversion functions. These conversion functions can be implemented in four methods: programming language, table lookup, on-line data source, and logical rules. Conversion functions also may be total/non-total, lossless/lossy, or orderpreserving/non-orderpreserving.

## 2.6 Summary

Heterogeneities can occur in several levels. In this research, they are classified into three main classes: Data Model, Schematic and Semantic Heterogeneities which require different conflict resolutions. A number of conflict resolutions were also reviewed in the chapter.

A number of data models has been investigated with the aim of obtaining useful characteristics for developing a data model appropriate for this study. The result is the formulation of an interchangeable data model, called the Mediated Data Model (MDM), to be used in the heterogeneous database integration in this research.

# CHAPTER 3 – INTEGRATION TECHNIQUES

Data heterogeneities and conflict resolutions have been reviewed in the previous chapter. Data integration approaches, which are the procedures to integrate or interoperate data from multiple data sources, are reviewed and presented in this chapter. The limitations of each approach are emphasised. This chapter also includes brief information of integration middleware such as CORBA.

## *3.1 Integration Approaches*

In the last twenty years, several approaches to provide an integrated view of heterogeneous data sources have been introduced to bring about the interoperability among heterogeneous systems. In this research, they are classified into translation, global schema, federated database, multidatabase, mediation and other integration approaches.

### 3.1.1 The Translation Approach

The Translation approach or point-to-point scenario needs highly specialised translation for each pair of local data sources, because it requires customising case-by-case interfaces. Therefore, the number of required translators grows geometrically especially when component data sources increase (the number of required translators is n*(n-1)/2 when n is the number of data sources). The development of these ad hoc programs/translators is expensive in terms of both time and money.

### 3.1.2 Global Schema Approach

The global schema approach is a tightly-coupled approach or a fully-integrated approach, by which individual schemas from multiple data sources are merged by a global schema to provide a single view as shown in Figure 3.1.

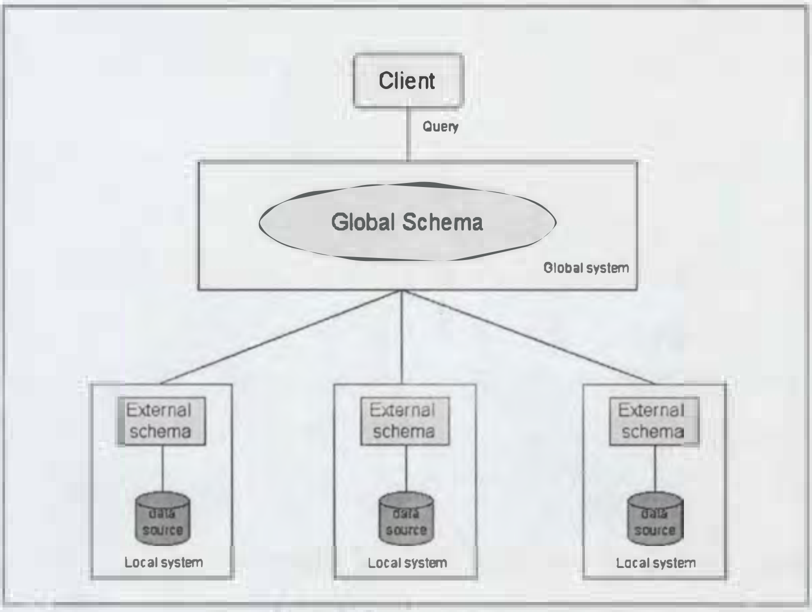


FIGURE 3.1 THE GLOBAL SCHEMA APPROACH

This approach allows accessing to multiple local data sources through the global schema interface. The conceptual global schema is provided as a logically centralized database (Hughes, 1991). This is another layer above the local external schemas and which accesses local systems through the external interface of local databases (Bright, Hurson, & Pakzad, 1992). Most global schema approaches are relational data models. Multiple local schemas are consolidated bottom-up for creating a global schema. It is quite convenient for users to have a uniform view and access to multiple data sources through the logically integrated global schema without knowledge of local schema heterogeneities. However, the schematic and semantic heterogeneities must be resolved during the process of creating the global schema. This causes a major difficulty in thoroughly understanding the schema and semantic differences of local schemas which have been designed autonomously in order to homogenise such differences (Kim, 1995). Therefore, the integration process of this approach is more complicated when the number of local schemas to be integrated increases. This approach is hard to automate because human understanding is necessary to identify the schema and semantic conflicts. There is no general solution when integrating more than two data sources whether all component schemas should be integrated once or two schemas should be integrated at a time (Bouguettaya, Benatallah, & Elmagarmid, 1999). Furthermore, in dynamic systems, when local schemas usually change, the pre-integrated global schema is affected and required to be recreated to correspondence to the local schemas.

Commonly, the integration is composed of two main steps: schema translation and integration. The purpose of the schema translation (schema mapping or operational mapping) is to translate local schemas which may be in different data models into a common data model that used in the integration. The main purpose of integration is to resolve the existing conflicts between different representations in different component systems to provide the correspondence information. This task can be divided into four steps:

- Pre-integration process, where the schemas to be integrated are selected and different requirements and constraints on the integrated system are collected.
- The comparison of component schemas to detect conflict in their representations and correspondences between them.
- The conformation process, which brings the components schemas into compatibility and resolve conflicts between them. The automation conflict resolution is not feasible, and the process has to be performed with close interaction with designers and users (Abdalla, 1998).
- The merging and restructuring of component schemas into global schema views.

This is a strict approach in that the global schema creation process is separated from the query process. Furthermore, the mapping between global and local schemas is required. The addition, the modification or deletion of local schemas influences the global schema being adjusted.

Critchlow (1997) presents a global schema approach by the assistance of the schema coercion technique that transforms sources' schemas to a reference schema before generating a transfer program to transfer data to the new created schema.

Abdalla (1998) provides a global integration by introducing a Functional Integration Technique (FIT) based on the object-oriented model. An abstract view in a common data model integrated from each local data sources is created. Conflicts are resolved before the local data sources are integrated into a global view. A descriptive language, the View Definition Language (VDL), is introduced to represent the local views. This VDL can be mapped to IDL modules. The View Correspondence Schema (VCS) is used to define the different correspondences between local views.

The Functional Integration Technique (FIT) is based on the object model providing the global schema mapping of local entities to resolve structural, semantic and behaviour conflicts (Abdalla, 1998). An example is given for the integration between two databases. However, the integration will be much more complex when the number of databases increase. Furthermore, in practical, entities probably cannot be mapped one by one.

### 3.1.3 The Federated Database Approach

The Federated Database Approach is more flexible than the previous approaches. A Federated Database System (FDBS) can be a tightly- or loosely-coupled approach. It depends on federation management and integration (Sheth & Larson, 1990) whether users or database administrators are the ones who control over the component schemas. A loosely-coupled FDBS has multiple federation schemas controlled by users while a tightly-coupled FDBS can have only a single federation schema or multiple federation schemas with constraints controlled by database administrators.

From Figure 3.2, the local schema is the conceptual schema of local data sources. Local schemas in different data models are transformed into component schemas in the common data model. Shared data for each federation can be specified in export schemas. A group of export schemas are then integrated by a federated schema. An external schema, a subset of a federated schema, will be defined for users if it is a tightly-coupled approach.

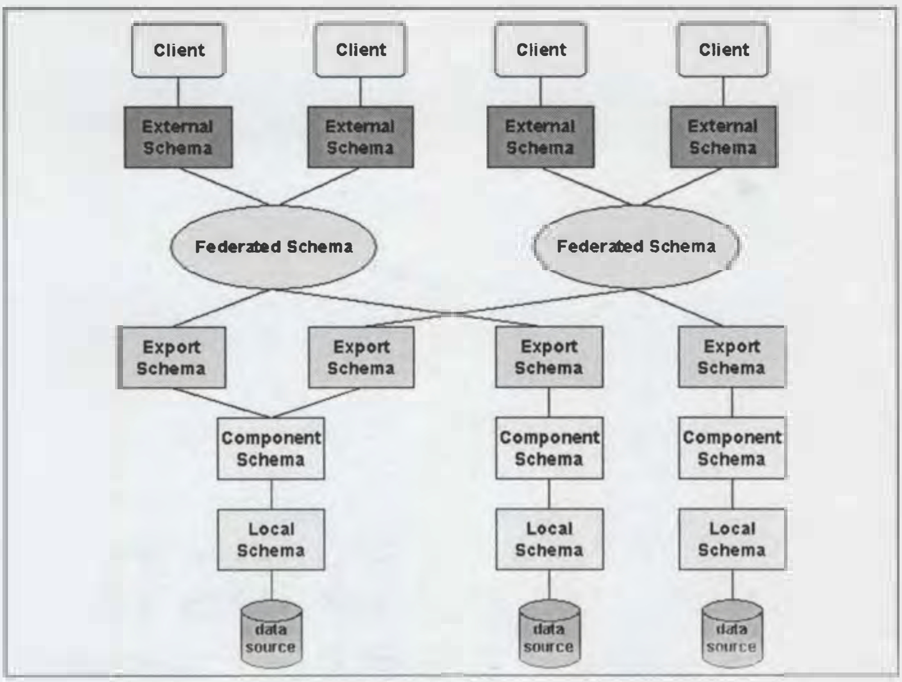


FIGURE 3.2 THE FEDERATED DATABASE APPROACH

Because this approach is quite broad, its advantages and disadvantages could be discussed separately by classifying FDBSs in terms of how schema are integrated: that is with tightly-coupled or loosely-coupled approaches.

Tightly-coupled FDBSs allow users to query one or more federated schemas without knowledge of local data sources. However, it still requires complete pre-integration. The federated schema must be developed before issuing any queries, so any changes in local schemas would affect the federated schemas. View updating is partially supported (Bouguettaya et al., 1999). This approach would violate component schema constraints and the autonomy of component schemas (Holowczak & Li, 1996).

In loosely-coupled FDBSs, it is flexible for users to map semantic meaning. However, view duplication may be generated by users, because they do not know that others use the same view. This also causes the problem of view updating with multiple semantic mappings. Even if the loosely-coupled FDBSs provide creating a new view easier than in the tightly-coupled FDBSs, it is still difficult to detect dynamic changes in the export level (Bouguettaya et al., 1999).

From a federated information system workshop (Conrad et al., 1999), it has been found that schema integration is a difficult process involving detecting and solving semantic heterogeneities among structures, constraints, and the behaviour of the component databases.

### 3.1.4 The Multidatabase Language Approach

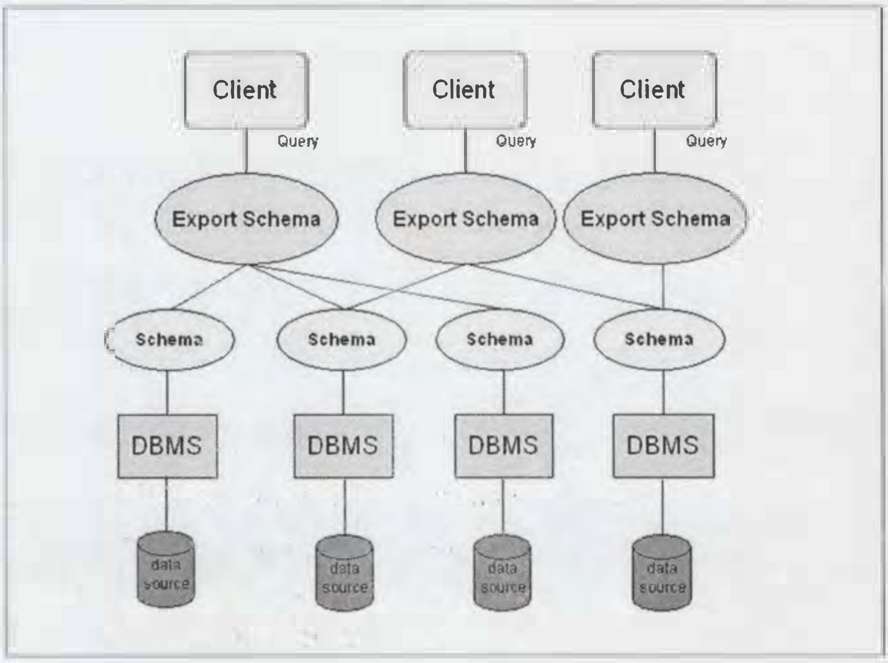


FIGURE 3.3 THE MULTIDATABASE LANGUAGE APPROACH

The multidatabase language approach shown in Figure 3.3 is more loosely-coupled than the previous approaches. It has been introduced in an attempt to resolve the problems of the previous approach by discarding the complete or partial schema integration. This approach allows users to query local database systems directly without any global schemas. It places the integration responsibility on users by providing a multi-database manipulation language as a query language tool which is able to communicate with the local databases and which is capable of managing semantic conflicts through their specification. Users can see all the local schemas and create their own logical export schema (Heimbigner & Mcleod, 1989) from selected schemas, which are relevant to information they need. The strong point of this approach is that it maintains the autonomy of local databases (Hurson & Bright, 1996). However, it requires users to find relevant data in component data sources and to understand their component schema and semantic contexts to be able to resolve conflicts in creating their own views. This will be more complicated when dealing with a large number of component data sources.

Kim and Seo (Kim & Seo, 1991) present UniSQL/M, a multidatabase system which utilises the relational model as a common data model. Component databases systems have to be converted firstly into relational schema, then a multidatabase schema would be created as a view of the component schemas.

This approach is more flexible. A new export schema can be defined easily when required by the query language tool. Users define the export schema and the mapping before querying. Therefore, it is easy to add data sources. However, the processes of defining export schemas and querying are still separate.

### 3.1.5 Mediation Approach

The mediation approach (Figure 3.4) is a recent approach to interoperate heterogeneous data sources. The main purpose of the mediation technique is to reduce the complexities of the integration and make it transparent to the users. This approach allows users to issue a query to the mediator as if it is a centralized homogeneous database. The query will be transformed by the mediator to other query languages corresponding to relevant logical data sources (Neild, 1999). Response data from each sub-query is composed by the mediator before such data is returned to users. The mediator, the major component in this approach, consists of a knowledge module placed in an intermediate position for bridging between clients and servers (Weiderhold, 1995; Wiederhold, 1992). The knowledge that a mediator provides would include information about where data is stored, and what structures and semantics of data representations are required for each user's view.

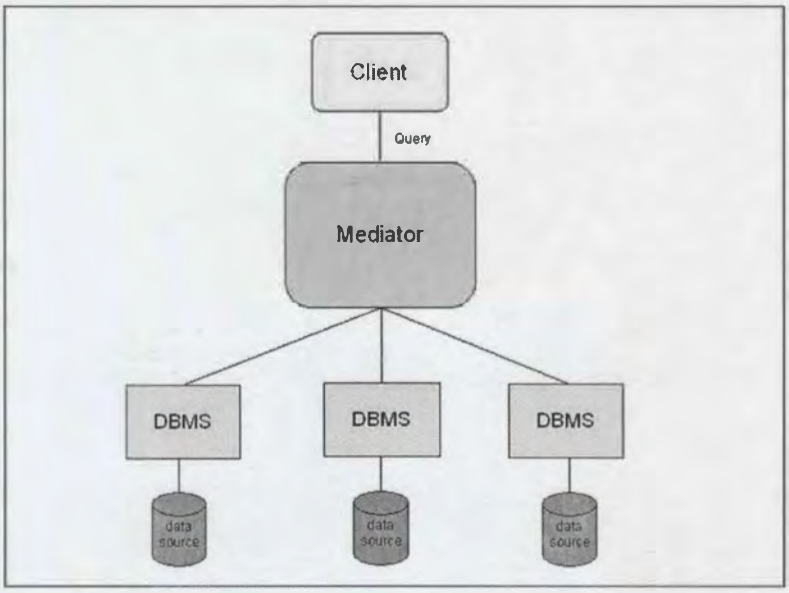


FIGURE 3.4 THE MEDIATION APPROACH

Context Mediation (Sciore et al., 1994) is an architecture consisting of information systems, data environments, context mediators, conversion libraries, and shared ontologies. The context mediator is the central component of the architecture. It acts as an agent exchanging values from one information system to another by using semantic values as the unit of exchange, together with semantic mappings from shared ontologies and functions in conversion libraries. In this approach, data values have their own associated contexts. A data value can be exchanged by converting it from a source context to a receiving context. A data environment has two components: semantic-value schema and semantic-value specification which provide attributes and properties information. The context mediation consults data environments to determine what conversions are needed. The shared-ontology specifies mappings which describe naming equivalences among information systems. The last component, the conversion libraries, contains all conversion functions. C-SQL (Context-SQL), the extended version of SQL is used to get benefits from meta-attributes.

TSIMMIS (Li et al., 1998), a project of the Stanford database group in conjunction with IBM, is a mediation architecture integrating data from heterogeneous systems by translating a query on the integrated view into a set of source queries. The mediators use the view definitions to translate the query on the user views into a logical plan. Object Exchange Model (OEM) is used to deal with exchanging heterogeneous data. It also provides wrappers as interfaces to the mediator.

The AURORA mediator system (Yan et al., 1997) is composed of an interactive mediator author's toolkit (MAT), a mediation enabling algebra, a query rewriting algorithm, and transformation rules that facilitate query optimisation. It integrates heterogeneous sources by a homogenisation methodology. The concept transforms the relation in the source to the relation format in the target. Thus, homogenisation removes the schematic conflicts of data sources relating to an integrated view. A data source can be integrated by a registration mechanism. The relational algebra and operators are extended and designed for expressing homogenising views. Queries against the views are mapped to subqueries against the data sources via wrappers. AURORA provides a collection of workbenches, each consisting of a mediator skeleton and a Mediator Author's Toolkit (MAT). Mediator skeletons are empty view mediators and become custom-made mediators when views are defined. Building a mediator means building a mediator view and a query processor. Mediators are constructed from mediator skeletons which have these built-in capabilities: a mediator enabling algebra (MEA) for defining views and a repository to maintain them, and a query processor that considers queries posed against views defined via the MEA.

Garlic (Roth et al., 1996; Roth & Schwarz, 1997) is another example of a mediator system working together with wrappers to provide an integrated view of multiple data sources. Each wrapper models data as objects and provides the method invocation on such objects.

Neild (1999) presents a mediation approach called the Virtual Data Integrator. It has two components: knowledge representation and query processor. A global schema is constructed by the knowledge representation from the information of related objects, contexts, and constraints. The query processor then can interpret the query.

The mediation approach is flexible in that it allows users to do the integration while issuing the queries. No prior creation of global schema is needed and new additional data sources are easily added to the system. However, a knowledge of data source structure is necessary.

### 3.1.6 Other Approaches

The limitations of the above integration approaches have led integration technologies towards a new variety of solutions. Various theories have been applied to solve integration problems such as the object-oriented model, knowledge base, and modelling. Examples of these approaches are discussed below.

Data Warehousing systems are different from integration systems in that a data warehouse is an instantiated view (Jakobovits, 1997) which serves to categorise data on a multi-dimension. Nonetheless, data warehousing systems are static; updating of local data sources does not affect them until reconciliation time. Query execution does not have to deal with complicated processes, for example, query translation, or to communicate with data sources which are in different data models. The main purpose of a data warehouse is to provide users with the summarised information from historical data. Data warehousing therefore derives selected information from data sources, removes inconsistencies, and transforms the information to suit the query and analysis (Seligman & Rosenthal, 1996).

DataFoundry (Critchlow, Ganesh, & Musick, 1998) is a mediated data warehouse supported by a domain-specific ontology. The mediators transform data from source format to data warehouse format and transfer query requests to data sources. Ontology is a resource to generate mediator, and supports the query processor and guides schema evolution. There are three types of knowledge: formal definitions of databases, mappings and methods; concrete instances of these descriptions; and domain-specific abstractions representing knowledge about a particular field. Database descriptions are language independent definitions of the information contained within a single database. They are used to identify the translations to transfer data between data sources and the target. Mappings identify the correspondence between database descriptions and abstractions at the class and attribute levels. Transformations describe which attributes contain the same data, but in different formats, and identify the methods that can be used to translate between them.

The Information Integration Wizard Project (I-WIZ) (Hammer, 1999) has been developed by using hybrid data warehousing and a mediation approach to integrate heterogeneous data sources. The warehouse is used to store frequently accessed data and the mediation is used to support data that is not in the warehouse. This project focuses on removing structural and semantic conflicts and the merging of corresponding data by using the process of information transformation and knowledge representation.

Reengineering approaches need to migrate databases to new environments (Seligman & Rosenthal, 1996). The mappings from old schema to new schema are required. KADBASE is a schemata information integration of the engineering databases into a single global schema based on a semantic model (a frame data model).

One of the knowledge representation techniques for heterogeneous database integration is the Carnot project (Woelk et al., n.d.), based on Cyc knowledge base integration, wherein Cyc is responsible for comparing difference schemata and merging them. Cyc was launched in 1984 by Microelectronics and Computer Technology Corporation (MCC). It is a large knowledge base which deals with a huge amount of common sense knowledge. It stores knowledge about real-world objects and their relationships, and also enables high-level queries to be posed directly against a database, instead of embedding them in an application program. Carnot provides articulation axioms to map between local models and the global context.

For testing schematic integration, the ConceptDISH of Srinivasan (Srinivasan, 1997) integrates six no-semantic-conflict systems. The system incorporates conceptual integration using background knowledge in database structure and data mining for automatically discovering a set of concepts and providing a conceptual layer above the legacy and object-oriented systems. The domain abstraction based on finding similar patterns of meta level information is used instead of a common model.

The Context Interchange approach (Goh, Bressan, Madnick, & Siegel, 1999) provides a disparate information system integration framework, which is mid-way between the two traditional approaches. This approach focuses on the semantics of individual data items. That is, the semantics are independently captured and this approach allows its mediator to detect conflicts when users issue queries. It does not require the users to detect the conflicts.

The modelling approach provides a high level, semantically-rich object oriented, model containing superclasses that encapsulate each component database used to resolve heterogeneity issues. Several methods are defined to address the issue of semantic heterogeneity (Holowczak & Li, 1996).

Heiler, Miller & Ventrone (1996) also concentrate on the semantic interoperability of databases and legacy systems. Their approach extracts the semantic incompatibilities of different systems and collects the metadata in a repository for easy detection. Then, their CASE tool is used to automatically create structured, semantic information. However, this approach is still not suitable for run-time systems.

The InforFED system (Phijaisanit, 1997) is a federated database system that uses an ontology as the shared conceptual specification of all export schemas. This architecture uses the mediation data model supporting the multiple value concepts, which can export their data in their own unit values, as the common data model.

SINGle Access POint for heterogeneous data REpositories (SINGAPORE) is an integration model in which the integration process is done after users issue queries. It applies the metadata repository to provide data source structures and knowledge. The structure of the metadata repository is defined formerly to capture such information in the preintegration process (Domenig & Dittrich, 2000).

Chang & Raschid (1996) present a technique to support interoperable query processing on multiple heterogeneous databases by utilising two canonical representations. One is resolving heterogeneity based on query languages. Another one provides the mapping information to resolve representational heterogeneity among different schemas and is used to build a mapping knowledge dictionary.

Bright, Hurson, & Pakzad (1994) provide a partially automatic integration framework for relational data sources to especially help semantic identification by using global data structure to refer to local database systems. This allows users to use their own terminology to manipulate data by applying linguistic knowledge theories to match global entry terms to local data source terms. Thus, the mapping hierarchy still needs human involvement.

McBrien & Poulovassilis (2001) present a method to integrate XML and structured data sources by transforming XML documents into an entity-relationship (ER) model using a low-level hypergraph-based data model (HDM). This represents an attempt to convert XML documents into schemas to work with structured data sources.

## *3.2 Related Tools and Techniques*

In this section, related integration tools and interfaces are reviewed. A number of useful client-server standard tools have already been developed in distributed heterogeneous systems, for example, CORBA, OLE and IDL.

### 3.2.1 Wrapping Techniques

Wrapping techniques are used to integrate legacy systems with other new systems. Layering, middleware and encapsulation are examples of wrapping techniques (Aronica & Rimel, 1996).

Layering is the most fundamental wrapping technique. This method maps one form of an interface onto another form. Its functions can accommodate the complexity of existing legacy systems. Layering is useful to aggregate legacy systems. This method is helpful because operating under layers reduces the complexity of legacy systems by dividing them into several business objects.

Middleware is system integration software for distributed processing and for database and user interfaces. The field of distributed processing middleware has been growing rapidly with the support of the Object Management Group's Common Object Request Broker Architecture (CORBA). Database middleware provides common mechanisms for accessing a variety of database systems and file structures. Some database middleware products map legacy systems such as IMS onto relational or object models. Database middleware allows a system to issue a single information request and to access several data sources, which may be different vendor's database systems.

Encapsulation is the most general technique of object wrapping. This method separates the interfaces out of an implementation. Encapsulation treats systems as a black box abstract and implementation details are hidden in the box. All accesses including direct and indirect accesses are performed through interface methods. Using interface methods allows implementation details to be changed without requiring other changes. CORBA and its IDL (Interface Definition Language) allow encapsulated systems to hide differences in programming languages, systems locations, OS, algorithms and data structures. Using IDL allows object encapsulation to be freely defined apart from implementation details. Encapsulation can be used with legacy systems whose source codes are lost, because wrappers can access legacy files and databases directly. If legacy systems have a reasonably robust application program interface (API), a wrapper can use it to perform most functions.

The Distributed Information Search Component (Disco) is an example of the wrapper-based approach (Kapitskaia, Tomasic, & Valduriez, 1997; Tomasic, Raschid, & Valduriez, 1995). It provides wrapper interfaces which support relational logical operators. Disco talks to wrappers via the abstraction level.

### 3.2.2 The Common Object Request Broker Architecture (CORBA)

CORBA, developed by the Object Management Group (OMG), is a specification for an application-level communication infrastructure. It is a standard technology infrastructure for the development and deployment of object-based applications in distributed, heterogeneous environments (Distributed Management Group, n.d.; OMG, 2001). The main purposes are for reusability, portability and interoperability. CORBA simplifies distributed environments using an object paradigm that hides all differences between programming languages, operating systems, and object location

(Mowbray & Zahavi, 1995). CORBA addresses interoperability and provides an object-based central layer which can communicate over heterogeneous platforms with language and platform independence (Segue Software, n.d.). The CORBA standard defines mechanisms whereby objects implemented in different languages can communicate transparently through an invocation method (Scallan, 1999). CORBA's characteristics allow the integrator or mediator to concentrate on database management heterogeneity and data representation heterogeneity by ignoring platform heterogeneity. The ORBs are the implementations of CORBA, which are effective for system integration and for Internet accesses. Object Transaction Service (OTS) is a horizontal service of OMG that allows users to access distributed transactions across multiple heterogeneous databases and transactional legacy systems (Vogel & Rangarao, 1999). CORBA Interface Definition Language (IDL) is defined by OTS to provide a common language and syntax for client and server access. Distributed objects can be located anywhere in a network.

Components of CORBA are  Object Request Broker, Object Services, Common Facilities and Application Objects (OMG, 2001).

*The Object Request Broker*, the central component of the architecture, provides a seamless infrastructure for distributed communication across heterogeneous systems. It is the core that allows objects requesting or being requested to be transparent. Clients need not be aware of where the object is located, what programming language is used, or any other relevant aspects. CORBA provides communication facilities to applications through two mechanisms: static interfaces and a Dynamic Invocation Interface (DII). An Interface Repository stores on-line descriptions of known OMG IDL interfaces. Any interface can be used with either mechanism. The Basic Object Adaptor (BOA) is an initial set of ORB interfaces for object implementations.

Interface Definition Language (OMG IDL) is a technology-independent syntax for describing object encapsulations. Its specifications are compiled into header files and stub programs for direct use by developers. Mappings from OMG IDL to C, C++, and Smalltalk are provided. From the header files, the OMG IDL compiler generates stub and skeleton programs for each interface. The client program links directly to the OMG IDL stub. The stub acts like a local functional call with transparent interface that encodes and decodes the operation parameters into communication formats suitable for transmission. The OMG IDL skeleton program is the corresponding server-side implementation of the OMG IDL interface.

Dynamic Invocation Interface (DII) is a generic facility for invoking any operation with a runtime-defined parameter list. A runtime interface description of the operation signature can be retrieved on-line from the CORBA Interface Repository. Programming with OMG IDL static interfaces is much more simple, but the DII provides a level of flexibility that is necessary in some applications.

An Object Adaptor contains the interface between the ORB and the object implementation. It supports many type of functions for general purpose uses, object database integration, legacy integration.

*Object Services* are a shared fundamental set of lower-level services performing basic function services for implementing an object. The object naming service provides basic operations including bind, unbind, and resolve. The object event service is a reusable set of interfaces for event posting and dissemination. The object relational service provides a capability for managing associations and linkages between objects.

*Common Facilities* are the set of shared high-level services that do not perform basic functions.

*Application Objects* contain all the software such as developer's programs, commercial applications, and legacy systems.

In conclusion, integration issues are simplified because CORBA can deal with heterogeneous hardware, software, compiler versions, data access mechanisms, component/module interfaces, and networking protocols. OMG IDL provides operating system and programming language independent interface. Programmers do not have to be concerned with the operating system, the server host hardware or the server location or activation state (Mowbray & Zahavi, 1995).

### 3.2.3 Enterprise JavaBeans

Vogel and Rangarao (1999) state that "Enterprise JavaBeans (EJB) is a higher-level component-based architecture for distributed business applications that use the transaction system's lower-level APIs". EJB was published by Sun in 1998. It is a Java-based component-oriented framework for developing, deploying and managing distributed, transactional applications. EJB is a specification for server-side. It allows developers to code business logic without worrying about managing transactions such as start or terminate transactions. EJB is mainly designed for distributed transactions, but it can be used to implement non-transaction systems (Thomas, 1998). Several services of EJB are interoperable with CORBA. Java Transaction Service (JTS) is a service binding with CORBA's OTS. JTS is an Application Programming Interface (API) which is able to manage distributed transactions operating with multiple databases in disparate systems (Matena & Hapner, 1999).

### 3.2.4 Extensible Markup Language (XML)

Extensible Markup Language (XML) is a specification developed by the XML Core Working Group of the World Wide Web Consortium (W3C) organisation as a standard way of representing structured data. XML is a subset of Standard Generalized Markup Language (SGML). The goal of XML development is to make SGML documents able to be processed simply on the Web and to bring about the interoperability of SGML and HTML (Bray, Paoli, Sperberg-McQueen, & Maler, 2000). XML is a format for structured data interchange over the Internet. It supports data exchange between heterogeneous systems. It becomes one of the means that are used in transforming data from heterogeneous sources including transaction legacy data (Goldfarb & Prescod, 2000). XML is the present and future specification with which all systems tend to conform.

XML is different from HTML in that HTML has a limited number of markup tags, but any markup tag can be used in XML (Goldfarb & Prescod, 2000). The designers of XML have attempted to take the power of SGML and the simplicity of HTML to create a new language for specifying document types that are tailored for the web, it is easy to use and light weight. In XML, the meaning of the information is embedded in the document. Information is separated into meaningful chunks called elements, which are bounded by start and end tags. Tag names describe the content of the elements. Elements can have attributes, which are property-value pairs embedded in the start tag. The document has a hierarchical structure, where elements can be contained in other elements. This structure implicitly describes the relationship between elements.

XML processors are software modules used in processing XML documents by accessing the structures and contents of XML documents (Morrison, Boumphrey, & Brownell, 2000). XML applications utilise the services of XML processors to get the structure and content of XML documents. XML processors can be plugged into an XML application to process XML documents. An XML parser, part of the XML processor, is used to analyse XML markups and identify the structure of a document.

From the investigation in this research, the characteristics of XML that allow for the integration are as follows:

- Metadata: Document Type Definitions (DTDs) are schema definitions of documents. DTD enables both syntactic and semantic checks of what is legal in a document (Goldfarb & Prescod, 2000).
- Self-describing: This makes it human-readable.
- Exchanging: XML is turning into a crucial tool support for exchanging information among databases. Especially, it is able to represent the complex structure of object-oriented information which simple file format cannot represent (Goldfarb & Prescod, 2000).
- Parsing: XML can be completely parsed because its data and metadata are separated from its rendition (Goldfarb & Prescod, 2000).
- Future: XML is a proper standard for structured data on the web. Many relevant specifications are being developed for supporting XML.
- Rendering: XML can be delivered to users differently (Goldfarb & Prescod, 2000).
- Transaction processing: To do a group of actions called a transaction, XML can combine such actions into a request by nesting them as a component in a

transaction element even though an output of the first action will be an input of the second action (Goldfarb & Prescod, 2000).

- Data interoperability (Tun, Goodchild, Bird, & Sue, 1999): It is a text-based format, making it platform- and software- independent. Thus, XML documents can be exchanged over existing protocols such as HTTP. Its hierarchical structure allows powerful data constructs from databases and other applications to be specified.
- Open standard: This makes it vendor independent. Several generic tools are bound to emerge that support XML applications.

The most significant reason that XML was chosen as one of the tools in the integration process in this research is that the data type of each element need not be specified in case of data type mismatches. Data values from different data sources defined by different data types do not have to be refixed or coerced into any specified data types, which would cause the loss of accurate information.

### 3.2.5 Ontologies

Ontologies are normally used in data integration to capture domain knowledge and provide a commonly agreed understanding of a domain, which may be reused and shared across applications. The knowledge represented inside an ontology can be formalised by using five components:

- Classes or concepts all the notions which are relevant for a given application domain describing objects, tasks, functions, actions, strategies, etc.
- Relations represent interactions between concepts and are defined as a subset of a Cartesian product.
- Functions.
- Instances represent the specific instantiations of concepts.
- Axioms are used to represent properties that concepts and instances have to satisfy.

Examples of the integration methodology based on ontologies are DataFoundry (Critchlow et al., 1998), The InforFED system (Phijaisanit, 1997), and The

Distributed Information Search Component (Disco) (Kapitskaia et al., 1997; Tomasic et al., 1995).

### 3.2.6 Metadata

Metadata is a repository of stored information of data sources, reference definitions, assertions about correspondences among data sources, libraries of conversion functions, and schemas for integrated views (Seligman & Rosenthal, 1996). Morgenstern (1997) states that a basic form of metadata is a schema definition providing a form of structural metadata. Data Dictionaries (Seligman & Rosenthal, 1996) also are suggested as a kind of useful metadata to capture information from data sources, but very limited in the amount of representation information.

A library of conversion functions has been an important part when data represented by different units in multiple data sources need to be compared. One aspect needed to be considered is whether that conversion is total, lossless, or orderpreserving (Sciore et al., 1994). A total conversion means it is possible to convert any value from any unit to any other units. Currency conversion is an example of total conversion. In contrast, the granularity conflicts mentioned in Chapter 2 are an example of a nontotal conversion. The conversion function is lossless if it still gets the same result when converted from a semantic context directly to another context or when converted by a sequence of steps. The opposite of lossless conversion is lossy or nonlossless conversion. An order preserving conversion occurs when two values in a semantic context are converted to another context and the converted values still follow in the same direction of the original values.

MetaData Specification (MDS) is used to construct a metadata repository to locate and guide access to distributed heterogeneous resources (Morgenstern, 1997). High level MetaData Specification is used to drive mediators which help to link heterogeneous information systems and provide a uniform data interface, hiding the underlying heterogeneity.

## 3.3 Summary

Major data integration approaches have been reviewed in this chapter. Each of them has limitations and each is appropriate for particular cases, for example, how tightly or loosely it may be required. The global schema approach is a tightly-coupled approach which allow user to simply query on the global view, but it is a fully-integrated approach which will generate critical problems in dynamic systems. Federated database approach is quite broad. It could be tightly- or loosely- coupled depending on who, the user or database administrator, has control over the component schemas. However, the same problem in the global schema approach also appears in the federated schemas. This problem can be solved when using multidatabase language approach, but it does not support legacy systems and users have to be responsible on creating their own schema which means the knowledge of component schemas is necessary.

Taken into account the strength and weakness of the integration approach reviewed above, an alternative integration architecture is proposed in the next chapter to address research questions presented earlier.

# CHAPTER 4 - THE MEDIATED DATA INTEGRATION ARCHITECTURE

When interoperation between multiple heterogeneous data sources is required, there would be a number of conflicts arising not only from different database designs, but also from different kinds of data models employed within heterogeneous databases. These conflicts generate the difficulties of homogenisation in terms of data model, schema and semantic. The Mediated Data Integration (MeDInt) architecture for the heterogeneous data integration framework is introduced in an attempt to overcome the above difficulties. Its main focus is to provide a solution to interoperate heterogeneous data sources through transparent transformation of both the queries and the data. Furthermore, MeDInt is capable of solving not only Schematic and Semantic Heterogeneities, but also conflicts from different query languages and data models, namely Data Model Heterogeneity.

Jakobovits (1997) classifies tightly-coupled database systems, mediator systems and decision-logic based systems as static integration systems and loosely-coupled database systems and metadata repository systems as dynamic systems. A static integration system is defined as the system which Schematic and Semantic Heterogeneities are resolved when a new component data source is added to the integration system, while a dynamic integration system is the system which such heterogeneities are resolved at query time. The integration approach proposed in the research incorporates the advantages of both the mediator systems and metadata repository systems. The MeDInt architecture requires that new data sources be registered when they are added to the integration system. However, the heterogeneities are resolved at the query time. That means the mediator system is extended to make it more dynamic through the inclusion of the metadata repository.

The ANSI/SPARC Study Group on Data Base Management Systems divides a database system architecture into three levels: internal, conceptual, and external levels (Date, 1990). The internal level is a low level representation relating to the physical storage side. The external level is the high level representation relating to the user side. It can be presented differently depending on the application. The conceptual level is between the internal and external levels representing the entire information of a database. This architecture is categorised as the conceptual level according to the ANSI/SPARC architecture.

This research will investigate and design an integration technique based on the mediation approach. The mediated architecture adds a third layer between applications and data sources.

## *4.1 Architecture Requirements*

Addressing the research questions proposed previously, the following architecture requirements have been formulated as the framework to develop the integration architecture.

Requirement number 1: The schema evolution should not affect the integration. This requirement is to cater for dynamic systems where schemas could be changed frequently. When schema modification is made on data sources, it should not cause large-scale modification to the integration system.

Requirement number 2: The integration should cover the major kinds of data sources widely used such as legacy, relational model, and object-oriented model systems.

Requirement number 3: This approach should increase automation and reduce amount of work required by end-users. Users should not have to deal with conflict resolutions once they issue queries. The different terminologies used in data sources and the different structures of data sources should not affect users when issuing queries.

Requirement number 4: Concerning on scalability, the integration architecture should only require minimum modifications when a new data source is added or removed.

## 4.2 Requirement Analysis

In order to accommodate dynamic systems, from the architecture requirement number 1, that schema evolution should not affect integration and from requirement number 4, when a new data source is added or removed, the integration should only require minimimum modification, it has been found that the pre-integration approach, such as tight-coupling and translation approaches, are not appropriate because they cannot fulfil these requirements. This is because any modifications made on the component data sources cause a lot of changes to the global schema or translators (Goh et al., 1999; Goh et al., 1994).

Requirement number 2 is introduced to allow the architecture to interoperate well. That is, the integration architecture should serve the most common kinds of data sources, for example text files, XML, relational, and object databasc management systems. According to this requirement, the loose-coupling approach, such as multidatabase approach, is not practical because it is able to serve only relational database management systems.

Concerning usability and transparency, the integration system should be easy to use. This is addressed by requirement number 3, that users should not be responsible for conflict resolution when they issue queries. In general, when users issue a query to multiple data sources, they have to deal with heterogeneities among multiple results from different data sources, for example, different currencies and different naming of objects or attributes in each source, etc. This is because different data models and database designs contain different data source schemas and terminologies. The Multidatabase approach whereby users have to deal with these heterogeneities themselves when issuing queries, is also not suitable.

The translation, tight-coupling, and loose-coupling approaches do not satisfy all of the requirements described above. To accomplish such requirements, other integration approaches have to be considered. Several experiments on generating conflicts and applying solutions to such conflicts have been done. The main processes are resolving the Data Model, Schematic and Semantic heterogeneities. Data model and Schematic heterogeneities can be resolved by translation processes.

Semantic Heterogeneities require conflict resolution processes. However, further experiments done by the author have revealed that the integration process is considerably more complicated when dealing with both translation and conflict resolution at the same time. In response to these difficulties, an architecture called the Mediated Data Integration Model (MeDInt) has been proposed. A mediator, along with wrappers, are designed to mediate both requested queries and query results from heterogeneous sources. The MeDInt Mediator handles common integration tasks, while the wrappers deal with integration tasks specific to individual data sources. Translation processes are handled by wrappers whereas conflict resolution processes are done by the MeDInt Mediator. In addition, these integration processes do not directly integrate data sources schemas, but integrate only the query results from multiple data sources. This feature is the strength of the architecture in that the integration processes do not directly force multiple schemas into a unique global schema, nor do they resolve semantic conflicts directly. Rather, it slightly adjusts only the result data to conform to the pre-defined referential template. The main architecture and components of the MeDInt solution are described in the next section.

## *4.3 The* **MeDInt** *Architecture*

MeDInt, which stands for the Mediated Data Integration Architecture as shown in Figure 4.1, is based on mediation and wrapping techniques. The two main components are the mediator and wrappers acting as the intermediate agents between clients and multiple data sources to communicate both request queries from clients to data sources and also query results from data sources to clients. In addition, a data model called the Mediated Data Model (MDM) has been developed as the backbone of the integration system to generate a common data model used by the MeDInt Mediator.

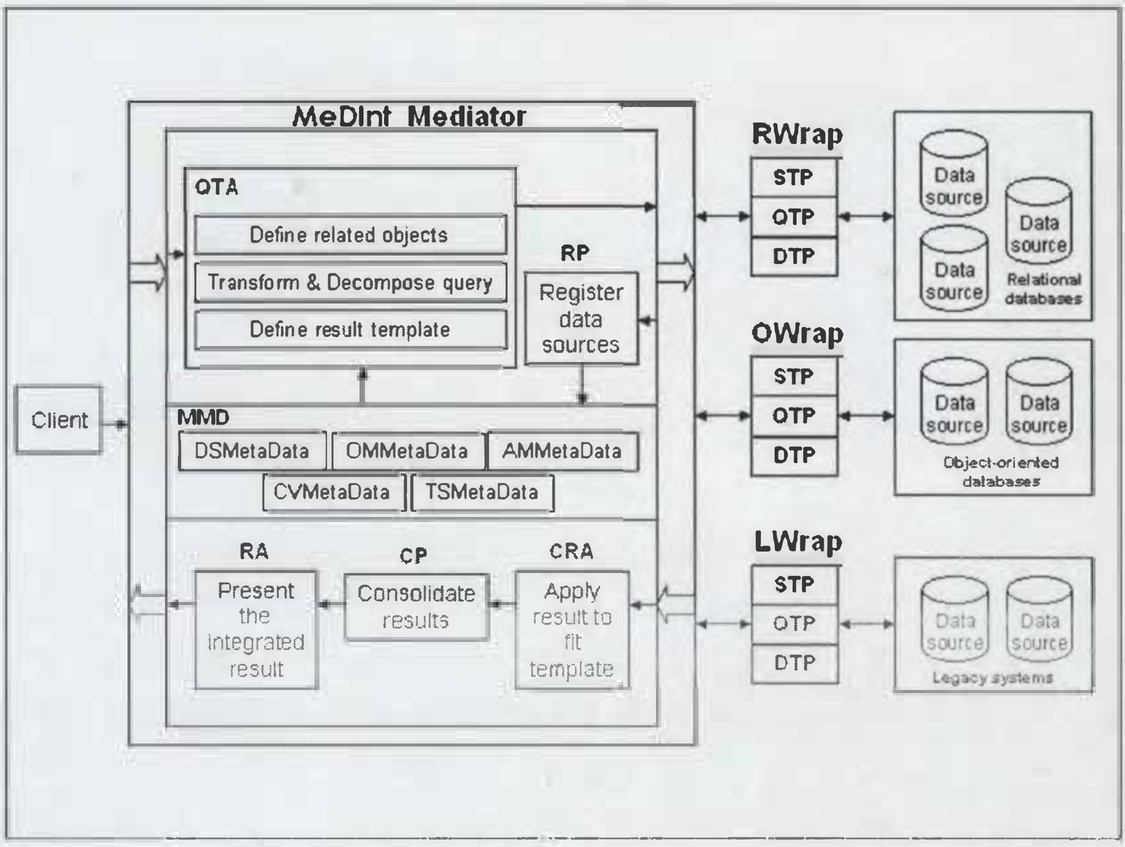


FIGURE 4.1 THE MEDINT ARCHITECTURE

### 4.3.1 MeDInt Components

The MeDInt architecture is represented by four-tiers of components: the application systems which interface to users, the mediator, wrappers and data sources (Chirathamjaree & Mukviboonchai, 2002b; Mukviboonchai & Chirathamjaree, 2001a, 2001b). In addition, the Mediated Data Model (MDM), a data model designed especially for the heterogeneous data integration framework, works along with the MeDInt Mediator and wrappers functioning as a central data model and working as the backbone of the integration facilitating the Mediator and wrappers in understanding each other.

#### 4.3.1.1 The User Interface

To get information from multiple data sources, there are two alternatives for users to issue queries to heterogeneous database systems. Firstly, users can use any query language to create the queries and the system provides translators to map from the local query language to the query language commonly used in the system. Secondly, a query language is provided for users to specify their queries. The latter option is selected in this architecture because generally query languages are not capable of utilising and specifying the heterogeneities between heterogeneous systems (Papakonstantinou et al., 1995). Therefore, this approach also provides a data model with a query language (see Chapter 5) which captures the heterogeneities for users so that they can specify their own queries, including semantic contexts.

#### 4.3.1.2 The MeDInt Mediator

The MeDInt Mediator provides middle-layer services, as an information integrator does, between the application and wrappers. In general, mediators are responsible for retrieving information from data sources, transforming received data into a common representation, and integrating homogenised data (Wiederhold & Genesereth, 1997). In this research, the MeDInt Mediator has been designed to include the following common characteristics of the integration processes:

- registering data sources information,
- defining associate objects and requesting object schemas from wrappers,
- decomposing and transforming a query to subqueries according to data sources,
- generating a result template,
- applying the multiple sets of results to a pre-defined template,
- consolidating the conflict-resolve sets of results, and
- displaying the integrated result to the user.

The components of the MeDInt Mediator and their functions are described next.

Registering Processor (RP). Once a new data source is added to the Mediated Data Integration system, it needs to be registered. This enables the integration system to incorporate the essential information from each data source.

Query Transformation Agent (QTA). When the MeDInt Mediator receives a submitted query, QTA is responsible for defining query-associated objects and requesting for object schema definitions which are in the Mediated Data Definition

Language (MDDL) format from wrappers. Furthermore, QTA transforms and decomposes the submitted query to the Mediated Query Language (MQL) format and sends a subquery to the wrapper of each source. QTA also creates a result template from the attributes requested in the submitted query.

The Mediated MetaData (MMD). MMD is a repository collecting the information necessary for the integration, for example, semantic information, data sources definitions, and conversion functions, etc. This information is critical for resolving both schematic and semantic conflicts. Many categories of MMD have been developed: Data Source MetaData (DSMetaData), Object Mapping MetaData (OMMetaData), Thesaurus MetaData (TSMetaData) and Conversion MetaData (CVMetaData) (See Chapter 6 for more detailed information).

Conflict Resolution Agent (CRA). After the MeDInt Mediator gets the query result from the wrappers in the Mediated Data Representation Structure (MDRS) format, CRA is responsible for applying each MDRS to fit the given template if they have different structures and contexts. The process of applying MDRSs to fit the template is one of the processes of indirect conflict resolution by resolving only the query result, and not the data source schemas. This is the most significant aspect of the architecture which can be described as data integration without schema integration.

Consolidation Processor (CP). CP integrates or consolidates the sets of MDRS results which have already been fitted to the template. These MDRSs already have the same structure or are structurally equivalent as all conflicts had been resolved before this step.

Rendering Agent (RA). The RA is an interface automatically generating the integrated conflict-resolved result of the query to the users.

The details of the MeDInt Mediator are described in Chapter 6.

#### 4.3.1.3 Wrappers

Wrappers are in the intermediate layer between the MeDInt Mediator and data sources. A wrapper is invoked when a data source in a difference data model is added to the integration system. Wrappers mainly act as translators providing the MeDInt Mediator with information in the common data model used in the integration system by dealing with the data model heterogeneities of different data sources. The principle objective of wrappers is dealing with data model heterogeneities including the different data definition languages and data manipulation languages by mapping different data models to the Mediated Data Model. Each MeDInt wrapper is composed of a Schema Translation Processor, a Query Translation Processor and a Data Translation Processor.

The Schema Translation Processor (STP) is responsible for translating the data definition of objects requested by the MeDInt Mediator from the data definition language of each source to the Mediated Data Definition Language (MDDL). It then sends the object schemas in MDDL to the Mediator.

The Query Translation Processor (QTP) is responsible for translating Mediated Query Language (MQL) subqueries into a specific query language which can be executed in the database management system of each data source.

The Data Translation Processor (DTP) gets a set of query results from each data source and then translates the data contents to the Mediated Data Representation Structure (MDRS).

It can be noted that unshared characteristics are pushed to the wrappers to reduce the amount of middleware modification when a data source is added, removed or modified. The details of the MeDInt wrappers have been provided in Chapter 7.

#### 4.3.1.4 The Mediated Data Model

According to the aspect of model heterogeneities, the conventional data models are not practical to represent and cover different characteristics of several data models or to be a broker to negotiate their heterogeneities. Most conventional data models are useful to describe the structure of data, but they are not suitable for describing the semantics or the context of data. This research provides the Mediated Data Model (MDM) which has been developed specifically for schematically and semantically describing data models for heterogeneous system integration. The Mediated Data Model consists of the following description languages.

- The Mediated Data Definition Language (MDDL),
- The Mediated Query Language (MQL), and
- The Mediated Data Representation Structure (MDRS).

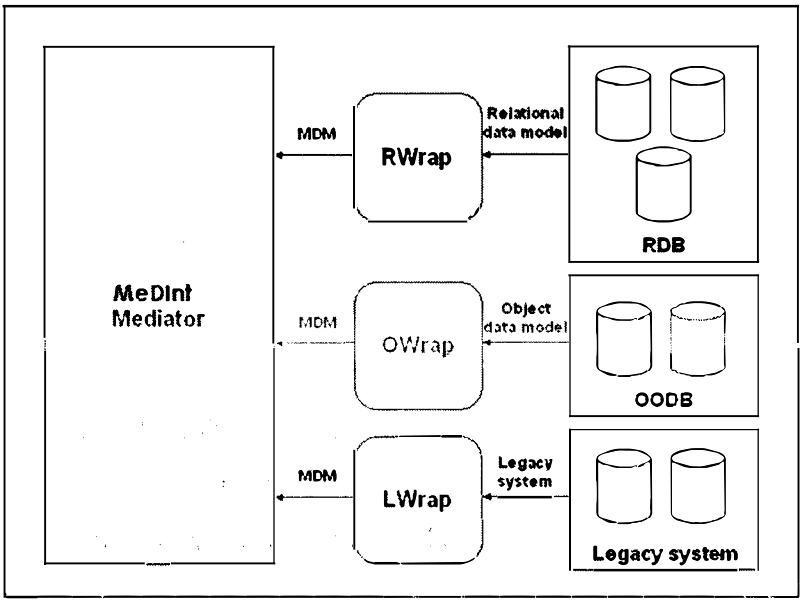


FIGURE 4.2 DATA MODEL TRANSLATION

Figure 4.2 depicts the mechanism of data model translation. A given type of data model used for a data source will be translated by its associated wrappers (such as RWrap for the relational data model) to be accommodated in MDM, which is the common data model acknowledged by components in the MeDInt Mediator. The MeDInt Mediator, therefore, does not have to deal with complications of different data models. Thus, problems relating to the Data Model Heterogeneity can be disposed of. Details of the Mediated Data Model are described in Chapter 5.

## 4.4 MeDInt *Processes*

The processes of the MeDInt Architecture can be illustrated by the following diagram (Figure 4.3).

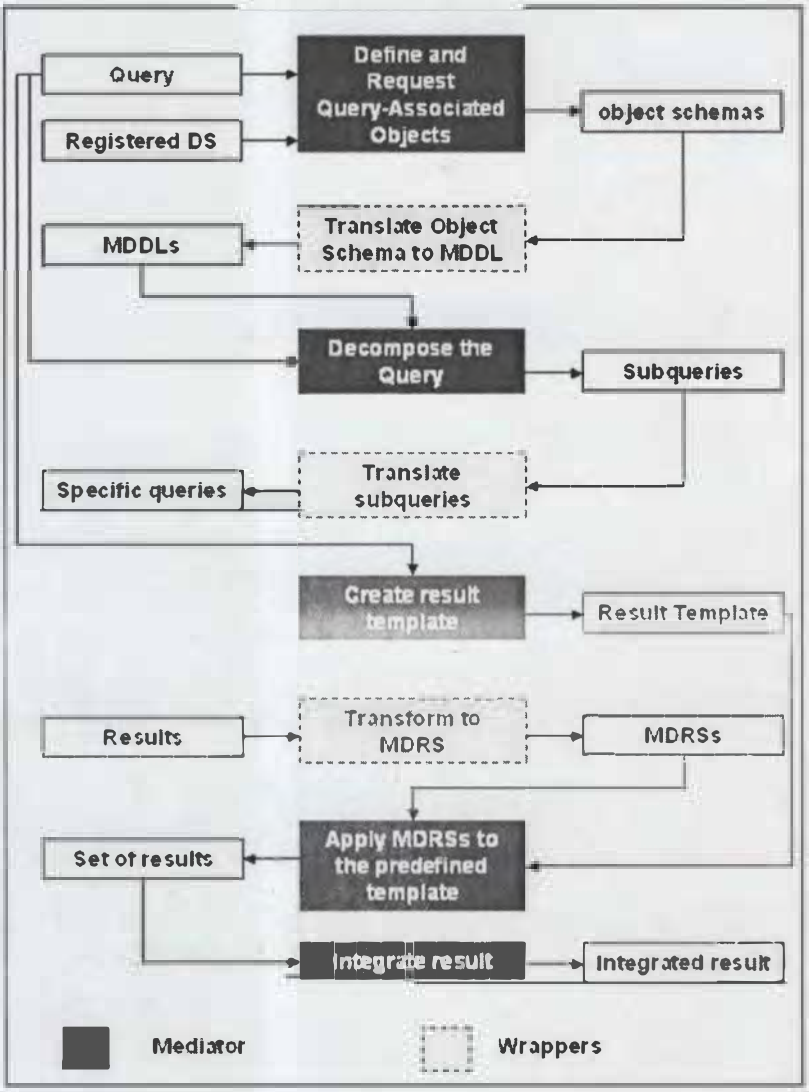


FIGURE 4.3 MEDINT PROCESSES

First, when a new data source is added to the integration system, an initialisation step is needed. The data source has to be registered to MMD by RP. Data source information, for example, assigned name, location, type, description, and constraints relating to its structure and semantics must be collected into the Data Source Metadata (DSMetaData), a category of MMD, as its schema knowledge to be provided to other components in MeDInt when required.

Generally, when a user submits a query in MQL syntax to retrieve the information they want from heterogeneous data sources, the query is submitted to the MeDInt Mediator instead of directly to the data sources. QTA then diagnoses the query, defines the objects required, and sends a request to the STP, a component in wrappers, to get the related object schema definitions. STPs translate disparate object schemas which are in different data definition languages to MDDLs. From these object MDDLs, QTA analyses again whether those gathered object schemas are sufficient to transform the query. If not, QTA specifies further indirectly associated objects from the relationships and subtypes, if any, of MDDLs of the direct objects. Therefore, QTA has to repeat the process of getting MDDLs from STPs again until there are enough object definitions for it to transform the requested query. The submitted query is transformed and decomposed by QTA to MQL subqueries which are submitted to QTPs. The QTP translates each MQL to a specific query language which depends on what kind of query languages each database management system can understand. QTA also prepares a template for the results after getting the results from multiple data sources. This method does not try to resolve conflicts directly which would be more difficult and complicated.

After getting a response data back from data sources, the DTP, a component of a wrapper, then translates the query results into MDRS. CRA resolves conflicts simply by applying all MDRSs to fit into the structure of the predefined template so that resultant MDRSs are structurally equivalent. CP then integrates the conflict-resolved results which are in the same structure and have the same semantics. The RA finally transforms the integrated result to users.

This architecture overcomes the weakness inherent in other approaches that require the physical or logical integration of component schemas as mentioned in Chapter 2. Only the query result from each source, according to the result template, will be integrated instead. The template will be created from the submitted query. The resultant data from each data source will be applied to fit to the template which is the means by which the heterogeneities are resolved.

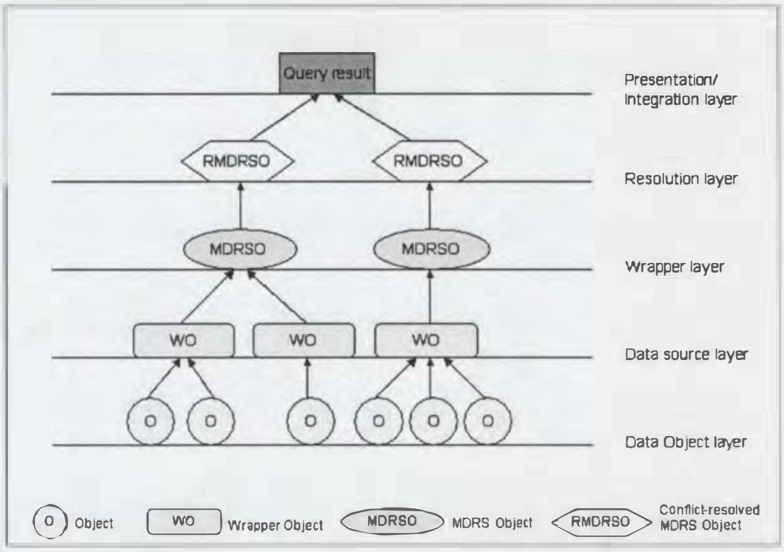


FIGURE 4.4 DATA LAYERS

An alternative view of the working of the MeDInt architecture is illustrated in Figure 4.4. Data representation is now described in terms of data layers and encapsulation. The lowest layer is the data object layer which contains objects. File or database management systems deal with their own objects in this layer. The requested objects are sent to the data source layer which presents wrapper objects to wrappers. These are encapsulated by wrappers which perform appropriate functions to get query results in MDRS objects. CRA gets the MDRS objects from the wrapper layer in order to resolve conflicts and sends RMDRS objects (conflict-resolved MDRS) to the resolution layer. Finally, the presentation/integration layer integrates the RMDSR objects to present the result of the query to users.

## 4.5 Summary

The requirements of heterogeneous data integration have been formulated and derived from both the literature and the research questions. The mediation and wrapping techniques are employed to satisfy these requirements. In this chapter, the Mediated Data Integration (MeDInt) architecture is presented. The MeDInt Mediator in collaboration with wrappers and the Mediated Data Model (MDM) have been introduced to overcome the problems in dynamic integration systems and to resolve the heterogeneity issue. The components of these three main components will be described in details in chapter 5, 6 and 7.

# CHAPTER 5 - THE MEDIATED DATA MODEL

Conventional data models have been designed concentrating on collecting and manipulating data, but they are not practical for representing heterogeneities for the integration purpose in that they are not capable of adequately brokering different kinds of data models. Basically, the object-oriented data model best describes a real-world object, but it is still not suitable to be used as a common data model because it is difficult to incorporate semantic concepts (Conrad et al., 1999). Most conventional data models are able to describe the structure of data, but are not rich enough to express the meaning or context of the data. The integration of data sources when the relevant databases have been designed dependently does not create heterogeneity problems. However, when databases have been designed independently, there are heterogeneity problems such as different terminology, data types, units of measurement, domains, scopes, and so on. Heterogeneous data integration requires a data model which is capable of describing data, schemas and contexts. This complexity suggests the need for a new data model having characteristics appropriate for supporting a mediated approach for the integration of databases and legacy systems. To accommodate this need, a model called the Mediated Data Model (MDM) which has been developed in this study specifically for describing and representing heterogeneous data both schematically and semantically.

## *5.1 The Design of the Mediated Data Model (MDM)*

With a relational data model, a relation or a table representing an entity or a relationship which users perceive can be described by a two-dimensional matrix where rows represent tuples, and columns represent attributes, as shown in Figure 5.1.

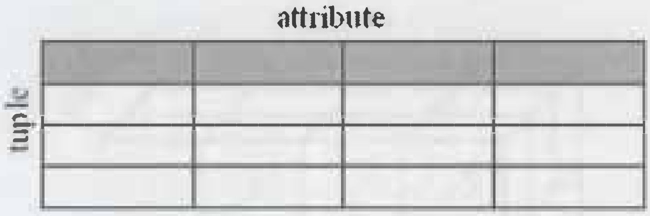


FIGURE 5.1 A 2-D RELATIONAL DATA MODEL

In general, a two-dimensional model is adequate to describe simple or atomic values in a single database system or in dependently-designed databases without heterogeneities. This is because they are normally designed according to the same context. However, such a model is not capable of expressing a number of independently-designed data sources meaningfully when interoperability is needed. Attributes from different sources may have the same name but occur in different contexts. For example, to represent an employee's salary quoted in Australian dollars on yearly basis, in a single database would not require the context parameter since all salary information within the same data source contains the same semantic context. However, when multiple data sources are designed independently, salary would probably be quoted in different semantic contexts, i.e. different currencies or different pay periods. Thus, the context of an attribute is critical when data integration is needed and two-dimensional data models would not be sufficient. This leads to the need for a new data model with semantic enrichment. The Mediated Data Model designed in this research provides a three-dimensional (3-D) approach (Figure 5.2) to denote semantic values by expressing those simple values meaningfully.

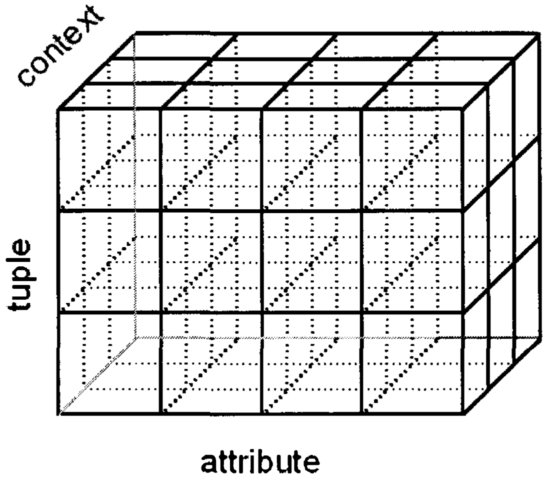


FIGURE 5.2 THE 3-D MEDIATED SEMANTIC DATA MODEL

For example, to explain an employee object type by three-dimensional semantic MDM; the first dimension, tuples, are object instances of the employee object type; the second dimension, attributes, are characteristics of the employee object type such as id, name, address, salary; and the extended third dimension, contexts, are characteristics of each attribute such as the salary attribute which is in Australian dollars and on a yearly basis. Its structure can be denoted by:

```
Salary (value, currency, period)
```

The first element is the value of the salary attribute; the second and third elements are semantic contexts of the salary attribute. An attribute value with its semantic values would be:

```
Salary (15000, 'AUD', 'yearly')
```

This value can describe the amount of 15,000 AUD salary on a yearly basis. Thus, the general syntax of an object instance can be represented in depth as:

Tuple$_i$ (Attribute$_1$ (Value, Context$_1$, Context$_2$,..., Context$_j$, ..., Context$_m$), Attribute$_2$ (...), ..., Attribute$_k$(...), ..., Attribute$_n$(...))

For example,

```
Employee (Id (value), Name (value), ..., Salary (value, currency,
period), ...)
```

An object instance would be:

Employee$_1$ (Id ('0995550'), Name ('Mark Johnson'),..., Salary (15000, 'AUD', 'yearly'), ...)

The formal definition of MDM and its components (described later in this chapter) is defined syntactically in a syntactic metalanguage notation, the Extended Backus-Naur Form (EBNF) (ISO/IEC, 1996; Scowen, 1998). EBNF's symbols are given in Appendix C.

The Mediated Data Model can be implemented by any language. The eXtensible Markup Language (XML), which is platform independent, has been selected to implement MDM. XML is based on an object-oriented model which is best for describing the schema and the semantics of objects in the real-world. XML also has flexible self-describing tags which are readable and easy to understand (Goldfarb & Prescod, 2000; Morrison et al., 2000). Moreover, XML is increasingly used as an exchange format (Conrad et al., 1999).

## 5.2 The Mediated Data Model Components

The Mediated Data Model has been developed as a schematically and semantically common data model which can be used to represent heterogeneous data models in the integration of heterogenous database systems (Chirathamjaree & Mukviboonchai, 2002a). With regard to its structural and manipulative parts, MDM consists of the Mediated Data Definition Language (MDDL) and the Mediated Data Representation Structure (MDRS) as the structural part, and the Mediated Query Language (MQL) as the manipulative part as shown in Figure 5.3. MDM reserved words are defined in Appendix D.

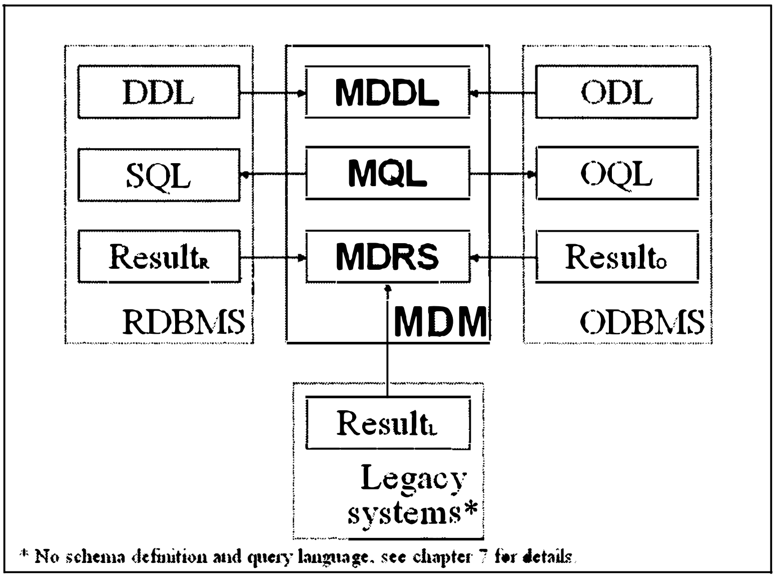


FIGURE 5.3 COMPONENTS OF THE MEDIATED DATA MODEL

As shown in Figure 5.3, MDM provides a common platform for translating relational, object, and other data definition languages into MDDL. This provides a common language for communication among components of the MeDInt Mediator and wrappers. By contrast, the submitted MQL query will be translated to the query languages of each data source to let its database management system perform its own query operation. Finally, the results from different data models will be applied to the pre-defined template MDRS. All of these translation tasks between MDM and other data models are performed by wrappers.

### 5.2.1 The Mediated Data Definition Language (MDDL)

Because each data source might be in a different data model, the MeDInt Mediator needs to be able to recognise their schemas. The Mediated Data Definition Language (MDDL) is a flexibly interchangeable definition language which can capture data definitions defined disparately in different data models. STPs (see Chapter 7) in wrappers are responsible for transforming data source definitions in any other specification languages into MDDL, so that all components in MeDInt can understand schema definitions unambiguously.

The syntax of the MDDL definition in EBNF notation is composed of the following rules:

```
MDDL_rule                  =   object_rule, {object_rule};
object_rule                =   object_identifier, ‘=’, ‘{’, [subtype_rule],
                                  [attribute_rule], [relationship_rule],
                                  [operation_rule], [key_rule], ‘}’, ‘;’;
object_identifier          =   letter, {letter |decimal digit};
subtype_rule               =   ‘subtype ’, ‘ ’, object_identifier, {‘ ’,
                                  object_identifier}, “;”;
attribute_rule             =   ‘attribute ’, attribute_defined_list,
                                  {attribute_definied_list}, ‘;’;
attribute_defined_list     =   attribute_identifier, data_type,
                                  [context_rules];
attribute_identifier       =   letter, {letter |decimal digit};
data_type                  =   ‘integer’ | ‘character’ | ‘date’ | ‘float’ |
                                  ‘string’ | user_defined;
context_rules              =   ‘{’, context_identifier, context_type_set,
                                  {‘,’, context_identifier, context_type_set},
                                  ‘}’;
context_identifier         =   letter, {letter |decimal digit};
context_type_set           =   ‘{’, context_type, {‘,’, context_type}, ‘}’;
context_type               =   letter| decimal digit, {letter | decimal
                                  digit};
relationship_rule          =   ‘relationship ’, relationship_list, {‘,’,
                                  relationship_list}, ‘;’
relationship_list          =   relationship_identifier, ‘ ’, [data_type],
                                  ‘ ’, inverse_relationship;
relationship_identifier    =   letter, {letter |decimal digit};
inverse_relationship       =   object_identifier, ‘.’,
                                  relationship_identifier;
operation_rule             =   ‘operation ’, operation_list, {‘,’,
                                  operation_list}, ‘;’;
operation_list             =   operation_identifier, ‘(’,
                                  {argument_list}, ‘)’,  ‘:’,
                                  returned_type;
argument_list              =   {argument};
argument                   =   letter, {letter |decimal digit};
returned_type              =   data_type;
```

```
key_rule                =    'key ', attribute_identifier, {'+',
                                  attribute_identifier}, ';';
```

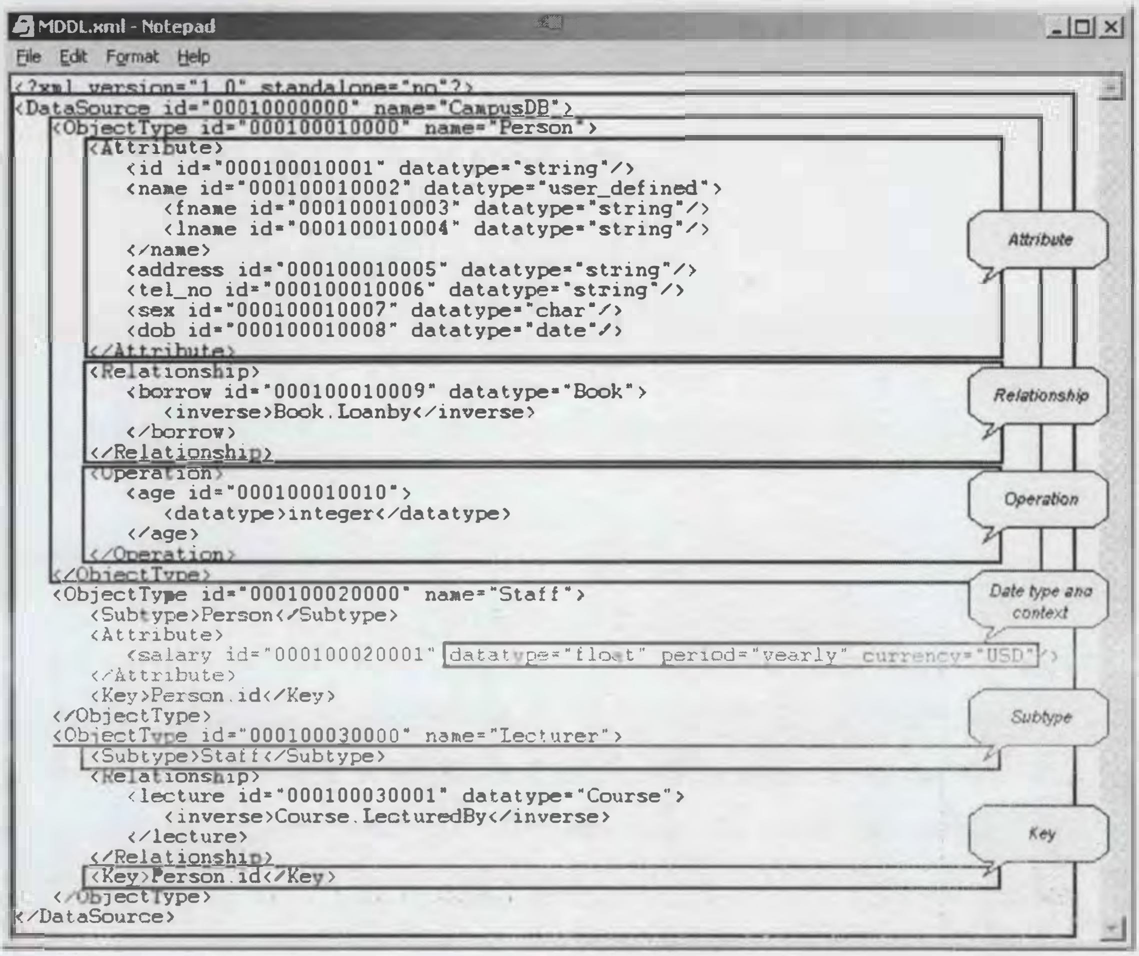
For example:

```
Lecturer     =   {
             subtype
                 Staff;
             relationship
                 Lecture      set(Course)          Course.LecturedBy;
             key
                 id;
             }
```

From MDDL above, a real-world object type, *Lecturer*, is a subtype of *Staff* class. This means that the properties of *Lecturer* are inherited from *Staff*. In addition, it associates to the *Course* object type; a lecturer can lecture a number of units. *Course.LecturedBy* is the inverse relationship of *Lecturer.Lecture*. *Id* is its primary key.

In summary, MDDL can carry out the following functions:

- object type identification,
- inheritance information identification if the object type is a subtype of any other object type,
- attribute declaration which describes the properties of the object type,
- context declaration which describes the context of an attribute,
- relationship information identification if an object associates to others.
  A relationship is the logical binary connection between two objects including one to one, one to many, many to many.
- operation information identification if the object has methods or behaviours, and
- key information which is the primary key to identify object instance.



```
<?xml version="1.0" standalone="no"?>
<DataSource id="00010000000" name="CampusDB">
   <ObjectType id="000100010000" name="Person">
      <Attribute>
         <id id="000100010001" datatype="string"/>
         <name id="000100010002" datatype="user_defined">
            <fname id="000100010003" datatype="string"/>
            <lname id="000100010004" datatype="string"/>
         </name>
         <address id="000100010005" datatype="string"/>
         <tel_no id="000100010006" datatype="string"/>
         <sex id="000100010007" datatype="char"/>
         <dob id="000100010008" datatype="date"/>
      </Attribute>
      <Relationship>
         <borrow id="000100010009" datatype="Book">
            <inverse>Book.Loanby</inverse>
         </borrow>
      </Relationship>
      <Operation>
         <age id="000100010010">
            <datatype>integer</datatype>
         </age>
      </Operation>
   </ObjectType>
   <ObjectType id="000100020000" name="Staff">
      <Subtype>Person</Subtype>
      <Attribute>
         <salary id="000100020001" datatype="float" period="yearly" currency="USD"/>
      </Attribute>
      <Key>Person.id</Key>
   </ObjectType>
   <ObjectType id="000100030000" name="Lecturer">
      <Subtype>Staff</Subtype>
      <Relationship>
         <lecture id="000100030001" datatype="Course">
            <inverse>Course.LecturedBy</inverse>
         </lecture>
      </Relationship>
      <Key>Person.id</Key>
   </ObjectType>
</DataSource>
```

FIGURE 5.4 AN MDDL IMPLEMENTATION EXAMPLE

In terms of implementation, the XML reviewed in Chapter 2, which is capable of serving MDDL characteristics, was chosen as the implementation tool. Figure 5.4 shows an example of using XML to represent MDDL. An XML document with a *DataSource* root can be applied to contain an `MDDL_rule` or the schemas in a database. The XML attributes, id and name, identify the data source object. The root element *<DataSource></DataSource>* consists of a number of nested elements *<ObjectType></ObjectType>* describing object types contained in the data source. Each has its own id and name. *<Subtype>, <Attribute>, <Relationship>, <Key>* and *<Operation>* are child elements of each *<ObjectType>*. Each *<Attributes>*, *<Relationship>* and *<Operation>* has its own id and name. *<Subtype>* and *<Key>* refer to other objects so they do not have their own object ids. XML attributes - *datatype="float" period="yearly" currency="USD"*- can be employed to represent data types and the semantic contexts of each *Attribute*.

### 5.2.2 The Mediated Query Language (MQL)

The general query languages used in database management systems are practical for manipulating a single database system, but not heterogeneous databases which consist of a number of different data models. Furthermore, general query languages are not rich enough to contain or be able to specify the contexts in the query statements. If data in multiple data sources are represented in different contexts, users need to specify the contexts of the attributes on the query in both the selection and the condition parts to ensure the correct query result. The problem of different semantic contexts in heterogeneous data sources has resulted in the need to decompose the query and create subqueries for those sources with different contexts. Thus, the central query language is required to take this into account. The Mediated Query Language (MQL) is a query language designed especially for this purpose. It is generated by QTA (see Chapter 6) for three significant purposes: as a semantic query language for users to specify their queries, as a query language used when decomposing the submitted query into subqueries to distribute to associated wrappers, and as the central query language being understood by all wrappers. MQL is an extended version of SQL which is able to capture semantic contexts. Users can identify within the *select_clause* which context of an attribute they want on the result of the query even when the data are stored in different contexts in component data sources. Moreover, they can also specify the condition of the query in the *condition_clause* in the appropriate context required.

The syntax of MQL in EBNF notation is:

```
MQL_rule            =    Select_clause, From_clause, In_clause,
                              [Condition_clause], ';';
select_clause       =    'SELECT', ' ', attribute_list, {attribute_list};
attribute_list      =    object_identifier, '.', attribute_identifier,
                              {context_list};
context_list        =    context_identifier, '=', context_type;
from_clause         =    'FROM', object_identifier, ',',
                              {object_identifier};
in_clause           =    'IN', datasource_identifier, ',',
                              {datasource_identifier};
```

```
Datasource_identifier   =   letter | decimal digit, {letter | decimal digit};
condition_clause        =   ‘CONDITION’, condition_list;
condition_list          =   condition_rule, {boolean_operator, condition_rule};
condition_rule          =   left_condition_rule, comparison_operator,
                                right_condition_rule;
left_condition_rule     =   attribute_list;
comparison_operator     =   ‘=’ | ‘>’ | ‘<’ | ‘>=’ | ‘<=’ | ‘<>’;
right_condition_rule    =   attribute_list | literal;
literal                 =   letter | decimal digit, {letter | decimal digit};
boolean_operator        =   ‘AND’ | ‘OR’;
```

The following is an example of MQL.

```
Select Staff.id, Staff.salary(currency="AUD", period="yearly")
From Staff
In DS1, DS2
Condition Staff.salary(currency="AUD", period="yearly") < 50000;
```

It can be explained from this MQL that the user wants to get an id and a yearly-based salary in Australian dollars of staff who have a salary of less than 50,000 Australian dollars from data sources *DS1* and *DS2*. MQL allows users to specify the semantic context of each attribute whose value has been stored in data sources with different contexts.

### 5.2.3 The Mediated Data Representation Structure (MDRS)

It has been found that heterogeneities also arise from the sets of query results returned from multiple data sources which are in different representations (i.e., with either schema or semantic contexts). Resultant data cannot be integrated until the Schematic and Semantic Heterogeneities have been resolved. The process of directly resolving these heterogeneities is very complicated. The Mediated Data Representation Structure (MDRS) has thus been introduced to avoid the foregoing complexities. MDRS which incorporates other components as a common data representation in MDM homogenises these different representations simply, as the practically defined-structure representing the structure of data contents with their semantic contexts, which are different in the component data sources. The DTP, a component in wrappers, takes care of translating data contents from data sources into MDRS so that the MeDInt components are able to understand it, and CRA then applies the sets of MDRS results which have different schemas and semantics to conform to the predefined template, which is also in the MDRS form.

Another significant reason why MDRS has to be implemented is that the result of the query has to be in the user-requested format. MDRS is applied as a predefined-reference for other components that deal with conversions to know what the context of that attribute should be and so that the result can be provided according to the target context.

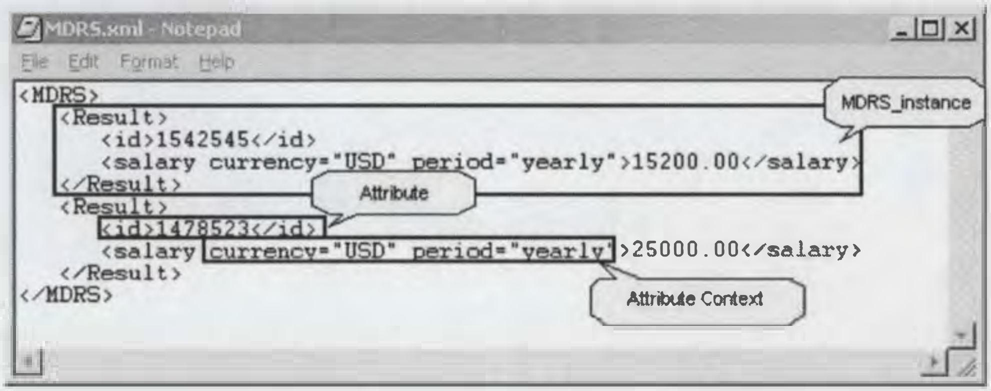
The specification of MDRS in EBNF notation is:

```
MDRS_result_set            =   '{', {MDRS_instance}, '}';
MDRS_instance              =   '(', attribute_context_value, {',',
                                    attribute_context_value}, ')';
attribute_context_ value   =   object_identifier, '.', attribute_identifier,
                                    '(', attribute_value, {context_value}, ')';
attribute_value            =   letter | decimal digit, {letter | decimal
                                    digit};
MDRS_template              =  .'(', attribute_template, {attribute_template},
                                    ')';
attribute_template         =   object_identifier, '.', attribute_identifier,
                                    '(value, ', {context_type}, ')';
```

The following is the query result that has already been translated into MDRS. It represents staff id and salary on a yearly-basis in US dollars.

```
(Staff.id, Staff.salary (currency="USD", period="yearly))

{("1542545", 15200.00 (currency="USD", period="yearly)),
 ("1478523", 25000.00 (currency="USD", period="yearly))}
```



```
<MDRS>
   <Result>
      <id>1542545</id>
      <salary currency="USD" period="yearly">15200.00</salary>
   </Result>
   <Result>
      <id>1478523</id>
      <salary currency="USD" period="yearly">25000.00</salary>
   </Result>
</MDRS>
```

FIGURE 5.5 AN MDRS IMPLEMENTATION EXAMPLE

In terms of implementation, MDRS can also be represented by XML which is flexible in exchanging information. From Figure 5.5 above, the root element - *<MDRS></MDRS>* - contains an MDRS_result_set; each element tag - *<Result></Result>* - inside represents each MDRS_instance which consists of elements -*<id></id>, <salary></salary>* - represents attribute_value of an MDRS_instance. The last important part, the XML attributes *currency="USD"* and *period="yearly"* within an attribute_value tag represent attribute contexts.

Through the MDDL, MQL and MDRS specifications, MDM is not only applicable for solving the model heterogeneities of component data sources, but it is also capable of solving Schematic and Semantic Heterogeneities.

## 5.3 Summary

One of the critical problems in heterogeneous data integration is dealing with different data models of data sources. This drastically increases complexity especially when a data integration system has to solve the Schematic and Semantic Heterogeneities simultaneously. MeDInt provides the Mediated Data Model (MDM) as an interchangeable data model used in the architecture to overcome the Data Model Heterogeneity issue. Moreover, MDM is capable of not only representing component schemas, but is also sufficiently rich in describing semantic contexts. To describe schemas and semantics, the Mediated Data Definition Language (MDDL), the Mediated Query Language (MQL) and the Mediated Data Representation Structure (MDRS) are provided as the media among different sources to give data definition and to manipulate data meaningfully. They provide semantic knowledge for the MeDInt Mediator during the integration process.

# CHAPTER 6 - THE MeDInt MEDIATOR

In this study, a heterogeneous database integration model has been proposed by incorporating a mediator and wrappers as intermediate layers between the application and data sources. The mediator, MeDInt, serves as an information integrator, between the application and wrappers. Generally, mediators are responsible for retrieving information from data sources, for transforming received data into a common representation, and for integrating the homogenised data (Wiederhold & Genesereth, 1997). In this model, the MeDInt Mediator acts as an interchangeable agent and facilitator for wrappers and clients. It consists of six components working together transparently to facilitate clients and data sources to achieve the following tasks:

- transforming and decomposing the submitted query into subqueries and then distribute them to associated wrappers;
- prcviding both schematic and semantic knowledge which is critical for query transformation and conflict resolutions;
- resolving conflicts; and
- consolidating query results.

All the functions above are served by six components (Figure 6.1), which are the Registering Processor (RP), the Query Transformation Agent (QTA), the Mediated MetaData (MMD), the Conflict Resolution Agent (CRA), the Consolidation Processor (CP) and the Rendering Agent (RA) whose functions will be described in this chapter.

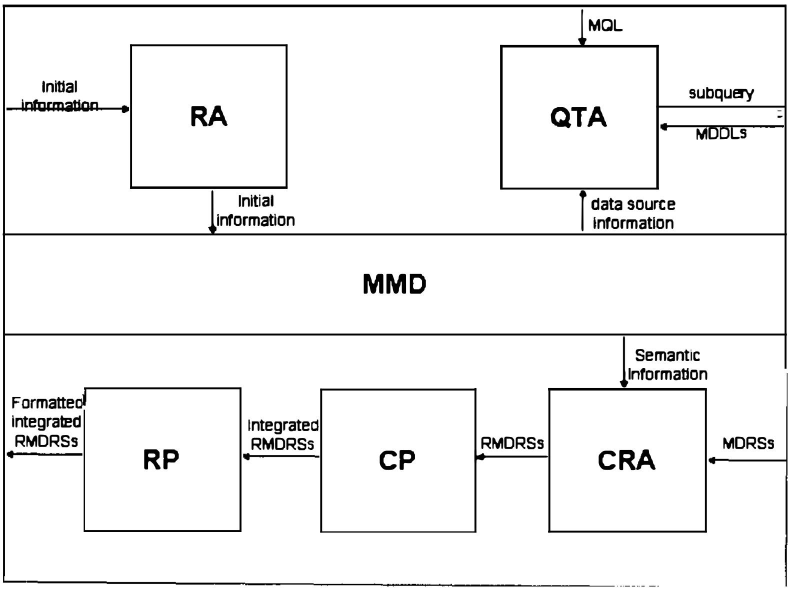


Figure 6.1 Six components in the MeDInt Mediator

## 6.1 Registering Processor (RP)

Because the required knowledge, such as different terminologies and different schema designs, in heterogeneous integration systems needs to be determined by a human, a partial automation methodology has been applied in the MeDInt architecture. The processes of schema and terminology determination will be specified manually in the initial phase. Then, the remaining of the integration process is automatic.

Data sources must be initially registered to the Mediated MetaData (MMD) when a new data source is added to the integration system. Registering Processor is responsible for capturing the principal data source information to be stored in MMD as knowledge for the integration.

The essential data source information needs to be registered to MMD, for example, data source assigned names, locations, data models, descriptions, and constraints. Moreover, in terms of terminology, all entities in each data source need to be mapped to global objects so that other components in MeDInt can perceive them. The object mapping information is also registered in MMD, and object unique ids must also be assigned to the global objects.

The significant objectives of registering new data sources are:

- To assign a unique name for each data source to avoid ambiguity, for example, if data sources in different systems have the same name;
- To identify the physical location of each data source, for example, in the form of an IP address or URL of the data source;
- To incorporate the definition of each data source;
- To capture the semantic information of each data source if there are any critically constraints to be considered. These semantic contexts must be defined to provide the context of the attributes, which might have different contexts in different sources; and
- To collect object information for mapping between local and global objects, so that the global object can be referred to in the query and can be recognised by MeDInt components.

As mentioned previously, data source and object mapping information registered in this process will be stored in MMD which will be discussed later in this chapter. Any programming or descriptive languages can be applied to serve MMD in terms of implementation. The eXtensible Markup Language (XML) was chosen in this research to represent MMD because of its self-describing tags and platform independent characteristics (Goldfarb & Prescod, 2000; Morrison et al., 2000). In addition, XML conforms to the MDM implementation which also uses XML. Examples of information registered in this initial phase are shown in the section on MMD.

## *6.2 Query Transformation Agent (QTA)*

When the MeDInt Mediator gets a user-requested query from a client, the Query Transformation Agent (QTA) cannot decompose the query at this point in time because of Schematic and Semantic Heterogeneities. Each required decomposed subquery should contain the same schema and semantic context as its related data source. To decompose the query, QTA does not have enough information about component data source schemas relating to the query nor about the different terminologies used in each source. QTA thus needs to get pre-registered data source information and object mapping information from MMD, so that it can determine query-associated objects. QTA can send a request for these query-associated object schema definitions to the STPs of the associated wrappers. However, these directly-associated object schema definitions may be insufficient to decompose the query because the objects may relate to other objects or may be a specialisation of others. Therefore, from these directly-associated object schema definitions, QTA defines further transitively-associated objects from subtypes and from the relationships of directly-associated objects. When getting enough schema information which has already been translated by STPs to MDDL and which can be utilised by the MeDInt components, QTA then transforms and decomposes the submitted query into *n* MQL subqueries (*n* depends on how many data sources the query originally related to), and submits these subqueries to the assorted wrappers. Furthermore, to facilitate the conflict resolution process, QTA creates an MDRS result template from the object, attribute and context information specified in the submitted query and homogenises query results to the template.. The process of QTA is shown below (Figure 6.2).

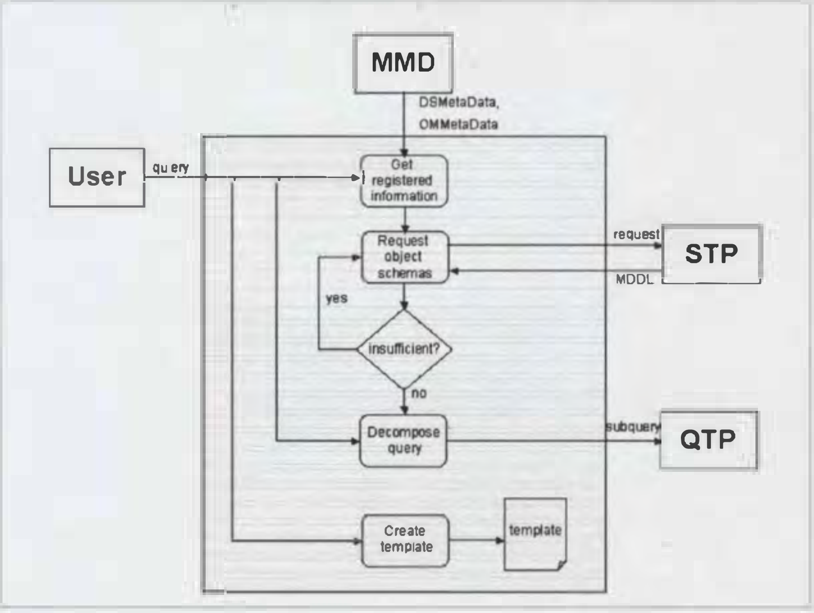


FIGURE 6.2 QTA PROCESSES

Because this architecture was designed to suit dynamic integration systems, no global schema has been created, so schema evolution is not an obstacle. The integration system fetches the schema definitions once a query has been issued. QTA requests only the necessary query-related object schema definition to transform and decompose the query.

To simplify the above QTA functions, its processes can be broken down into three parts: fetching object schema definitions, decomposing the query, and creating the MDRS template.

### 6.2.1 Fetching Object Schema Definition Process

Firstly, after receiving a user-requested query from a client, QTA has to fetch object schema definitions from query-associated data sources. To achieve this, QTA analyses which objects in which data sources are required in order to get the necessary data source information from MMD to identify query-required associated objects. Then, QTA requests the STPs for the object schema definitions. Each STP passes this request to its data source, receives the object schema definitions, and translates them to MDDLs, because they are in different data definition languages. They are then returned to QTA. After QTA has received MDDLs from the STPs, it analyses the components of the object schemas and determines further transitively-associated objects, which are also necessary in transforming the query. These may associate to, or be a specialisation of, the direct-associated objects. This means that QTA has to examine the directly-associated object MDDLs to find out:

- whether each object is a subtype of others; and
- whether there are any relationships among those objects.

If the examination falls into any of the criteria above, QTA has to request STPs for further schema definition. If the object is a subtype of any other objects, the complete object schema definitions include not only the requested object, but its superset schema definition. For example,

```
Interface Person {
    attribute       string          id;
    attribute       struct<string fname, string lname>   name;
    attribute       string          address;
    attribute       string          tel_no;
    attribute       string          sex;
    attribute       date            dob;
    relationship    Book            borrow
        inverse     Book::loanby}

Interface Staff:Person {
    attribute       float           salary;}

Interface Lecturer:Staff (key id) {
    relationship    set<Unit>       lecture
        inverse     Unit::lecturedby;}
```

*Lecturer* is a subtype of *Staff* and *Staff* is a subtype of *Person*, if the *Lecturer* information is specified in the user-requested query, not only the directly-associated object schema definition *(Lecturer)* is required, but also *Staff* and *Person* are required to assist in decomposing the query. This is because the characteristics of *Lecturer* were defined by its superset attributes and relationships in addition to its own. For example, if a query requests the names and salary of lecturers, *name* is defined in the *Person* class, and *salary* is defined in the *Staff* class, then *Person* and *Staff* schema definitions are both required in conjunction with the *Lecturer* schema definition.

For the second criterion above, if any two or more objects requested by the query are associated with each other, the relationship definition is also necessary for the query. If the requested query specifies the names of students enrolled in unit ‘CSP1143’, QTA recognised that, in addition to the *Student* and *Unit* schema definitions, the relationship between them, *EnrolRec,* is required as well.

```
CREATE TABLE Student
    (   id          CHAR(7)     NOT NULL,
        fname       CHAR(30)    NOT NULL,
        lname       CHAR(30)    NOT NULL,
        address     CHAR(50),
        tel_no      CHAR(10),
        sex         CHAR(1),
        dob         DATE,
        level       CHAR(1)     NOT NULL,
        PRIMARY KEY (id));

CREATE TABLE Unit
    (   id          CHAR(7)     NOT NULL,
        name        CHAR(30)    NOT NULL,
        PRIMARY KEY (id));

CREATE TABLE EnrolRec
    (   student_id      CHAR(7)     NOT NULL,
        unit_id         CHAR(7)     NOT NULL,
        PRIMARY KEY (student_id, unit_id)
        FOREIGN KEY (id) REFERENCES Student,
        FOREIGN KEY (id) REFERENCES Unit);
```

From the QTA analysis process described above, QTA can determine transitively-associated objects in addition to directly-associated objects from the two criteria of whether it is a specialisation of any particular type or whether there are any relationships between them. This object schema definition fetching process has to be performed repeatedly until QTA gets enough object schema definitions from the STPs for the query.

The main reason why this architecture was not designed to get all schema definitions from all connected data sources at the beginning of the request, but firstly diagnosing the query and determining which object schema definitions are required, and repeatedly getting only the query-associated object schema definitions, is that by doing so it is more efficient in terms of query performance and resource utilisation, especially when there are a few related objects in each data source relating to the requested query. This means QTA does not have to get all component schema definitions which may not be necessary for the query, but, instead, QTA can capture only few associated object schema definitions.

In the *FetchDef(Đ, Ö)* algorithm below, while *Đ* and *Ö* are arrays of the data source and the object identifications specified in *from_clause* and *In_clause* (see also MQL in Chapter 5) of $\theta$, the requested query presents the process of fetching associated objects.

```
Process FetchDef(Đ, Ö);
{Fetch object schema definitions from multiple data sources.}
Type  SourceInfo = Record of
                        DSname   : DataSourceName;
                        DTModel  : DataModelType;
                        Oname    : ObjectName;
                 End Record;
      MDDL_Str    =  MDDL_rule (see also Chapter 5)
Var   DataSource  : DataSourceName;
      Object      : ObjectName;
      DSInfo      : Array of SourceInfo;
      i, j        : Integer;
      MDDL        : MDDL_Str;
```

```
    Function GetSchDef(DSname, DTModel, Oname);
    {Get object schema definitions in MDDL syntax from wrappers.}
    Begin { GetSchDef }
        Case DSInfo.DTModel of
            'Relational'  : MDDL[Oname]:=RschmTrans(DSinfo.DSname, DSinfo.Oname);
                              {see also STP in Chapter 7.}
            'Object'      : MDDL[Oname]:=OschmTrans(DSinfo.DSname, DSinfo.Oname);
                              {see also STP in Chapter 7.}
        End Case;
    End { GetSchDef };

Begin { FetchDef }

    {Check data source validity and get essential information for query decomposition and
    transformation.}
    For all Đ[i] in In_clause
        Search for Đ[i] in DSMetaData;
        If found() then Begin
            Get SourceName to DSInfo.DSname;
            Get Type to DSInfo.DTModel;
        End;
        Else return error message that such data source has not been registered;

    {Check object validity and get object mapping information.}
    For all Ö[j] in From_clause
        Search for Ö[j] in OMMetaData;
        If found() then
            Get SourceObject to DSInfo.Oname[j] for each DSinfo.DSname;
        Else DSInfo.Oname[j]:= Ö[j];

    {Get directly-associated object schema definition from wrappers.}
    For all DSInfo.Oname[j] of each DSInfo.DSname;
        GetSchDef(DSInfo.DSname, DSInfo.DTModel, DSInfo.Oname[j]);

    {Get transitively-associated object schema definition from wrappers: specialization.}
    For each MDDL[α]
        If it is a subtype of others Then Begin
            DSInfo.Oname[j] := MDDL(α].subtype;
            GetSchDef(DSInfo.DSname, DSInfo.DTModel, DSInfo.Oname[j]);
        End;
    {Get transitively-associated object schema definition from wrappers: association.}
    For each pair of MDDL[α], MDDL[β]
        If they are related to each other Then
            DSInfo.Oname[j]:= MDDL[α].relationship;
            GetSchDef(DSInfo.DSname, DSInfo.DTModel, DSInfo.Oname[j]);
End { FetchDef }.
```

### 6.2.2 Decomposing and Transforming the User-requested Query to the Mediated Query Language Process

When QTA gets enough object schema definitions from STPs in MDDL syntax which can be utilised by all components in the MeDInt Mediator, QTA can then translate and decompose the user-requested query to MQL subqueries which conform to the schemas of each source. These MQL subqueries will be submitted to related wrappers to allow each wrapper to translate them into a specific query language that can be processed by the query engine in each source.

The processes of query transformation and decomposition begin with replacing global objects in the requested query with the local mapping objects (from OMMetaData) of each source first, and then replacing global attributes with the local attributes (from MDDL of each object, AMMetaData, and TSMetaData). These subqueries are generated in the MQL syntax and submitted to the corresponding wrappers.

In addition, Semantic Heterogeneities have to be considered in this step when the semantic contexts of an attribute value specified in the *condition_clause* of the query are different from the semantic contexts of the same attribute in component data sources. QTA has to convert the different context values transparently to users, so each subquery sent to the associated wrapper has the same context with the target data source and the wrapper does not have to deal with the context heterogeneity. Note that MQL subqueries sent to wrappers have no semantic contexts attached.

*Qtransform(Ä, Ð, Ö, Ç)* is the process of decomposing and transforming the user-requested query to MQL subqueries. *Ä, Ð, Ö,* and *Ç* are arrays of attributes, data sources, objects, and conditions specified in *select_clause, from_clause, In_clause,* and *condition_clause* of a user-requested query.

```
Process QTransform(Ã, Ð, Ō, Ç);
{Decompose and transform the user-requested query to MQL subqueries.}
Type  Φ_Rec      : Record of
                      Projection:  Array of AttrRec;
                      Object    :  Array of ObjectName;
                      DS        :  Array of DataSourceName;
                      Selection :  Array of ConditionRec;
                      Join      :  Array of RelRec;
      Attribute_context         :  String;
Var       Φ                       : Φ_Rec;
          i, j, α, β, m           : integer;
          fr_context, to_context  : Attribute_context;

   Function GenSubQ(DS);
   {Generate a subquery.}
   Begin { GenSubQ }
      Φ.Projection:= Ã;
      Φ.Object:= Ō;
      Φ.DS:= DS;
      Φ.Selection:= Ç;
      For each Φ.Object, Φ.Projection, Φ.Selection
         Search for matching objects and attributes in OMMetaData, AMMetaData, and
            TSMetaData;
         Replace Φ for all matching objects and attributes;
   End { GenSubQ };

   Function CreateJoin(Φ.Object[α], Φ.Object[β]);
   {Create a relationship condition.}
   Begin { CreateJoin }
      For each pair of Φ.Object[α] & Φ.Object[β]
         Φ.Join[m]:= Φ.Object[α].ref_key, "=",Φ.Object[β].ref_key;
   End { CreateJoin };

   Function ConvF(attr_val, fr_context, to_context);
   {Convert different semantics.}
   Begin { ConvF }
         Call the related conversion function in CVMetaData
```

```
            If fr_context = default then
                ConvF := attri_val, CVoperator,  CVfactor;
            Else if to_context = default then
                ConvF := attri_val, CVreverse,  CVfactor;
            Else Error Message 'CVMetaData needs to be maintained."
    End { ConvF };

Begin { QTransform }
    {Generate subqueries for all sources indicated in the user-requested query (θ).}
    For all Đ[i]
        GenSubQ(Đ[i]);

    {Create relationship conditions if two objects have association.}
    IF more than one object stated in from_clause Then
        CreatJoin(Φ.Object[α], Φ.Object[β]);

    {Convert attribute values if semantic contexts are different.}
    For each attribute with context specified;
        Check the constraint information in DSMataData
            If any attributes have contexts different from specified in the query
                attri_val := ConvF(attri_val, fr_context, to_context);
End { QTransform }.
```

The following is an example of a user query to *DS1* and *DS2* data sources. Users defined *Staff.salary* in Australian dollars and on yearly basis.

```
Select          Staff.id, Staff.salary(currency="AUD", period="yearly")
From            Staff
In              DS1, DS2
Condition       Staff.salary(currency="AUD", period="yearly") < 50000;
```

After the query decomposition and transformation process, two subqueries are generated. The first subquery is:

```
SELECT          Staff.id, Staff.salary(currency="USD", period="yearly")
FROM            Staff
IN              DS2
CONDITION       Staff.salary(currency="USD", period="yearly") < 25500;
```

Due to salary in *DS2* is based on US dollars (Appendix J), the conversion is required to convert *"AUD"* quoted in the user query to *"USD"*. As well as the second subquery to *DS1*, *Staff.salary* has to be converted to *"monthly"*.

```
SELECT       Staff.id, Staff.salary(currency="AUD", period="monthly")
FROM         Staff
IN           DS1
CONDITION    Staff.salary(currency="AUD", period="monthly") < 4166.67;
```

### 6.2.3 Creating a Pre-defined Template Process

From a user-requested query, it has been specified which attributes of an object users want to be shown in the result. QTA is responsible for creating an MDRS template as a basis for incorporating results from multiple data sources to this template. This MDRS template represents the semantic context as predefined references for other components that deals with conversion to determine which contexts of an attribute should be presented to users, so that the component data sources set it as the target context to produce the final query result. Without a predefined template, results from multiple data sources with both different structures and semantic contexts will be more complicated to resolve straight away. Thus, the template has to be set in prior as the target that all data have to fill in suggestively.

*TemplCreate(Ä),* is the process of the predefined template creation, while *Ä* is an array of attributes specified in *select_clause*.

```
Process TemplCreate(Ã);
{Create a pre-defined MDRS template.}
Type    context_rec   = Record of
                 name  : Context_Name;
                 value  : Context_Value;
        EndRecord;
        Project_Rec   = Record of
                 attribute  :   Attribute_Name;
                 context    :   Array of context_rec;
        EndRecord;

Var     Projection     : Array of Project_Rec;
        i, j           : Integer;

Begin { TemplCreate }
   For each attribute Ã[i];
      Projection[i].attribute:= Ã[i];
```

```
    For each context [j] of attribute Ä[i]
        Projection[i].context[j].name:= Ä[i].context[j].name;
        Projection[i].context[j].value:= Ä[i].context[j].value;
End { TemplCreate }.
```

For example, the query is

SELECT Lecturer.name, Lecturer.salary (currency="AUD", period="Monthly").

QTA prepares a pre-defined template that is:

(Lecture.name, Lecture.salary (currency="AUD", period="Monthly") )

The *Lecturer.salary* attribute and its contexts could be presented in the following 3-D MDM concept model:

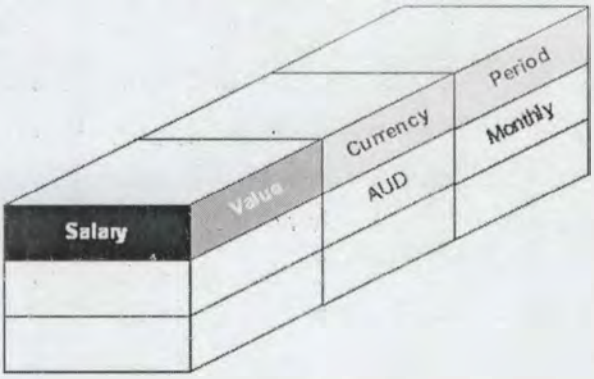


FIGURE 6.3 A 3-D MEDIATED DATA MODEL REPRESENTING MDRS TEMPLATE

From the above figure (Figure 6.3), the pre-defined template of salary has been created. It is represented by a three-dimension MDM concept model with its underlying semantic context, i.e. currency and the period of payment. The value of the query result has to be converted to conform to further contexts which are *"AUD"* currency and *"Monthly"* basis.

In summary, the main role of QTA is to decompose a user-requested query to subqueries, each of which is distributed to its related data source to query data. This task leads QTA, firstly to determine which data sources need to provide a result for, secondly to transform the query into subqueries, and thirdly to submit them to the data sources for execution by the query processing.

## 6.3 The Mediated MetaData (MMD)

Basically, metadata is "data that defines and describes other data" or "information and documentation which makes data understandable and sharable for users over time". (ISO/IEC/TC JTC 1, 2002). The ISO 11179 – Information Technology – Metadata registry, has been developed to provide an international standard for sharing and exchanging data elements: It is a significant issue in data interoperability. Metadata is highly relevant for interoperability (Conrad et al., 1999). To interoperate heterogeneous data, a strong, flexible, and incremental metadata is required. The benefits of employing metadata are: increased data sharing and data integration (Newton, 1996). In this research, the Mediated MetaData (MMD) was developed as a repository for collecting knowledge information which is necessary for the integration, such as semantic constraints, data source definitions, schemas, and conversion functions, etc. The main purpose of MMD is to provide a knowledge base to be used in resolving both schematic and semantic conflicts. In this research, MMD is divided into Schematic MetaData and Semantic MetaData.

### 6.3.1 Schematic MetaData

Data sources and their definitions initially registered by RP are reposed in MMD which is simply and meaningfully implemented by XML with its readable self-described tag characteristics. Generally, any programming or descriptive languages can be used to represent MMD. The Schematic MetaData consists of the Data Source MetaData (DSMetaData), the Object Mapping MetaData (OMMetaData), and the Attribute Mapping MetaData (AMMetaData)which contains data source schemas, object mapping, and attribute mapping information respectively. DSMetaData, OMMetaData, and AMMetaData therefore provide the required information for QTA to define the associated objects required for the requested query and to decompose the query to subqueries.

#### 6.3.1.1 The Data Source MetaData (DSMetaData)

The Data Source MetaData contains initialised component data source information recorded by the RP. The following items are the types of information relating to data sources which are contained in DSMetaData.

- Assigned name –the unique name for each data source to resolve any schematic naming conflict which might cause name crashing.
- Description –the definition of each data source.
- Location – the physical location of the data source.
- Data model and database type –knowledge for the MeDInt Mediator to determine what kinds of data models of the data source in order to take the appropriate action, for example, for sending the appropriate query language.
- Constraints –semantic information about whether the data source has any constraints.

The DSMetaData specification is as follows.

```
DSMetaData_rule       =    DataSource_rule, {DataSource_rule}
DataSource_rule       =    '{', AssignedName, DataModel, Location, Source,
                      Object_list,
                               Description, Constraint_rule, '}';
AssignedName          =    'AssignedName ', letter, {letter |decimal digit};
DataModel             =    'DataModel ', Relational | Object | Legacy;
Location              =    'Location ', letter, {letter |decimal digit};
Source                =    'SourceName ', letter, {letter |decimal digit};
Object_list           =    'Objects ', Object_identifier, {',', Object_identifier};
Object_identifier     =    letter, {letter |decimal digit};
Description           =    letter, {letter |decimal digit};
Constraint_rule       =    'Constaint ', attribute_rule;
Attribute_rule        =    Attribute_identifier, Context_rule;
Attribute_identifier=      letter, {letter |decimal digit};
Context_rule          =    Context_identifier, Context_type;
Context_identifier    =    letter, {letter |decimal digit};
Context_type          =    letter, {letter |decimal digit};
```

An example of a registered data source is given below:

```
{
AssignedName   DS2;
DataModel      object;
Location       campusO/DB;
SourceName     CampusDB;
Objects        Person, Staff, Lecturer, Student, Book, Unit;
Description    Campus database;
Constraint     Salary(Currency = "AUD");
}
```

From the above DSMetaData example, the *CampusDB* is a data source in an object data model located in *campusO/DB*. The unique name, *DS2*, is assigned to this data source. *Person, Staff, Lecture, Student, Book* and *Unit* object classes are entities in the *DS2* data source. The constraint attribute indicates that the currency used in this data source is Australian dollars.

### 6.3.1.2 The Object Mapping MetaData (OMMetaData)

In addition to data source information which has to be registered in the Mediated MetaData, the object mapping information must be gathered to identify the corresponding objects of component data sources. Object mapping information refers to the same real world objects mapped to global objects so that the global objects can be identified and referred to in the query and can be acknowledged by the components in the MeDInt Mediator unambiguously. The object mapping information is registered in the Object Mapping MetaData (OMMetadata). The main objective of OMMetaData implementation is to solve schematic naming conflicts in the entity level. The information required to be captured in OMMetaData are:

- A global object identifier – the assigning of a global identical identifier for each real-world object to achieve naming equivalence and to be indistinguishable from other collaborative components.
- Mapped data source – used to identify the component data source to which this global object maps.
- Mapped object – used to identify the object of the data source to which this global object maps to.
- Mapped object condition – used to describe mapping conditions.

The followings are the specification describing OMMetaData:

```
OMMetaData_rule          =   ObjectMapping_rule, {ObjectMapping_rule};
ObjectMapping_rule       =   '{', MappingObject, '}';
MappingObject            =   GlobalObject, MappedObject, {MappedObject};
GlobalObject             =   'GlobalObject ', letter, {letter |decimal digit};
MappedObject             =   'MappedObject ', Source, Object, {Constraint};
Source                   =   'SourceAssignedName ', letter, {letter |decimal
digit};
Object                   =   'SourceObject ', letter, {letter |decimal digit};
Constraint               =   'Constraint ', Attribule_defined;
Attribute_defined        =   Attribute_identifier, Comparison_operator,
                                  Attribute_value;
Attribute_identifier     =   letter, {letter |decimal digit};
comparison_operator      =   '=' | '>' | '<' | '>=' | '<=' | '<>';
Attribute_value          =   letter|decimal digit, {letter |decimal digit};
```

The following is an example of OMMetaData.

```
{
GlobalObject        Lecturer
MappedObject        SourceAssignedName     DS1
                    SourceObject           Staff
                    Constraint             type='L'
MappedObject        SourceAssignedName     DS2
                    SourceObject           Lecturer
}
```

The above OMMetaData example shows that a global object assigned name, *Lecturer,* which is mapped to the *staff* object class in the *DS1* data source which has the constraint of *type* = "L", and is mapped to the *Lecturer* object class in another data source, *DS2,* without any constraint.

### 6.3.1.3 The Attribute Mapping MetaData (AMMetaData)

The same attributes in multiple data sources which were assigned different names can be mapped and reposed in the Attribute Mapping MetaData (AMMetaData) to identify their correspondence. Similar to OMMetaData, attribute mapping information refers to the same real world attributes mapped to global objects first so that the global attributes can be identified and referred to in the query and can be acknowledged by the components in the MeDInt Mediator. The main objective of AMMetaData implementation is to solve schematic naming conflicts in the attribute level. The information required to be captured in AMMetaData are:

- A global attribute identifier is assigned as a unique name of a group of the same real-world attributes from multiple data sources to achieve naming equivalence and to be indistinguishable from other collaborative components.
- Mapped data sources are used to identify the component data source to which this global attribute maps.
- Mapped objects are used to identify the objects of the data sources to which this global attribute maps to.
- Mapped attributes are used to identify the attributes of the data sources to which this global attribute maps to.
- Mapped attribute conditions are used to describe mapping conditions.

The followings are the specification describing AMMetaData:

```
AMMetaData_rule          =   AttributeMapping_rule, {AttributeMapping_rule};
AttributeMapping_rule    =   '{', MappingAttribute, '}';
MappingAttribute         =   GlobalAttribute, MappedAttribute, {MappedAttribute};
GlobalAttribute          =   'GlobalAttribute ', letter, {letter |decimal digit};
MappedAttribute          =   'MappedAttribute ', Source, Object,
                                  Attribute |Constraint;
Source                   =   'SourceAssignedName ', letter, {letter |decimal
digit};
Object                   =   'SourceObject ', letter, {letter |decimal digit};
Attribute                =   'SourceAttribute ', letter, {letter |decimal digit};
Constraint               =   Attribute_identifier, Comparison_operator,
                                  Attribute_value;
Attribute_identifier     =   letter, {letter |decimal digit};
comparison_operator      =   '=' | '>' | '<' | '>=' | '<=' | '<>';
Attribute_value          =   letter|decimal digit, {letter |decimal digit};
```

The following is an example of AMMetaData.

```
{
GlobalAttribute     Student.Name
MappedAttribute     SourceAssignedName     DS2
                    SourceObject           Student
                    SourceAttribute        fname+lname
}
```

### 6.3.2 Semantic MetaData

The Mediated MetaData is intended not only for serving the schematic conflict resolution but also semantic conflict resolution by applying aliases to resolve semantic naming conflicts, and by acting as a library of functions collecting conversion functions to resolve scaling conflicts.

To resolve semantic conflicts and provide meaningful information exchange among data sources, the semantic contexts of data need to be considered (Sciore et al., 1994). The implicit context information has to be identified explicitly to share among heterogeneous sources. For example, product price is normally represented only by a real number 120.50. If it is coded by US dollars, without a semantic context, it could be compared incorrectly to 146.78 Australian dollars in another source. Both figures need to be explicitly specified in their currency in addition to its value. Then, *120.50(Currency=“USD”)* can be compared correctly to *146.78(Currency=“AUD”)* from another data source by the facilitation of conversion functions. Therefore, attribute values in different representations or contexts, can be compared by converting them into the same semantic context before comparing their values. If the conversion functions are not available, it can be implied that they have not been defined in advance, so it is impossible to convert the sum because of lack of information. Therefore, the semantic contexts and conversion information must be explicitly defined for distinct representations in multiple data sources. Once the system needs to integrate heterogeneous semantic values, it has to consult the Semantic MetaData to homogenise the data.

In this study, the Semantic MetaData can be classified into two types, Thesaurus MetaData (TSMetaData) and Conversion MetaData (CVMetaData).

### 6.3.2.1 The Thesaurus MetaData (TSMetaData)

The 3-D semantic model has been proposed in this study to represent differences in semantic values, i.e. representation conflicts, by gaining the advantage of aliases to define corresponding domains. Aliases are collected in the Thesaurus MetaData (TSMetaData). Whenever the system has to integrate heterogeneous semantic values, it consults this agent to homogenise the data. For example, days in a week can be represented in numerous ways:

```
Days of week = {1, 2, 3, 4, 5, 6, 7}
Days of week = {Sun, Mon, Tue, Wed, Thu, Fri, Sat}
Days of week = {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}
```

This semantic heterogeneity could be modelled as a general tree (Figure 6.4) grouping the same meaning aliases. Then, XML documents which are based on the object-oriented model which is best for describing schema and semantic of objects in the real-world are capable to collect these aliases.

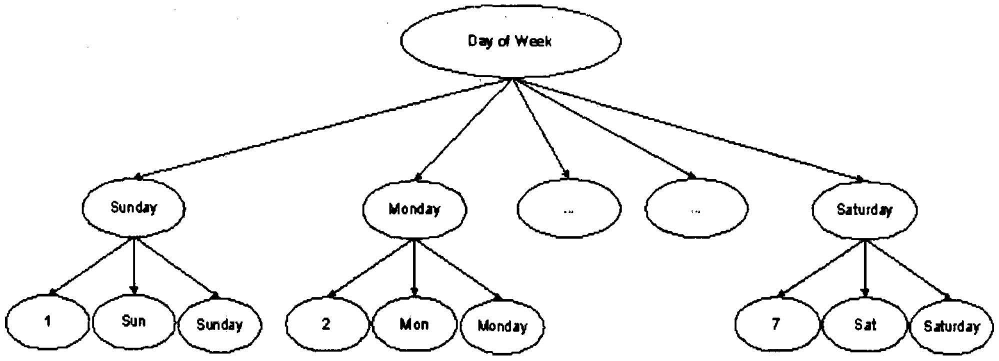


FIGURE 6.4 AN ALIAS TREE

The following is the TSMetaData specification syntax.

```
TSMetaData_rule       =    TS_rule, {TS_rule};
TS_rule               =    '{', TSMapping, '}';
TSMappingO            =    GlobalCategory, MappedInfo, {MappedInfo};
GlobalCategory        =    'GlobalObject ', letter, {letter |decimal digit};
```

```
MappedInfo          =    Default, Aliases;
Default             =    'Default ', letter, {letter |decimal digit};
Aliases             =    'Aliases { ', Alias, {Alias}, ' }';
Alias               =    'Alias ', letter, {letter |decimal digit};
```

The following is an example of TSMetaData.

```
{
GlobalCategory      Days
MappedInfo          Default                  Sunday
                    Aliases
                        {
                             Alias                  1
                             Alias                  Sun
                             Alias                  Sunday
                        }
MappedInfo          Default                  Monday
                    Aliases
                        {
                             Alias                  2
                             Alias                  Mon
                             Alias                  Monday
                        }
MappedInfo          Default                  Tuesay
                    Aliases
                        {
                             Alias                  3
                             Alias                  Mon
                             Alias                  Monday
                        }
MappedInfo ...
}
```

```
<MetaData>
    <DayOfWeek>
        <Day name="Sunday">
            <alias>1</alias>
            <alias>Sun</alias>
            <alias>Sunday</alias>
        </Day>
        <Day name="Monday">
            <alias>2</alias>
            <alias>Mon</alias>
            <alias>Monday</alias>
        </Day>
        .
        .
        .
        <Day name="Saturday">
            <alias>7</alias>
            <alias>Sat</alias>
            <alias>Saturday</alias>
        </Day>
    </DayOfWeek>
</MetaData>
```

### 6.3.2.2 The Conversion MetaData (CVMetaData)

Conversion plays a significant role in the data integration of heterogeneous sources, especially when data are represented in different contexts. Query results with varied semantic contexts are meaningless if the results cannot be compared for analysis or decision-making. This is why a library of conversion functions is necessary when the interoperation of data represented differently among heterogeneous sources is required. The MeDInt architecture encompasses the Conversion MetaData (CVMetaData) to provide conversion knowledge. The major objective of CVMetaData is resolving scaling conflicts. A conversion function will be invoked when the same real world attributes from multiple data sources with different semantic contexts are included in the user-requested query. For example, a weight attribute in one system is collected in kilograms (kgs), but in another data source it is collected in grams (gms). To interoperate them, a conversion is required to transform weight values from grams to kilograms or from kilograms to grams depending on the unit requested in the query. The following is the CVMetaData specification syntax.

```
CVMetaData_Rule     =    CVFunction, {CVFunction};
CVFunction          =    CVF_identifier, 'Default = ', DefaultContext, CVFbody;
CVF_identifier      =    letter, {letter |decimal digit};
DefaultContext      =    letter, {letter |decimal digit};
CVFbody             =    CVto, CVfactor, CVoperator, CVreverse;
CVto                =    'CVto ', letter, {letter |decimal digit};
CVfactor            =    'CVfactor ', letter |decimal digit, {letter |decimal
digit};
CVoperator          =    'CVoperator ', '+' | '-' | '*' | '/';
CVreverse           =    'CVreverse ', '+' | '-' | '*' | '/';
```

The following is an example of CVMetaData for resolving different unit of measurements.

```
{Weight_cnv         Default                   Kgs
                        {
                            CVto          gms
                            CVfactor      1,000
                            CVoperator    *
                            CVreverse     /
                        }
                        {
                            CVto          mgs
                            CVfactor      1,000,000
```

```
                                        CVoperator          *
                                        CVreverse      /
                                  }
}
```

From the CVMetaData specification above, the default unit of weight used in the integration system is kilograms. The conversion factors are defined based on the standard unit of measurement used in the integration system, so the conversion factor from one kilogram to grams is multiplying by 1,000 and to milligrams is multiplying by 1,000,000. In the reverse conversion, from grams to kilograms, the same conversion factor can be used, but using the division operator instead of the multiplication.

For example, to interoperate *Weight= 50(unit=“kgs”)* to *Weight= 49999(unit=“gms”)* from multiple data sources which are in different contexts, immediate comparison cannot occur. If the context requested in the query is *kgs,* Weight_cnv(kgs=>“gms”) will be invoked to transform *49999(unit=“gms”)* to *49.999(unit=“kgs”)* to provide the same semantic context as requested. *50(unit=“kgs”)* does not need to be converted because it is in the same unit as the requested context. Then, the values of *49.999(unit=“kgs”)* and *50(unit=“kgs”),* which have the same semantic context, can be compared or interoperated. On the other hand, if the required conversion function cannot be found, this means no conversion factor is available for these attributes; the context information should be attached to its values on the query results so that the semantic differences can be noticed.

The conversion of an attribute with multiple contexts needs a sequential conversion action. For example, when a salary attribute of *25000(currency=“USD”, period=“yearly”)* which represents US dollars on a yearly basis is compared with *2500(currency=“AUD”, period=“monthly”)* which represents Australian dollars on a monthly basis, multiple conversions are required to convert the currency and then the period. In this case, the conversion is non-order preserving, so it does not matter which conversion should be done first, but the priority of conversion is significant in some cases.

Therefore, it can be concluded that DSMetaData provides data source information. OMMetaData resolves schematic naming conflicts while TSMetaData resolves semantic naming and representation conflicts. Finally, CVMetaData provides conversion knowledge for the MeDInt Mediator to homogenise the scaling conflict due to different semantic contexts from multiple data sources.

## *6.4 Conflict Resolution Agent (CRA)*

After the MeDInt Mediator gets the MDRS query results from wrappers, the model heterogeneity has been resolved. However, Schematic and Semantic Heterogeneities have not been handled. The Conflict Resolution Agent (CRA) has this responsibility. To deal with both schematic and semantic conflicts, CRA simply applies each MDRS result set to the pre-defined template. This pre-defined template is created from the query. Thus, a varied result structure will be transformed to the structure of the pre-defined template. This means that structural conflicts have been resolved. In addition, different semantic contexts will be homogenised in this stage to have a context compatible with the template, so CRA resolves problems with semantic contexts such as scaling conflicts. However, naming conflicts in the semantic level may still remain, but can be handled by aliases in TSMetaData.

### 6.4.1 Applying MDRS Results to the Pre-defined Template

After CRA has received the MDRS result sets from the wrappers, CRA can apply each MDRS instance to its predefined template to resolve schema and semantic conflicts.

For example, given the following:

(Lecturer.fname, Lecturer.lname, Lecturer.salary (currency="AUD", period="Monthly") ),

it could be represented visually by an example of 3-D MDM as shown in Figure 6.5.

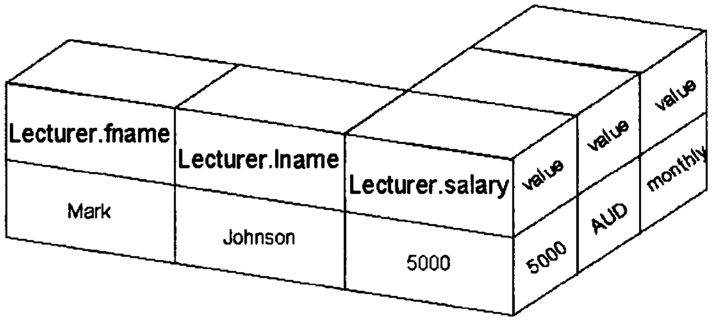


FIGURE 6.5 REPRESENTATION OF ATTRIBUTES AND SEMANTIC CONTEXTS

The role of CRA is to transform the values of query results corresponding to the structure and semantic contexts of the pre-defined template. For example, if the MDRS results of *Lecturer.salary* are not *"AUD"* currency or *"monthly"* period, it is necessary to convert these into the pre-defined semantic context during this process.

Assume that the first MDRS is

{ (Lecturer.fname, Lecturer.lname, Lecturer.salary (currency="USD", period="yearly") ) }

And the second MDRS is

{ (Lecturer.name, Lecturer.salary (currency="AUD", period="monthly") ) }
which name = (fname, lname)

CRA needs to apply different structures of the MDRS results from the wrappers to the predefined template. The conflict resolution method for the first MDRS result is the value of *Lecturer.salary,* which is in *"USD"* currency on a *"yearly"* basis and needs to be converted to *"AUD"* currency on a *"monthly"* basis by consulting CVMetaData. The second set of MDRS results also needs a conversion function to break *Lecturer.name* into *Lecturer.fname* and *Lecturer.lname.* Then, both sets of MDRS results can be filled into the template. Finally, the structural conflicts and semantic conflicts will be resolved.

*ApplTemp*$(\rho, \tau, \theta)$ is the process of applying a set of MDRS results $(\tau)$ from a data source $\alpha$ to the predefined template, where $\rho$ is the predefined template created from

```
Process ApplTemp(ρ, τ, θ);
{Apply MDRSs to fit into the pre-defined template.}
Type      context_rec   = Record of
                    name  : Context_Name;
                    value : Context_Value;
          EndRecord;
          Project_Rec   = Record of
                    attribute  :  Attribute_Name;
                    context    :  Array of context_rec;
          EndRecord;

Var       Projection        : Array of Project_Rec;
          AttrConstraint    : Array of Project_Rec;
          RMDRS             : Record of Projection;
          i, j              : Integer;

   Function ConvF(attr_val, fr_context, to_context);
   {Convert different semantics.}
   Begin { ConvF }
          Call the related conversion function in CVMetaData
             If fr_context = default then
                ConvF := attri_val, CVoperator,  CVfactor;
             Else if to_context = default then
                ConvF := attri_val, CVreverse,  CVfactor;
             Else Error Message 'CVMetaData needs to be maintained."
   End { ConvF };

Begin { ApplTemp }
   Fill τ in RMDRS;
   Get AttrConstraint from DSMetaData.constraint;
   Attach AttrConstaint to RMDRS;
   Check each attribute in RMDRS against ρ;
      If unmatched semantic contexts are found Then Begin
          Attr_val := ConvF(attr_val, RMDRS.context, ρ.context);
          Replace RMDRS.context with ρ.context;
      End;
End { ApplTemp }.
```

From *ApplTemp(ρ, τ, θ)*, the set of results returned from the wrapper does not have any semantic context attached. Constraints retrieved from DSMetaData are thus necessary to create a new semantic data set before comparing its semantic contexts with the pre-defined template in order to convert result values to have the semantic contexts conforming to the semantic contexts required by the user.

For example, the following is the set of results from *DS1*.

```
{("2158015", 3750.00(currency="AUD", period="monthly)),
("4125101",2125.00(currency="AUD", period="monthly))}
```

It will be applied to fit the pre-define template.

```
(Staff.id, Staff.salary (currency="AUD", period="yearly") )
```

*Staff.salary* needs to be converted to "yearly" basis according to the pre-defined template. The following is the set of results after the *ApplTemp(ρ, τ, θ)* process.

```
{("2158015", 45000.00(currency="AUD", period="yearly")),
"4125101",25500.00(currency="AUD", period="yearly"))}
```

## 6.5 The Consolidation Processor (CP)

The Consolidation Processor (CP) as a data integrator consolidates the conflict-resolved MDRS result sets which have structure and semantic contexts corresponding to the predefined template. In other words, model, schematic, and semantic conflicts have already been resolved. Thus, the result sets are structurally equivalent. At this point, the sets of conflict-resolved results can be integrated simply by set operations.

### 6.5.1 Integrating the Mediated Data Representation Structures

After CRA applies the MDRS results according to the predefined template format, all result sets then conform to each other and also to the requested query both in their schemas and semantics. CP integrates only the structurally and semantically equivalent conflict-resolved sets by appropriate set operators, for example, the union or interception operators, depending on the condition of the query.

*Integrate(υα, υβ, Ω)* is the process of integrating conflict-resolved MDRS result sets, where *υα* is a conflict-resolved set from data source *α,* and *υβ* is from data source *β,* and *Ω* is a relational algebra.

```
Process Integrate(υα, υβ, Ω);
{Integrate two conflict-resolved MDRS result sets.}
Type    context_rec   = Record of
                  name  : Context_Name;
                  value  : Context_Value;
        EndRecord;
        Project_Rec   = Record of
                  attribute  :   Attribute_Name;
                  context    :   Array of context_rec;
        EndRecord;

Var     Projection         : Array of Project_Rec;
        υα, υβ             : Record of Projection;
        Ω                  : relation algebra;

Begin { Integrate }
   Case Ω is 'U'
      Union(υα, υβ);
   Case Ω is '∩'
      Intersect(υα, υβ);
   Case Ω is 'X'
      Cartesian(υα, υβ);
   Case Ω is '∞'
      Join(υα, υβ);
End { Integrate }.
```

## 6.6 The Rendering Agent (RA)

After all results from multiple data sources have been integrated by CP, the Rendering Agent automatically generates the integrated results to the users. To achieve flexibility, the Hyper Text Markup Language (HTML) format has been chosen here to present the final query results.

### 6.6.1 Generating the Integrated Results

The MDRS integrated result has to be transformed to produce output to users in HTML. Because XML documents have been used to represent the integrated results in the MeDInt architecture, rendering from XML to HTML is quite simple.

Cascading Style Sheets (CSS) and eXtensible Style Language (XSL) are alternative approaches (Morrison et al., 2000). A CSS or an XSL can be defined to generate an HTML document from an XML document. Some XML parser software also provides this feature. Therefore, the implementation of the RA will not be discussed in detail in this study.

## 6.7 Summary

The MeDInt Mediator is a layer between clients and wrappers. Its main functions include the decomposition of the user query into subqueries, provision of knowledge about mapping information, resolution of conflicts, and consolidation of data. It is independent from data sources and does not have to deal with the data model heterogeneities itself. The mediator deals only with Schematic and Semantic Heterogeneities. MDM is the data model used in the MeDInt Mediator.

# CHAPTER 7 - WRAPPERS

The MeDInt Mediator discussed in the previous chapter is responsible for transforming a query to subqueries to request data and for integrating heterogeneous data returned from multiple data sources. To reduce complexity, the MeDInt Mediator does not have to communicate with data sources directly. If that were the case, it would have to handle heterogeneous data definition languages and heterogeneous query languages in addition to dealing with conflict resolution. In this study, wrappers take this responsibility by acting as intermediate translators communicating with both the MeDInt Mediator and component data sources even though they may be in different data models.

## *7.1 The Design of Wrappers*

The MeDInt Mediator cannot communicate to multiple data sources directly because of the data model heterogeneities of multiple data sources including different schema definitions, different query languages and different data representation structures. Interpreters are necessary to translate these to the Mediated Data Model (MDM) which is the common data model used in the MeDInt architecture. MDM consists of the Mediated Data Definition Language (MDDL), the Mediated Query Language (MQL), and the Mediated Data Representation Structure (MDRS) which are the common data definition, query language and data representation respectively. This study introduces wrappers to satisfy the above purpose A wrapper is associated with each data source to translate source schemas into MDDL schemas, MQL subqueries to source specific queries, and data from data sources to MDRS data objects.

Wrappers, in this research, act as translators, including firstly *schema definition translators* which accommodate queries by translating heterogeneous schema definitions into MDDLs, secondly *query translators* which translate MQLs used in the MeDInt Mediator into specific data source query languages, and thirdly *data* *content translators* which translate data in disparate representations into MDRSs. Each wrapper is composed of its own Schema Translation Processor (STP), Query Translation Processor (QTP) and Data Translation Processor (DTP) serving functions described previously. Briefly, a wrapper is responsible for translating between the data model used in a data source and MDM used in the MeDInt Mediator. Therefore, only one additional wrapper implementation is required for a pair of a particular data model and MDM, when a new data source in a different data model is added to the integration system. Let us say, if there are *m* data sources to be integrated, and from such data sources, there are *n* (which *n* <= *m*) different data models, there will be only *n* wrappers to be implemented. This is more beneficial when comparing it with the tradition translation approach in which *m*(m-1)* translators are required. It will be exponentially more efficient when there are many data sources (*m* increases) to be integrated and more than one data model (*n*> *1*, where *n* is a natural number).

The algorithms of the components of each wrapper are different. They depend on what kind of data model used in the data source. This study investigates developing wrappers for the relational data model called RWrap, for the object-orientea data model called OWrap, and for legacy text files called LWrap.

## 7.2 Wrapper Components

There are three components in each wrapper: a Schema Translation Processor (STP), a Query Translation Processor (QTP), and a Data Translation Processor (DTP).

### 7.2.1 Schema Translation Processor (STP)

The MeDInt Mediator needs schema definitions from data sources as information for decomposing and transforming the query. To reduce complexity, the Mediator was not designed to get the schema definitions from heterogeneous data sources. Thus wrappers have the responsibility to communicate with each source to capture schema definitions and to provide them in a format that can be recognised by the MeDInt Mediator.

A Schema Translation Processor (STP) is responsible for translating the data definitions from source schema definitions into MDDL definitions which can be employed by the QTA, a component of the MeDInt Mediator when decomposing the user's submitted query to subqueries. An STP supplies only the schema definitions necessarily requested by the MeDInt Mediator, and will not supply all object schema definitions of component data sources to the MeDInt Mediator. This results in great benefits in terms of time and resource efficiency. Furthermore, it has been designed to be suitable for dynamic systems whose source schemas could be changed frequently.

An STP gets object schema definitions from data sources which may be represented by Data Definition Language (DDL) in the relational data model, by Object Definition Language (●DL) in the object data model, or by other definition languages in other data models. STPs transform this variety data definitions to MDDLs the syntax of which is provided in Chapter 5 in the Extended Backus-Naur Form (EBNF). Only the relational data model, the object data model and text legacy systems have been studied in this research, so there are three algorithms of STPs. *RSchmTransl(Ši,Ŏj)* is an algorithm for the relational data model, *OSchmTransl(Ši,Ŏj)* is for the object data model, and *LSchmTransl(Ši,Ŏj)* is for legacy text files. Ši is data source i and Ŏj is object j in the data source i.

### 7.2.1.1 STP Algorithm for the Relational Data Model

*RSchmTransl(Ši,Ŏj)* will generate an object schema definition tree (Figure 7.1).

```
Process RSchmTransl(Ši,Ŏj);
Type    SchmDefRec is record of SchmName, SchmDesc, AttrSet, RelSet, KeySet;
        AttrSet    :  set of AttrRec;
        RelSet     :  set of RelRec;
        KeySet     :  set of KeyRec;
        ObjSchmTr:    Tree;
Var     SchmDef :     SchmDefRec;
        SchmName:     String;
        SchmDesc:     String;
        VattrSet  :   AttrSet;
        VrelSet    :  RelSet;
        VkeySet  :   KeySet;
        Đi       :   ObjSchmTr;

    Function FetchSchm(Ši,Ǒj);
    Begin { FetchSchm }
        FOR SchmDef:
            SchmName:= (Ši.Ǒj);
            RETRIEVE description from Ši.Ǒj to SchmDesc;
            RETREIVE attribute from Ši.Ǒj to VattrSet until no more attribute;
            RETREIVE relationship from Ši.Ǒj to VrelSet until no more relation;
            RETREIVE key from Ši.Ǒj to VkeySet until no more key;
    End { FetchSchm };

    Function TranslSchm(SchmDef):
    Begin { TranslSchm }
        CREATETREE Đi;
            CREATE root node from SchmDef.SchmName, SchmDef.ScheDesc;
            CREATE attribute child node;
                CREATE child node from VattrSet until no more attribute;
            CREATE relationship child node;
                CREATE child node from VrelSet until no more relationship;
            CREATE key child node;
                CREATE child node from VkeySet until no more key;
    End { TranslSchm };

Begin { RSchmTransl }
    FetchSchm(Ši,Ǒj);
    TranslSchm(SchmDef);
    Return Đi;
End { RSchmTransl }.
```

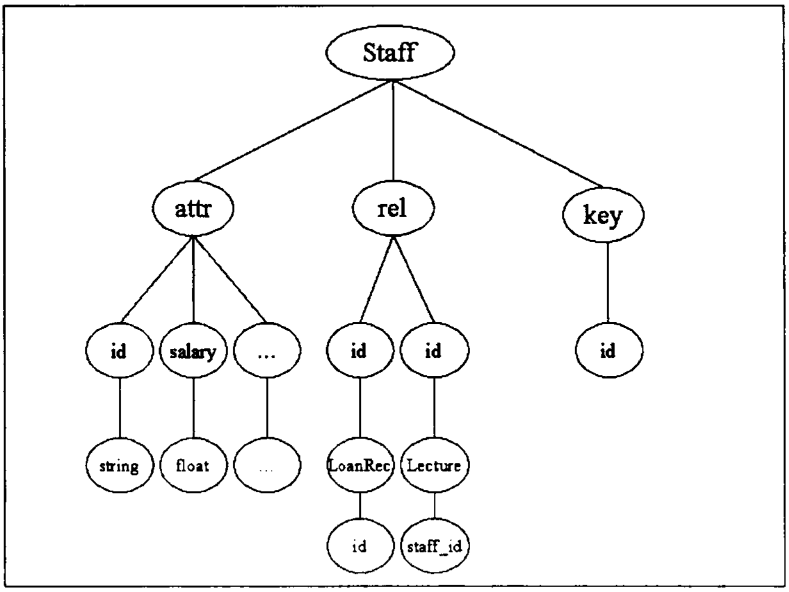


FIGURE 7.1 AN EXAMPLE OF AN EXPORTED SCHEMA DEFINITION TREE BY RWRAP

Figure 7.1 shows an example of an object schema definition tree exported by the RWrap via the request *(RSchmTransl(Ši,Ŏj),* while *Ši* is a data source name and *Ŏj* is the object *'Staff'*) from the MeDInt Mediator.

From the above *'Staff'* data definition tree, an MDDL definition can be simply generated.

```
Staff  =  {
          attribute
              id           string
              salary       float
              ...
          relationship
              id           LoanRec.id
              id           Lecture.staff_id
          key
              id;
          }
```

### 7.2.1.2 STP Algorithm for the Object-oriented Data Model

OSchmTransl(Ši,Ŏj) will create an object schema definition tree (Figure 7.2).

```
Process OSchmTransl(Ši,Ŏj);
Type  SchmDefRec is record of SchmName, SchmDesc, SubTSet, AttrSet, KeySet;
      SubTSet  :  set of string;
      AttrSet  :  set of AttrRec;
      KeySet   :  set of KeyRec;
      ObjSchmTr: Tree;
```

```
Var     SchmDef :   SchmDefRec;
        SchmName:   String;
        SchmDesc:   String;
        VsubtSet :  SubtSet;
        VattrSet  : AttrSet;
        VkeySet   : KeySet;
        Đi        : ObjSchmTr;

   Function FetchSchm(Ši,Ōj);
   Begin { FetchSchm }
      FOR SchmDef:
         SchmName:= (Ši.Ōj);
         RETRIEVE description from Ši.Ōj to SchmDesc;
         RETRIEVE subtype from Ši.Ōj to VsubtSet until no more subtype;
         RETREIVE attribute from Ši.Ōj to VattrSet until no more attribute;
            IF attribute is related to other attribute THEN
               RETRIEVE related attribute until no more related attribute
         RETREIVE relationship from Ši.Ōj to VrelSet until no more relation;
         RETREIVE key from Ši.Ōj to VkeySet until no more key;
   End { FetchSchm };

   Function TranslSchm(SchmDef):
   Begin { TranslSchm }
      CREATETREE Đi;
         CREATE root node from SchmDef.SchmName, SchmDef.ScheDesc;
         CREATE subtype child node;
         CREATE attribute child node;
            CREATE child node from VattrSet until no more attribute;
               IF there is related attribute THEN
                  CREATE child node from VsubtSet until no more related attribute;
         CREATE key child node;
            CREATE child node from VkeySet until no more key;
   End { TranslSchm };

Begin { OSchmTransl }
   FetchSchm(Ši,Ōj);
   TranslSchm(SchmDef);
   Return Đi;
End { OSchmTransl }.
```

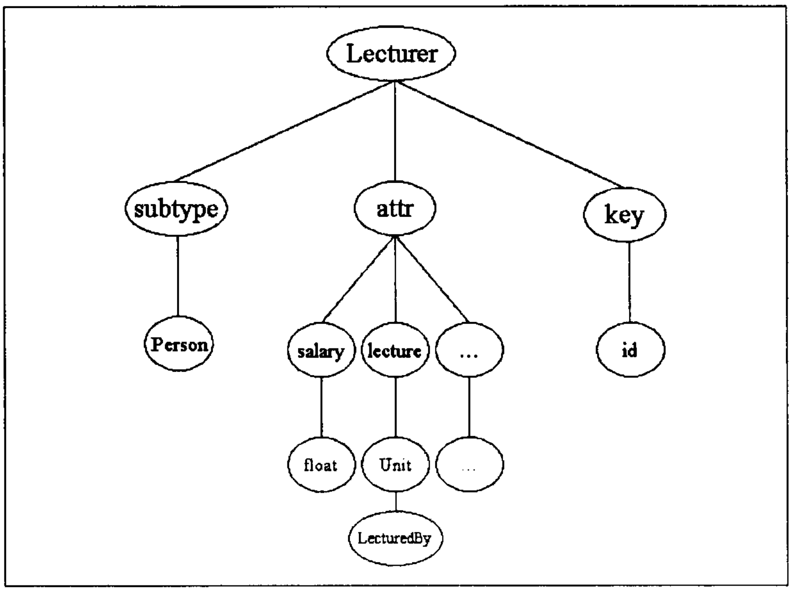


FIGURE 7.2 AN EXAMPLE OF AN EXPORTED SCHEMA DEFINITION TREE BY OWRAP

Figure 7.2 shows an example of an object schema definition tree exported by the OWrap by the request (*OSchmTransl(Ši, Ǒj),* while *Ši* is a data source name and *Ǒj* is the object *'Lecturer'*) from the MeDInt Mediator.

From the above *'Lecturer'* data definition tree, an MDDL definition can be simply generated.

```
Lecturer   =  {
              subtype
                 Person
              attribute
                 salary       float
                 lecture      Unit.lecturedBy
                 ...
              key
                 id
              }
```

### 7.2.1.3 STP Algorithm for Legacy File Processing Systems

The characteristics of legacy file processing systems are quite different from those of the relational data model and the object data model in database management systems. They do not have metadata, so schema information cannot be drawn like the previous two data models. The STP of the LWrap thus takes advantage of only the first row of text files to indicate the name of each field by ignoring data types. Moreover, the data in each file in legacy file processing systems are separated (Kroenke, 2002), so no relationship information is involved.

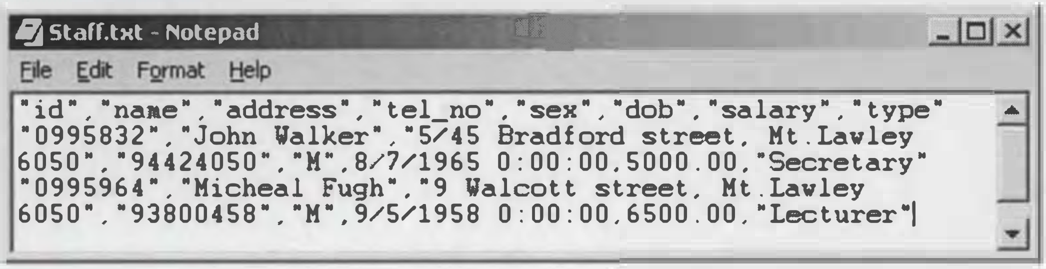
```
Process LSchmTransl(Ši,Ŏj);
Type   SchmDefRec is record of SchmName, AttrSet;
       AttrSet    :  set of AttrRec;
       ObjSchmTr:  Tree;
Var    SchmDef :   SchmDefRec;
       SchmName:  String;
       VattrSet  :  AttrSet;
       Đi         :  ObjSchmTr;

   Function FetchSchm(Ši,Ŏj);
   Begin { FetchSchm }
      From the first row of Ŏj in Ši
      FOR SchmDef:
         SchmName:= (Ši.Ŏj);
         RETREIVE attribute from Ši.Ŏj to VattrSet until no more attribute;
   End { FetchSchm };

   Function TranslSchm(SchmDef):
   Begin { TranslSchm }
      CREATETREE Đi;
         CREATE root node from SchmDef.SchmName
         CREATE attribute child node;
            CREATE child node from VattrSet until no more attribute;
   End { TranslSchm };

Begin { LSchmTransl }
   FetchSchm(Ši,Ŏj);
   TranslSchm(SchmDef);
   Return Đi;
End { LSchmTransl }.
```

An example of a legacy text file is shown in Figure 7.3.



```
"id","name","address","tel_no","sex","dob","salary","type"
"0995832","John Walker","5/45 Bradford street, Mt.Lawley
6050","94424050","M",8/7/1965 0:00:00,5000.00,"Secretary"
"0995964","Micheal Fugh","9 Walcott street, Mt.Lawley
6050","93800458","M",9/5/1958 0:00:00,6500.00,"Lecturer"
```

FIGURE 7.3 AN EXAMPLE OF A LEGACY TEXT FILE

Figure 7.4 shows a schema definition tree exported by the LWrap from the previous example (Figure 7.3) by the request (*LSchmTransl(Ši,Ōj)*, while *Ši* is a data source name and *Ōj* is the file *'Staff'*) from the MeDInt Mediator.

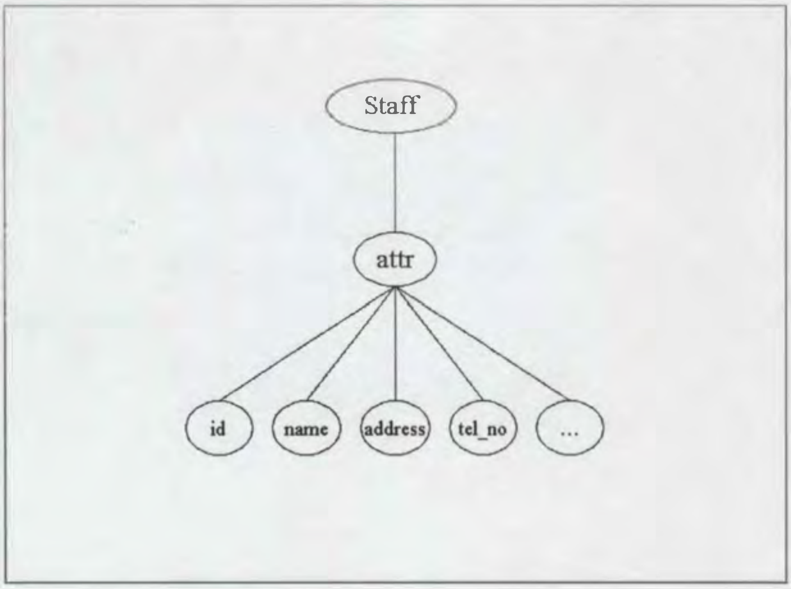


FIGURE 7.4 AN EXAMPLE OFAN EXPORTED SCHEMA DEFINITION TREE BY LWRAP

From the above *'Staff'* data definition tree (Figure 7.4), an MDDL definition can be simply generated.

```
Staff    =  {
                 attribute
                     id          string
                     name        string
                     address     string
                     tel_no      string
                 }
```

### 7.2.2 Query Translation Processor (QTP)

Due to the complexity of dealing with heterogeneity, the processes of conflict resolution and query translation and transformation have been split. The MeDInt Mediator handles the heterogeneity both on the query and the data. To the query, the MeDInt Mediator decomposes and transforms it to MQL specifications before passing the decomposed- and transformed-subqueries to wrappers. Wrappers do not have to deal with heterogeneity, but only translate subqueries to the query languages, which can be operated by the connected data sources.

From MDDLs of associated objects, a QTP translates MQL submitted from QTA to a specific query language, for example, Structured Query Language (SQL) and Object-oriented Query Language (OQL), etc, that each source can execute. QTPs sense what query language should be generated from DSMetaData.

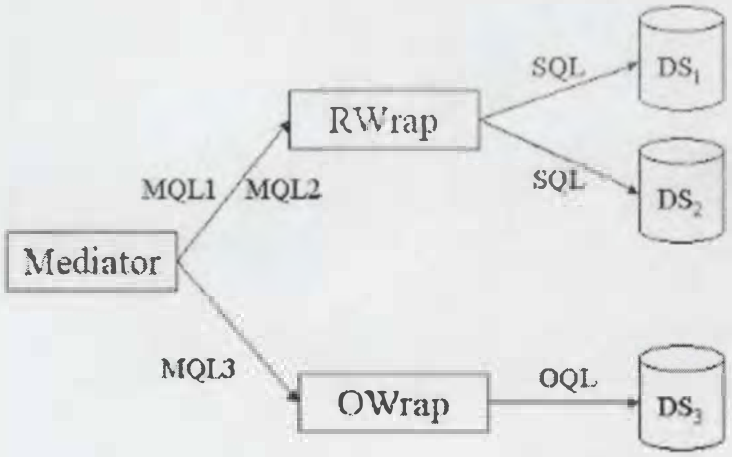


FIGURE 7.5 QUERY DISTRIBUTION AND TRANSLATION

From Figure 7.5, assume that the MeDInt Mediator submitted $MQL_1$ to $DS_1$ and $MQL_2$ to $DS_2$ passing through RWrap since $DS_1$ and $DS_2$ are relational models using SQL as their query language. The MeDInt Mediator also submits $MQL_3$ to $DS_3$ passing through OWrap because $DS_3$ is an object-oriented model using OQL as its query language. $MQL_1$ and $MQL_2$ will be translated by the QTP of the relational wrapper to SQL which is the query language used in $DS_1$. Also $MQL_3$ has to be translated by the QTP of the object wrapper before submitting to data sources to process the query.

The algorithm of each QTP is varied depending on what kinds of query language a QTP has to be translated into.

### 7.2.2.1 QTP Algorithm for the Relational Data Model

According to relational algebra (Date, 1990), the special relational operators are Restriction or Selection, Projection and Join (Figure 7.6). The Restriction or Selection operator extracts specified tuples from a relation. The Projection operator extracts specified attributes from a relation, while the Join operator builds a relation from two specified relations (Date, 1990).

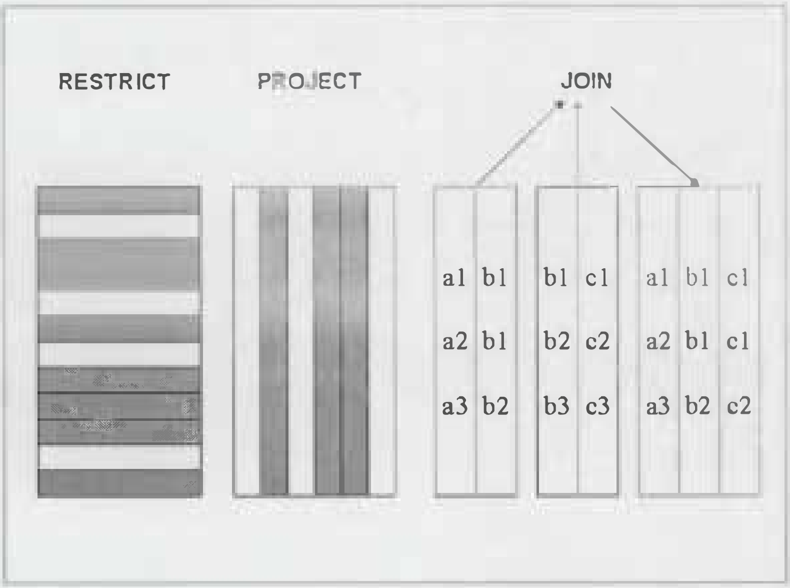


FIGURE 7.6 FUNDAMENTAL RELATIONAL OPERATORS (DATE, 1990)

Considering a basic SQL statement,

```
SELECT item(s)
FROM table(s)
[WHERE condition_expression];
```

relating to the relational algebra mentioned above, the *SELECT item(s)* clause is where the Projection operator is stated and the *WHERE condition_expression* statement is where the Restriction and Join operators can be stated.

Consider an MQL statement used in the MeDInt Mediator,

```
SELECT attribute(s) with context
FROM object(s)
IN datasource(s)
[CONDITION condition_expression with context];
```

As a result of the decomposition and transformation processes, the semantic context heterogeneities on the subqueries have been removed and each subquery thus has the same context as the associated data source. MQL subqueries submitted to the wrappers are:

```
SELECT attribute(s)
FROM object(s)
IN datasource(s)
[CONDITION condition_expression];
```

It can be noted from the MQL statement that the *SELECT attribute(s)* clause is where the Projection operator is stated and the *WHERE condition_expression* statement is where the Restriction can be stated.

By the previous comparison of both SQL and MQL statements, it is a simple task to generate an SQL statement from an MQL statement. The algorithm can be explained by the following SQLGen process.

```
Process SQLGen(χ);
Type  Φ_Rec      : Record of
                   Object        :  array[1..h] of ObjectType;
                   Projection    :  array[1..i] of AttrRec;
                   Restriction   :  array[1..j] of ConditionRec;
                   Join          :  array[1..k] of RelRec;
Var        Φ       : Φ_Rec;
           h, i, j, k      : integer;
           SQL_statement: string;

   Function CreateJoin(x);
   Begin { CreateJoin }
      For each pair of tableα & tableβ
         Φ .Join[k]:= tableα.ref_key, "=", tableβ.ref_key;
   End { CreateJoin };

Begin { SQLGen }
   For all x.From[h]
      Φ .Object[h]:= x.From[h];
```

```
    For all x.select[i]
        Φ .Projection[i]:= x.Select[i];
    For all x.Condition[j]
        Φ .Restriction[j]:= x.Condition[j];
    IF more than one object stated in FOR clause
        CreatJoin(x);
    SQL_statement =    "SELECT ", Φ .Projection[i],
                       "FROM", Φ .Object[h],
                       ["WHERE", Φ .Restriction[j]],
                       ["AND", Φ .Join[k]];
End { SQLGen }.
```

The algorithm above generates an SQL statement by

- creating Projection from attributes specified in the *SELECT* clause,
- creating objects from the *FROM* clause, and
- creating Restriction and Join from the *CONDITION* clause and relationship statements.

Note that from *IN* clause of an MQL statement, the wrappers know which data sources that subqueries should be submitted to. This QTP algorithm is only suitable for basic SQL statements. However, it can be extended to cover more complex statements.

### 7.2.2.2 QTP Algorithm for the Object-oriented Data Model

The Object Query Language (OQL) is an extension of the SQL and is similar to it. However, an object's attribute in OQL can easily be navigated by using path expressions. The MQL design is also based on the object-oriented data model which is suitable for representing the OQL. Consider a basic OQL statement,

```
SELECT list of typevar.item
FROM list of typevar type
[WHERE condition_expression];
```

The *SELECT list of typevar.item* clause is where the Projection operator is stated and the *WHERE condition_expression* statement is where the Restriction and Join operators can be stated similar to an SQL statement. Therefore, the algorithm can be explained by the following OQLGen process.

```
Process OQLGen(χ);
Type  Φ_Rec      : Record of
                      Object        :  array[1..h] of ObjectType;
                      Projection    :  array[1..i] of AttrRec;
                      Restriction   :  array[1..j] of ConditionRec;
                      Join          :  array[1..k] of RelRec;
Var      Φ            : Φ_Rec;
         h, i, j, k        : integer;
         OQL_statement: string;

   Function CreateJoin(x);
   Begin { CreateJoin }
      For each pair of tableα & tableβ
         Φ .Join[k]:= tableα.ref_key, "=", tableβ.ref_key;
   End { CreateJoin };

Begin { OQLGen }
   For all x.From[h]
      Φ .Object[h]:= x.From[h];
   For all x.select[i]
      Φ .Projection[i]:= x.Select[i];
   For all x.Condition[j]
      Φ .Restriction[j]:= x.Condition[j];
   IF more than one object stated in FOR clause
      CreatJoin(x);
   OQL_statement =    "SELECT ", Φ .Projection[i],
                      "FROM", Φ .Object[h],
                      ["WHERE", Φ .Restriction[j]],
                      ["AND", Φ .Join[k]];
End { OQLGen }.
```

### 7.2.2.3 QTP Algorithm for Legacy File Processing Systems

Querying data from legacy text files is not as simple as from database management systems because specific ad hoc coding will be required. Conversely, converting text files to other forms such as objects in a database or to XML documents is not as complex, since query languages can then be used to retrieve data. In this study, XML documents have been chosen, so the query language used to perform on XML documents is XQuery developed by the World Wide Web Consortium (*XML query* *uses cases*, 2002; *XQuery 1.0: an XML query language*, 2002). The basic syntax of XQuery is

```
FOR var IN expr
WHERE expr
RETURN expr
```

From the text file (Figure 7.3), the generated XML document *(staff.xml)* is shown below.

```
<root>
    <Staff>
        <id>0995832</id>
        <name>John Walker</name>
        <address>5/45 Bradford street, Mt.Lawley 6050</address>
        <tel_no>9442 4050</tel_no>
        <sex>M</sex>
        <dob>8/7/1965</dob>
        <salary>5000.00</salary>
        <type>Secretary</type>
    </Staff>
    <Staff>
        <id>0995964</id>
        <name>Micheal Fugh</name>
        <address>9 Walcott stree, Mt.Lawley 6050</address>
        <tel_no>93800458</tel_no>
        <sex>M</sex>
        <dob>9/5/1958</dob>
        <salary>6500.00</salary>
        <type>Lecturer</type>
    </Staff>
</root>
```

Based on the above XML document, the following query is an example of XQuery that requires *id* and *name* of staff whose *type* equals *"Lecturer"*.

```
FOR $s IN document("Staff.xml")/root/Staff
WHERE $s/type="Lecturer"
RETURN
    <Staff>
        {$s/id}
        {$s/name}
    </Staff>
```

Firstly, the query declares a variable *s* as *staff* in *root* in the *"Staff.xml"* document. The *WHERE* clause can be compared to the restriction part of the relational algebra. Elements stated in the *RETURN* clause can be compared to the projection part. Therefore, the algorithm can be explained by the following XQLGen process.

```
Process XQLGen(χ);
Type  Φ_Rec     : Record of
                      Object       :  String;
                      Projection   :  array[1..i] of AttrRec;
                      Restriction  :  array[1..j] of ConditionRec;
```

```
Var     Φ            : Φ_Rec;
        i, j         : integer;
        XQL_statement: string;

Begin { XQLGen }
   Φ .Object:= x.From;
   For all x.select[i]
      Φ .Projection[i]:= x.Select[i];
   For all x.Condition[j]
      Φ .Restriction[j]:= x.Condition[j];
   XQL_statement =   'FOR $r IN document(" ', Φ .Object, '.xml")/root/ ', Φ .Object,
                     ['WHERE $r/', Φ .Restriction[j]],
                     "RETURN",
                        '<', Φ .Object, '>',
                           '{$r/', Φ .Projection[i], '}',
                        '</', Φ .Object, '>',;
End { XQLGen }.
```

### 7.2.3 Data Translation Processor (DTP)

Data returned from heterogeneous data sources by the request of subqueries cannot be interoperated by the MeDInt Mediator instantly because they are represented in different data models. This responsibility has been given to wrappers. A Data Translation Processor (DTP), a component within a wrapper, handles this by transforming the data content received from data sources to the common data model used in the MeDInt Mediator which is the Mediated Data Representation Structure (MDRS). The MeDInt Mediator can recognise MDRSs and can take further action to solve conflicts. However, the semantic contexts of query results returned from the data source are ignored in this phase. They are attached later by the MeDInt Mediator. This step aims only to resolve the Data Model Heterogeneity of data returned from data sources.

*DataTrans($\rho$)* is a process of translating data from relational data sources to MDRS, while $\rho$ is a resultant data set from the data source.

```
Process DataTransl(ρ);
Type  DataSet  :  Set of Record;
Var   π        :  DataSet;

    Function RecTrans(ρ);
    Begin { RecTrans }
        For all attributes
            Put(π) separating each attribute by comma;
    End { RecTrans };

Begin { DataTransl }
    Repeat
        Read next record;
        RecTrans(ρ);
    Until no more record;
    Return π;
End { DataTransl }.
```

Next, an example of the different structures of data returned from two data sources is shown. The first one, $D_1$, is data structure returned from a relational data source.

```
D1          = {
            Attribute
                id      Integer
                fname   string
                lname   string
            Key
                id
            };
```

$D_2$ is data structure returned from an object data source.

```
D2          = {
            Attribute
                id      Integer
                name    struct
                        (fname string,
                        lname string)
            Key
                id
            };
```

$D_1$ should be translated into { *(id, fname, lname) }, for example,

```
{("0995547","John","Mc.Klen"),("0995550","Susan","Johnson")}
```

$D_2$ should be translated into { *(id, (fname, lname)) }, for example,

```
{("0995152", ("Jame", "Carter")),
("0994521", ("Catherine","Foster"))}
```

These two result sets will then have the MDRS format which could be sent to the MeDInt Mediator for conflicts to be resolved.

## 7.3 Summary

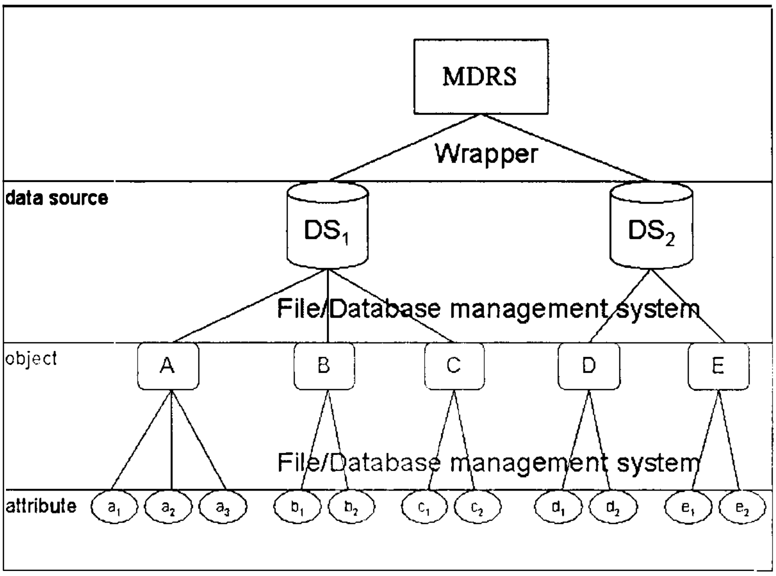


FIGURE 7.7 DATA SOURCE AND WRAPPER RESPONSIBILITY CLASSIFICATION

Wrappers are described in Figure 7.7 in terms of the responsibility of data source and wrapper management in the MeDInt framework. Objects and attributes are handled by the file/database management system of each data source. However, to be represented in MDRS objects, the data model heterogeneities have to be resolved and handled by wrappers.

This research only focuses on the relational data model, the object data model and legacy text files which are widely used in the real world. Thus, three wrappers were designed: an RWrap for the relational data model, an OWrap for the object-oriented data model, and an LWrap for legacy text files. Inside each wrapper (Figure 7.8), there are three algorithms serving as a Schema Translation Processor (STP), a Query Translation Processor (QTP) and a Data Translation Processor (DTP).

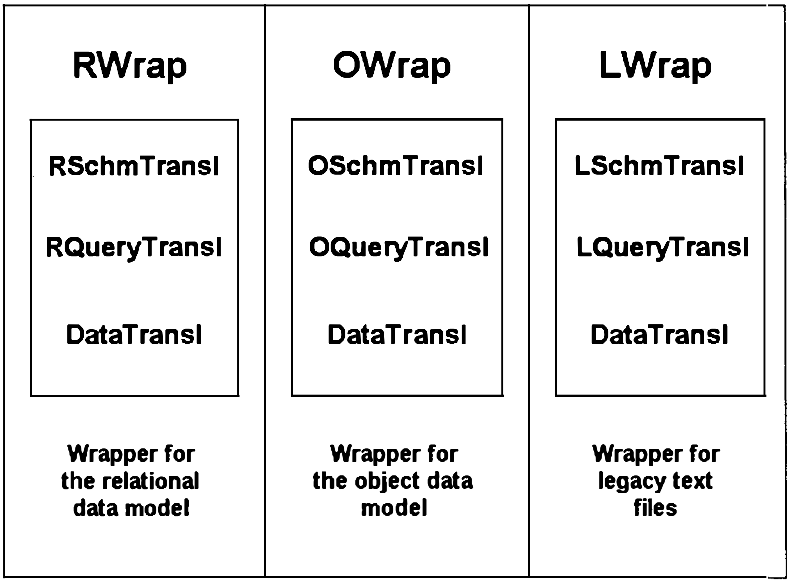


FIGURE 7.8 THREE WRAPPERS DEVELOPED IN THIS STUDY

An STP translates schemas from the data source into the Mediated Data Definition Language (MDDL). A QTP is responsible for translating the Mediated Query Language (MQL) subqueries to a specific query to be processed by each data source. A DTP gets the query result from each data source, and then translates this into the Mediated Data Representation Structure (MDRS) where each unit is a set of required object attributes or properties.

# CHAPTER 8 – SYSTEM EVALUATION AND RESULTS

The critical problem in a data integration process is the heterogeneity of component data sources. The causes of heterogeneities can be from the autonomy of data sources, different database design, and so on. Conflicts or heterogeneities between heterogeneous data sources in this study have previously been classified into three major types: Data Model Heterogeneity, Schematic Heterogeneity, and Semantic Heterogeneity. Brief descriptions are given below:

**Data Model Heterogeneity**

Data Model Heterogeneity occurs when there is a problem with data integration from multiple data sources when component data sources use different data models, for example, some may be relational data models, some may be object-oriented data models, and others may be legacy file processing systems. Data Model Heterogeneity involves using different data definition languages and manipulation languages.

**Schematic Heterogeneity**

Schematic Heterogeneities exist when the structures of same real-world objects are defined differently in their component data sources. They can be classified as:

- Naming Conflicts which include conflicts between entity-entity and attribute-attribute,
- Structural Conflicts which include entity-attribute and attribute-data,
- Generalisation/specialisation Conflicts, and
- Relationship Conflicts.

### Semantic Heterogeneity

Semantic Heterogeneities occurs when data in component data sources are represented differently. These include Naming Conflicts, Representation Conflicts, Scaling Conflicts, and Level of Abstraction Conflicts.

In this chapter, example problems of heterogeneities from a number of information systems that require integration are described. The conflicts classified previously are then resolved. The objectives are to demonstrate the integration process using the MeDInt architecture and to evaluate its correctness. Each example problem is chosen to demonstrate a different set of conflicts.

## *8.1 System Experimentation and Evaluation*

### 8.1.1 Test problem 1 – Hotel Chain Information System

The example is a Hotel Reservation Information System which provides information for travel agencies. The information systems of contacted hotels need to be interoperated. Heterogeneities have been found when integrating them. Following are the object schema definitions of component data sources only which relate to this query example.

### HOTEL CHAIN A - OBJECT-ORIENTED DATA MODEL

```
CREATE TYPE Address_type (
        Number                  CHAR,
        Street                  CHAR,
        City                    CHAR,
        State                   CHAR,
        Country                 CHAR,
        Postcode                CHAR)

CREATE Type HotelObj (
        Name                    CHAR,
        Address                 Address_type,
        Phone                   CHAR,
        Fax                     CHAR,
        Rooms                   NUMBER,
        Description             CHAR)
```

```
CREATE TYPE Loc_type (
        Building                CHAR,
        Floor                   CHAR,
        Wing                    CHAR)

CREATE TYPE Class_type (
        RoomClass               CHAR,
        NumberPersons           NUMBER)

CREATE TYPE RoomObj (
        Hotel                   HotelObj,
        Number                  CHAR,
        Location                Loc_type,
        Class                   Class_type,
        Price                   NUMBER)

CREATE RoomStatus (
        Room                    RoomObj,
        Date                    DATE,
        Status                  {checkin, checkout, available, reserved})
```

## HOTEL CHAIN B – RELATIONAL DATA MODEL

```
CREATE TABLE HOTELINFO
        (Name                   CHAR,
        Address                 CHAR,
        City                    CHAR,
        State                   CHAR,
        Country                 CHAR,
        Postcode                CHAR,
        Phone                   CHAR,
        Fax                     CHAR,
        Rooms                   NUMBER,
        Description             CHAR,
        PRIMARY KEY (Name))

CREATE TABLE ROOM
        (HotelName              CHAR,
        Number                  CHAR,
        Building                CHAR,
        Floor                   CHAR,
        Class                   CHAR,
        NumberPersons           NUMBER,
        Price                   NUMBER,
        PRIMARY KEY (HotelName, Number),
        FOREIGN KEY (HotelName) REFERENCES HOTELINFO)

CREATE TABLE STATUS
        (HotelName              CHAR,
        RoomNumber              CHAR,
        Date                    DATE,
        Status                  CHAR,
        PRIMARY KEY (HotelName, RoomNumber, Date)
        FOREIGN KEY (HotelName, RoomNumber) REFERENCES ROOM)
```

## HOTEL CHAIN C – LEGACY FILE PROCESSING SYSTEM

HOTEL(Name, Address, City, State, Country, Postcode, Phone, Fax, Rooms, Description)

ROOM (HotelName, Number, Building, Floor, Class, NumberPersons, Price)

STATUS (HotelName, RoomNumber, Date, Status)

Each data source is the data source of a hotel chain which includes a number of hotels of its chain. Hotel data sources may be served by different data models, for example, an object-oriented data model (HotelA), a relational data model (HotelB), and a legacy file processing system (HotelC). These cause **Data Model Heterogeneities**.

**Schematic Heterogeneities** also exist, for example:

- Hotel location, room classification and address are declared as object types in the Object-oriented data model (HotelA), which is different from the Relational data model (HotelB) and the file process system (HotelC).
- Attributes of room status, for example, HotelA.RoomStatus, HotelB.Status, and HotelC.status are declared differently.
- Naming conflicts occur i.e. HotelA.RoomStatus.Room.Number, HotelB.STATUS.RoomNumber, HotelC.STATUS.RoomNumber.

**Semantic Heterogeneities** also exist, for example:

- Different currencies used in the price quoted of each of the hotels which are located in different countries. These cause Scaling Conflicts.
- Representation Conflicts or Domain Mismatches
    - Domain of HotelA.RoomStatus is user-defined type which is {checkin, checkout, available, reserved}.
    - Domain of HotelB.Status is CHAR which could be 'I', 'O', 'A' and 'R'.
    - Domain of HotelC.Status is CHAR which could be 'In', 'Out', 'Av' and 'Re'.

Before integration occurs, the five prerequisites of the MeDInt architecture which form the components of the Mediated MetaData (MMD) have to be maintained:

**Prerequisite 1** - New data sources have to be registered in the Data Source MetaData (DSMetaData).

```
{
AssignedName   HotelA;
DataModel      object;
Location       http://A.com/HotelDB;
SourceName     HotelA;
Objects        RoomStatus;
Description    Hotel A's database;
Constraint     Price(Currency = "USD");
}
{
AssignedName   HotelB;
DataModel      relational;
Location       http://B.com.au/HotelDB;
SourceName     HotelB;
Objects        HotelInfo, Room, Status;
Description    Hotel B's database;
Constraint     Price(Currency = "AUD");
}
{
AssignedName   HotelC;
DataModel      legacy;
Location       ;
SourceName     HotelC;
Objects        Hotel, Room, Status;
Description    Hotel C's files;
Constraint     Price(Currency = "AUD");
}
```

**Prerequisite 2** - Entity equivalences have to be indicated in the Object Mapping MetaData (OMMetaData).

```
{
GlobalObject      HotelInfo
MappedObject      SourceAssignedName     HotelA
                  SourceObject           HotelObj
MappedObject      SourceAssignedName     HotelB
                  SourceObject           HotelInfo
MappedObject      SourceAssignedName     HotelC
                  SourceObject           Hotel
}
{
GlobalObject      RoomInfo
MappedObject      SourceAssignedName     HotelA
                  SourceObject           RoomObj
MappedObject      SourceAssignedName     HotelB
                  SourceObject           Room
MappedObject      SourceAssignedName     HotelC
                  SourceObject           Room
}
{
GlobalObject      RoomStatus
MappedObject      SourceAssignedName     HotelA
                  SourceObject           RoomStatus
MappedObject      SourceAssignedName     HotelB
                  SourceObject           Status
MappedObject      SourceAssignedName     HotelC
                  SourceObject           Status
}
```

**Prerequisite 3** - Attribute equivalences have to be indicated in the Attribute Mapping MetaData (AMMetaData).

```
{
GlobalAttribute     city
MappedAttribute     SourceAssignedName      HotelA
                    SourceObject            Hotel●bj
                    SourceAttribute         Address.city
}
{
GlobalAttribute     country
MappedAttribute     SourceAssignedName      HotelA
                    SourceObject            HotelObj
                    SourceAttribute         Address.country
}
{
GlobalAttribute     class
MappedAttribute     SourceAssignedName      HotelA
                    SourceObject            RoomObj
                    SourceAttribute         Class_Type.RoomClass
}
```

**Prerequisite 4** - Data equivalences have to be defined in the Thesaurus MetaData (TSMetaData).

```
{
GlobalCategory      RoomStatus
MappedInfo          Default                 Check in
                    Aliases
                        {
                            Alias                   I
                            Alias                   In
                            Alias                   Checkin
                            Alias                   Check in
                        }
MappedInfo          Default                 Check out
                    Aliases
                        {
                            Alias                   ●
                            Alias                   Out
                            Alias                   Checkout
                            Alias                   Check out
                        }
MappedInfo          Default                 Available
                    Aliases
                        {
                            Alias                   A
                            Alias                   Av
                            Alias                   Available
                        }
MappedInfo          Default                 Reserved
                    Aliases
                        {
                            Alias               R
                            Alias               Re
                            Alias               Reserved
                        }
}
```

**Prerequisite 5** - Conversion factors of different units have to be specified in the Conversion MetaData (CVMetaData).

```
{Currency_cnv       Default                 AUD
                    {
                        CVto            USD
                        CVfactor        0.596
                        CVoperator      *
                        CVreverse       /
                    }
}
```

All the prerequisite tasks above are performed by the Registering Processor (RP). In terms of implementation, the XML documents are used to represent MMD (See Appendix I).

Assume that a user wants to enquire about the price of a standard room in hotels in 'Perth, Australia' which are available on 1st March 2003, the Mediated Query Language (MQL) is stated as follows:

```
SELECT          HotelInfo.Name, RoomInfo.Class, RoomInfo.Price (currency = 'AUD')
FROM            HotelInfo, RoomInto, RoomStatus
IN              HotelA, HotelB, HotelC
CONDITION       (HotelInfo.City = 'Perth' and
HotelInfo.Country = 'Australia' and
                RoomStatus.Status = 'Available' and
                RoomStatus.Date = '01/03/2003' and
                RoomInfo.Price < 200 (currency='AUD'))
```

Because of these data sources use different currencies, it has been stated on the query that the price shown on the output must be Australian dollars *(RoomInfo.Price (currency = 'AUD'))* which is easier for accommodation price comparison. Also, the contexts of the values stated in condition of the query can be defined clearly *(RoomInfo.Price < 200 (currency='AUD'))*.

The major task of the MeDInt Mediator after getting a query from a client is to decompose the query to subqueries and to distribute the subqueries to associated wrappers. This task is assigned to QTA. Before doing this, QTA has to fetch object schema definitions which are related to the query.

### The Process of Fetching Object Schema Definition

Following the algorithm stated in the Process *FetchDef(Đ, Ö)* (See Chapter 6), from the query, DSMetaData, and OMMetaData, QTA realises that the required object schema are as shown in Table 8.1.

TABLE 8.1 OBJECT SCHEMA DEFINITIONS REQUIRED

| HotelA | HotelB | HotelC |
|---|---|---|
| HotelObj | HotelInfo | Hotel |
| RoomObj | Room | Room |
| RoomStatus | Status | Status |

QTA send requests for the MDDLs of those objects to the STI's of associated wrappers as shown in Figure 8.1.

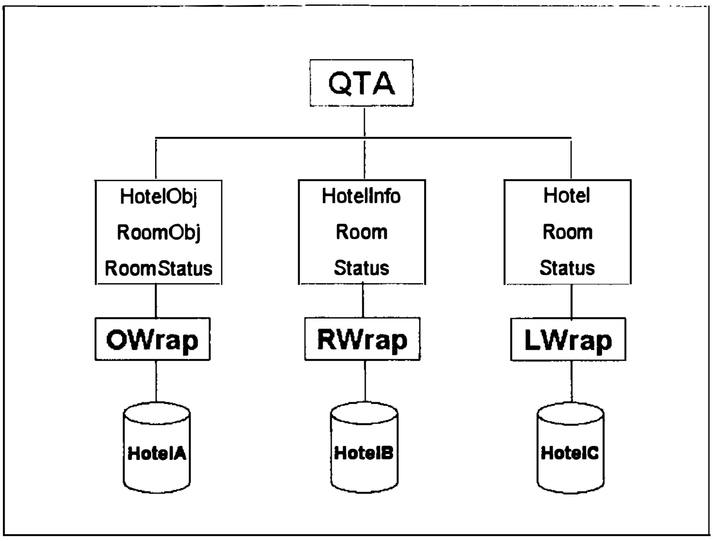


FIGURE 8.1 OBJECTS REQUESTED FROM WRAPPERS

### Schema Translation Processes

The STPs, by the *RSchmTransl(Ši,Ŏj), OSchmTransl(Ši,Ŏj)*, and *LSchmTransl(Ši,Ŏj)* processes (See Chapter 7), translate the disparate object schema definitions into MDDLs.

## From HotelA

```
HotelObj    ={
            attribute
                Name                string;
                Address             address_type;
                Phone               string;
                Fax                 string;
                Rooms               numeric;
                Description         string;
            }
RoomObj     ={
            attribute
                Hotel               HotelObj;
                Number              string;
                Location            loc_type;
                Class               class_type;
                Price               numeric;
            }
RoomStatus  ={
            attribute
                Room                room_obj;
                Date                date;
                Status              {checkin, checkout, available,
                                          reserved};
            }
```

## From HotelB

```
HotelInfo   ={
            attribute
                Name                string;
                Address             string;
                City                string;
                State               string;
                Country             string;
                Postcode            string;
                Phone               string;
                Fax                 string;
                Rooms               numeric;
                Description         string;
            key
                Name;
            }
Room        ={
            attribute
                HotelName           string;
                Number              string;
                Building            string;
                Floor               string;
                Class               string;
                NumberPersons       numeric;
                Price               numeric;
            relationship
                HotelName           HotelInfo.Name;
            key
                HotelName+Number;
            }
```

```
RoomStatus   ={
             attribute
                 HotelName             string;
                 Room                  string;
                 Date                  date;
                 Status                string;
             relationship
                 HotelName             Room.HotelName;
                 Room                  Room.Number;
             key
                 HotelName+Room+Date;
             }
```

From HotelC

```
Hotel        ={
             attribute
                 Name                  string;
                 Address               string;
                 City                  string;
                 State                 string;
                 Country               string;
                 Postcode              string;
                 Phone                 string;
                 Fax                   string;
                 Rooms                 numeric;
                 Description           string;
             }
Room         ={
             attribute
                 HotelName             string;
                 Number                string;
                 Building              string;
                 Floor                 string;
                 Class                 string;
                 NumberPersons         numeric;
                 Price                 numeric;
             relationship
                 HotelName             Hotel.Name;
             }
Status       ={
             attribute
                 HotelName             string;
                 Room                  string;
                 Date                  date;
                 Status                string;
             relationship
                 HotelName             Room.HotelName;
                 Room                  Room.Number;
             }
```

From the above MDDLs from HotelA, the *FetchDef(Đ, Ö)* process also analyses that there are further user-defined type definitions (address_type and class_type) required from data sources. Then, QTA sends another request to OWrap.

```
Address_type={
            attribute
                Number                  string;
                Street                  string;
                City                    string;
                State                   string;
                Country                 string;
                Postcode                string;
            }
Class_type  ={
            attribute
                RoomClass               string;
                NumberPersons           numeric;
            }
```

**Query Decomposing Process**

Now, QTA has enough object schema definitions for decomposing the query by the *Qtransform(Ä, Đ, Ö, Ç)* process (See Chapter 6)).

All object and attribute identifiers defined on the users' query are global identifiers which can be mapped to local identifiers with the assistance of information in OMMetaData and AMMetaData. From TSMetaData and CVMetaData, attribute values and contexts will be converted to the corresponding source values and contexts.

MQL to HotelA

```
SELECT       HotelObj.Name, RoomObj.Class_Type.RoomClass, RoomObj.Price
FROM         HotelObj, RoomObj, RoomStatus
IN           HotelA
CONDITION    (HotelObj.Address.City = 'Perth' and
             HotelObj.Address.Country = 'Australia' and
             RoomStatus.Status = 'Available' and
             RoomStatus.Date = '01/03/2003' and
             RoomObj.Price < 119.2)
```

200 (currency = 'AUD') is converted with assisting information in CVMetaData to 119.2 corresponding to the currency used in this data source.

MQL to HotelB

```
SELECT          HotelInfo.Name, Room.Class, Room.Price
FROM            HotelInfo, Room, Status
IN              HotelB
CONDITION       (HotelInfo.City = 'Perth' and
                HotelInfo.Country = 'Australia' and
                Status.Status = 'A' and
                Status.Date = '01/03/2003'  and
                Room.Price < 200)
```

MQL to HotelC

```
SELECT          Hotel.Name, Room.Class, Room.Price
FROM            Hotel, Room, Status
IN              HotelC
CONDITION       (Hotel.City = 'Perth' and
                Hotel.Country = 'Australia' and
                Status.Status = 'Av' and
                Status.Date = '01/03/2003'  and
                Room.Price < 200)
```

### Creating a Pre-defined Template Process

By *TemplCreate(Ä),* QTA also prepares a template in MDRS format

**(HotelInfo.Name, RoomInfo.Class, RoomInfo.Price (currency='AUD'))**

### Query Translation Processes

Each subquery will be sent to the QTP of its associated wrapper for query translation which is performed by the *SQLGen(χ),OQLGen(χ),* or *XQLGen(χ).*

OQL to HotelA

```
SELECT          HotelObj.Name, RoomObj.Class_Type.RoomClass, RoomObj.Price
FROM            HotelObj, RoomObj, RoomStatus
WHERE           (HotelObj.Address.City = 'Perth' and
                HotelObj.Address.Country = 'Australia' and
                RoomStatus.Status = 'Available' and
                RoomStatus.Date = '01/03/2003'  and
                RoomObj.Price < 119.2)
```

SQL to HotelB

```
SELECT      HotelInfo.Name, Room.Price
FROM        HotelInfo, Room, Status
WHERE       (HotelInfo.City = 'Perth' and
            HotelInfo.Country = 'Australia' and
            Status.Status = 'A' and
            Status.Date = '01/03/2003' and
            Room.Price < 200 and
            (HotelInfo.Name = Room.HotelName and
            Room.HotelName = Status.HotelName and
            Room.Number = Status.Room))
```

For a pair of related objects declared on a query in a relational data model, relationship statements have to be included in the condition statement.

XQuery to HotelC

```
<result>
      FOR $h IN document("http://C.com/HotelFiles/Hotel.xml")//hotel
      FOR $r IN document("http://C.com/HotelFiles/room.xml")//room[hotelname=$h/name]
      FOR $s IN document("http://C.com/HotelFiles/status.xml")//status[hotelname=$r.hotelname
                                                                        and room=$r.number]
      WHERE          ($h/city = 'Perth' and
                     $h/country = 'Australia  and
                     $s/status = 'Av' and
                     $s/date = '01/03/2003'  and
                     $r/price < 200 and )
      RETURN
             <room>
                     {$h/name}
                     {$r/price}
             </room>
</result>
```

**Data Translation Processes**

The subqueries above will be performed by the  query processing of the local database management systems. Then, the query results will be returned to wrappers. The DTPs will translate query results which are in disparate models to MDRS:

**HotelA**

```
{("Sheraton Perth Hotel", "Deluxe", 102 (currency=USD))}
```

**HotelB**

```
{("Novotel Langley Perth", "Standard", 140.00 (currency=AUD)),
("Novotel Langley Perth", "Apartment", 170.00 (currency=AUD))}
```

### HotelC

**{("City Stay Apartments", "Standard", 106.00 (currency=AUD))}**

However, the results still cannot be integrated because they are still in different contexts.

### Applying MDRS Results to the Pre-defined Template Process

The result from Hotel Chain A still needs the conflict resolving process *ApplTemp(ρ, τ, θ)* to be performed by CRA to apply the result corresponding to the predefined template. CVMetaData provides currency conversion information.

**(Hotellnfo.Name, Roomlnfo.Class, Roomlnfo.Price (currency='AUD'))**

### HotelA

{("Sheraton Perth Hotel", "Deluxe", 171.14 (currency=AUD))}

### Integrating the Mediated Data Representation Structure Process

Now all query result can be integrated by CP using the union operator.

{("Sheraton Perth Hotel", "Deluxe", 171.14 (currency=AUD)),
("Novotel Langley Perth", "Standard", 140.00 (currency=AUD)),
("Novotel Langley Perth", "Apartment", 170.00 (currency=AUD))
("City Stay Apartments", "Standard", 106.00 (currency=AUD))}

### Generating the Integrated Result Process

Finally, RA can present the integrated query result to users as shown in Table 8.2.

TABLE 8.2 INTEGRATED RESULT OF TEST PROBLEM 1

| Hotellnfo.Name | Roomlnfo.Class | Roomlnfo.Price (currency='AUD') |
|---|---|---:|
| Sheraton Perth Hotel | Deluxe | 171.14 |
| Novotel Langley Perth | Standard | 140.00 |
| Novotel Langley Perth | Apartment | 170.00 |
| City Stay Apartments | Standard | 106.00 |

From this example, the following heterogeneities (Table 8.3) have been resolved:

TABLE 8.3 HETEROGENEITIES IN THE TEST PROBLEM 1

| Heterogeneities | Conflicts | HotelA | HotelB | HotelC |
|---|---|---|---|---|
| Model | | Relational | Object | Legacy |
| Schema | Naming | RoomStatus, RoomNumber | | |
| | Structural | Address, Location, Class | | |
| Semantic | Scaling | currency='USD' | currency='AUD' | currency='AUD' |
| | Representation | RoomStatus | | |

## 8.1.2 Test Problem 2 – University Information System

This sample is a university information system which is composed of a relational system namely *UnivDB* (Figure 8.2 and 8.3)and an object-oriented system *CampusDB* (Figure 8.4).

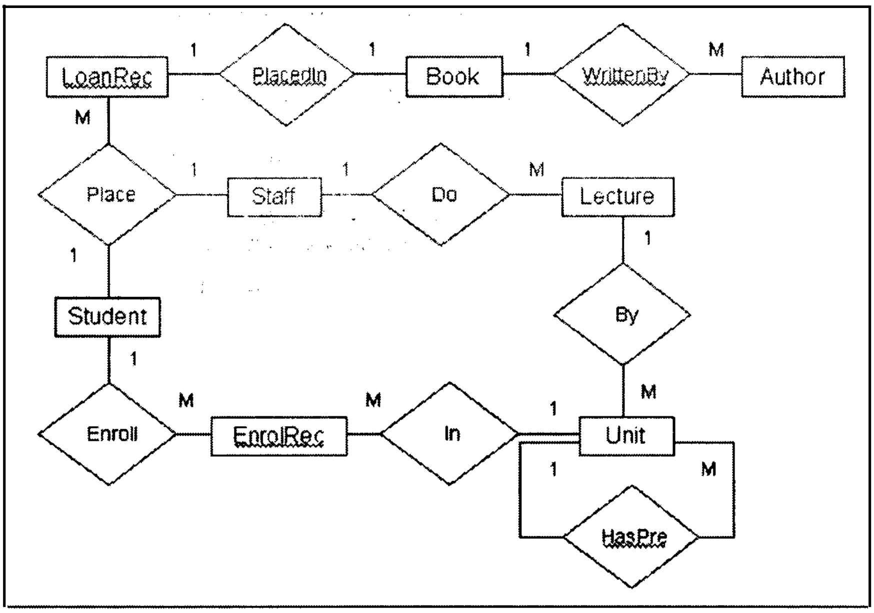


FIGURE 8.2 THE *UNIVDB* ENTITY RELATIONSHIP DIAGRAM

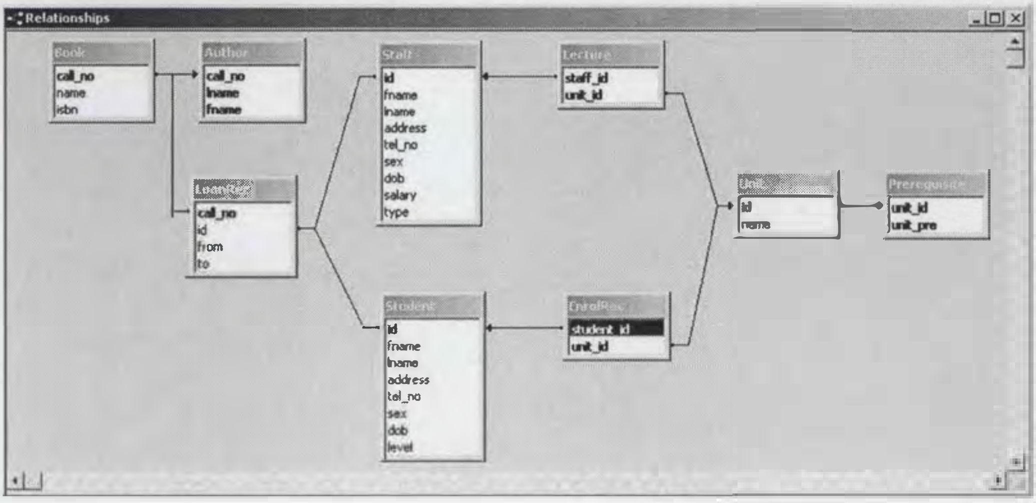


FIGURE 8.3 THE *UNIVDB*'S RELATIONSHIP

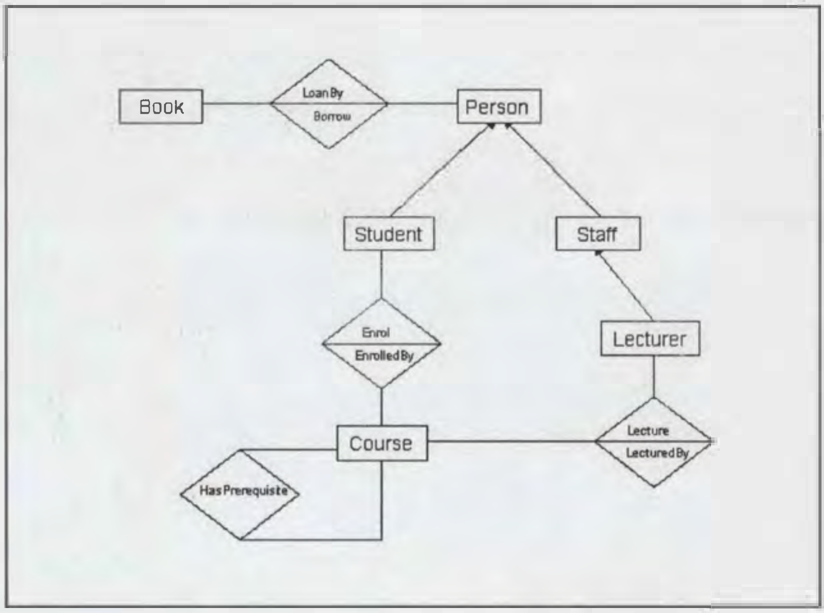


FIGURE 8.4 THE *CAMPUSDB*'S ENTITY RELATIONSHIP DIAGRAM

From this example, all three categories of heterogeneities have occurred.

Firstly, *UnivDB* is a relational data model, while *CampusDB* is an object data model (see Appendix E and F for data definitions); this causes a **Data Model Heterogeneity**.

Secondly, there is a Structural conflict in the **Schematic Heterogeneity** category which has been caused by using different structures to represent the same real-world object in both data sources. For example, in *UnivDB*, *Staff* and *Student* objects have their own attributes, relationships and key, while in *CampusDB*, *Staff* and *Student* are subtypes of *Person.* It means that *Staff* and *Student* share some equivalent characteristics. *Lecturer* is another object defined in *CampusDB* as a subtype or a specialisation of *Staff.* Furthermore, one to many and many to many relationships are normally represented differently in a relational model than from an object model which is able to distinguish between *EnrolRec, LoanRec, Prerequisite, Lecture,* and *Author* in *UnivDB*, and *Student.Enrol, Book.loanby, Course.hasprerequisite, Lecturer.Lecture,* and *Book.author* in *CampusDB.* There are also conflicts from using the structure data type *struct* in the object data model to amalgamate many attributes, for example, *name* has been defined as *struct<string fname; string lname>*. This falls into the Attribute-attribute conflicts in structural conflicts.

Finally, a number of **Semantic Heterogeneities** occur between both sources. Student level in *UnivDB* is represented by {P, U}, but in *CampusDB* it is represented by {postgrad, undergrad}; this causes a Representation conflict. Staff salary in *UnivDB* is quoted in US dollars, but in *CampusDB* is quoted in Australian dollars; this causes a Scaling conflict.

### 8.2.2.1 Query 1

The first query example is a request for the id and name of postgraduate students who enrol in 'CSP1143' from both *DS1* and *DS2*.

```
SELECT      Student.id, Student.name
FROM        Student, Unit
IN          DS1, DS2
CONDITION   Unit.id = 'CSP1143' and
            Student.level="postgrad";
```

In this example, the following heterogeneities (Table 8.4) have been resolved:

TABLE 8.4 HETEROGENEITIES IN THE QUERY 1 OF TEST PROBLEM 2

| Heterogeneities | Conflicts | UniDB | CampusDB |
|---|---|---|---|
| Model | | Relational | Object |
| Schema | Entity-entity | Unit | Course |
| | Attribute-attribute | Unit.id | Course.code |
| | Structural | Fname, lname | Name |
| | Specialisation | Student.Person | |
| Semantic | Naming | D(level)={U,P} | D(level)={postgrad, undergrad} |

All have been solved by the MeDInt Mediator and wrappers algorithms. The entire integration process is mostly the same as the previous example problem but only some details are different because of the distinction of conflict types. The details of the integration process are presented in Appendix J.

#### 8.2.2.2 Query 2

A user may want to get the id and yearly salary of staff who earns less than 50,000 AUD$ from *UnivDB(DS1)* and *CampusDB(DS2)*. This query initiates conflicts which are different from the first query.

```
Select      Staff.id, Staff.salary(currency="AUD", period="yearly")
From        Staff
In          DS1, DS2
Condition   Staff.salary(currency="AUD", period="yearly") < 50000;
```

In this query example, a Scaling conflict is added. The submitted query needs yearly salary information from *UnivDB* and *CampusDB* in Australian dollars, but in the data sources registered information in DSMetaData, the currency using in *CampusDB* is US dollars and salary is quoted on a monthly basis in *UnivDB*. Therefore, the condition in the query submitted to *CampusDB* has to be converted to US dollars and then after getting the result from *CampusDB*, again the result in US dollars has to be converted back into Australian dollars. Moreover, the query submitted to UnivDB has to be transformed into a monthly basis to compare to data in the source, and the result has to be converted back into a yearly basis by the query requested.

In this example, the following heterogeneities (Table 8.5) have been resolved:

TABLE 8.5 HETEROGENEITIES IN THE QUERY 2 OF TEST PROBLEM 2

| Heterogeneities | Conflicts | UniDB | CampusDB |
|---|---|---|---|
| Model | | Relational | Object |
| Schema | Specialisation | Staff | Staff:Person |
| Semantic | Scaling | currency='AUD' | currency='USD' |
| | Abstraction | Period='monthly' | period='yearly' |

All have been solved by the Mediator and wrappers algorithms. The details of the integration process are presented in Appendix J.

## 8.2 Summary

By applying the MeDInt architecture to a number of information systems, the correctness of the integration results are shown in the previous section. Different sets of conflicts have been resolved (Table 8.6).

TABLE 8.6 SUMMARY OF THE HETEROGENEITIES RESOLVED BY THE MEDINT ARCHITECTURE IN EACH EXAMPLE

| Heterogeneities | Conflicts | Test Problem1 | Test Problem2 | |
|---|---|---|---|---|
| | | | Query 1 | Query 2 |
| Model | | √ | √ | √ |
| Schema | Naming | √ | √ | √ |
| | Structural | √ | √ | |
| | Specialisation | | √ | √ |
| | Relationship | | √ | |
| Semantic | Naming | | √ | |
| | Scaling | √ | | √ |
| | Abstraction | | | √ |
| | Representation | √ | | |

The result from the integration process can be described in terms of conflict resolutions and functionality as follows:

### 8.2.1 Conflict Resolution In MeDInt

Conflicts between heterogeneous data sources in this study are classified into three major types which are Data Model Heterogeneity, Schematic Heterogeneity, and Semantic Heterogeneity. The previous evaluation shows that these three category conflicts can be removed successively and correctly.

**Data Model Heterogeneity**

From the example problems, component data sources of which some are relational data models, some are object-oriented data modes, and others are legacy file processing systems pose Data Model Heterogeneities. In MeDInt, the Mediated Data Model (MDM) consisting of the Mediated Data Definition Language (MDDL), the Mediated Query Language (MQL), and the Mediated Data Representation Structure (MDRS) have been employed to create a common data model to be used in communicating between the MeDInt Mediator components and wrappers. The problems of local data sources using different data definition languages can be solved by translation into MDDL by wrappers. The mediator components make uses of MDDL. Similar to the problem of different data manipulation languages, MQL is used when decomposing a user query into subqueries, before the wrappers translate these subqueries to the query language used in each data source.

**Schematic Heterogeneity**

Schema Heterogeneities in the example problems occur when the structures of same real-world objects have been defined differently in their component data sources. They are classified into Naming conflicts, Structural conflicts, Generalisation/Specialisation conflicts, and Relationship conflicts. They are solved by the assistance of mapping and constraint information defined in OMMetaData and AMMetaData.

**Semantic Heterogeneity**

Semantic Heterogeneities occur when the data in component data sources have been represented differently. These Naming conflicts and Representation conflicts are solved by TSMetaData. Scaling conflicts and Level of Abstraction conflicts are solved by the extended dimension of the Mediated Data Model in conjunction with CVMetaData. Heterogeneities resolved in the example problems are summarised in Table 8.7.

TABLE 8.7 SUMMARY OF THE HETEROGENEITIES RESOLVED BY THE COMPONENTS OF THE MEDINT ARCHITECTURE

| Resolved by | Heterogeneities | | |
|---|---|---|---|
| | Data Model | Schema | Semantic |
| MDM | √ | | √ |
| OMMetaData | | √ | |
| AMMetaData | | √ | |
| TSMetaData | | | √ |
| CVMetaData | | | √ |

## 8.2.2 The Integration Functions of the MeDInt Components

In terms of functionality, the MeDInt architecture is mainly separated into two parts which are facilitation and translation. The function of facilitation is performed by the MeDInt Mediator which has been designed especially for homogenising heterogeneities both on users' queries and on query results. Wrappers are created for the translation purpose including schema definition, query and data translation. The MeDInt component functionalities are shown in Table 8.8.

TABLE 8.8 SUMMARY OF THE FUNCTIONS OF THE MEDINT COMPONENTS

| Functions | Mediator | | | | | Wrapper | | |
|---|---|---|---|---|---|---|---|---|
| | RA | QTA | MMD | CRA | CP | STP | QTP | DTP |
| Data sources autonomy information | √ | | √ | | | | | |
| Data sources' schema definitions translation | | | | | | √ | | |
| Query decomposition and translation | | √ | | | | | √ | |
| Data Translation | | | | | | | | √ |
| Conflict Resolution | | | √ | √ | | | | |
| Data Consolidation | | | | | √ | | | |

# CHAPTER 9 – DISCUSSION AND CONCLUTION

Many organisations have put much effort to deal with information scattering from multiple data sources with the aim of providing a unique view of the information. A number of heterogeneities can arise from platform, database and data levels. At database and data levels, there are Data Model, Schematic, and Semantic Heterogeneities that need to be solved. Several integration techniques have been presented such as global schema, federated database, multidatabase approaches and so on. However, some of them are suitable for particular data models, some do not support legacy file repositories, and some generate problems in dynamic systems.

This research introduces a framework called the Mediated Data Integration (MeDInt) architecture based on the mediation approach and incorporating with wrappers and a semantic-rich data model, the Mediated Data Model (MDM), to resolve the problems of integrating heterogeneous data sources. MDM enriches the MeDInt architecture to capture different semantic contexts from data sources. No pre-integration is required before users issue their queries thus avoiding the problem of local schema evolution in dynamic systems. Furthermore, instead of schema and semantic integration, the pre-defined template in collaboration with the mediator components provides the query result consolidation without global schema integration.

This chapter presents the discussion of the MeDInt architecture, thesis contribution, limitations and future research directions.

## 9.1 Discussion

From the review and extensive investigation, it has been found that heterogeneities, which are the major problem of heterogeneous data integration, can be classified into three categories: Data Model Heterogeneities, Schematic Heterogeneities, and Semantic Heterogeneities.

Data Model Heterogeneities exist when different data models are used to describe component data sources. This includes the use of different data definition languages to describe component schemas and the use of different data manipulation languages to describe user queries. Schematic Heterogeneities can be found at the schema level of component data sources when different structures are used to represent the same concept. In addition, they can result from different data model characteristics and/or design autonomy. Semantic Heterogeneities are found at the data level when the same set of data is represented in different terminologies or different contexts. A number of efforts have been introduced to resolve heterogeneities, for example, mapping techniques, schema translation, meta-data repositories, join methods, homogenising, the Object Exchange Model (OEM), semantic specification, superclasses, and so on.

Several integration approaches have been introduced to interoperate heterogeneous data sources and to resolve the heterogeneities. The global schema approach is a fully-integrated approach or tightly-coupled approach. The component schemas are integrated by a single view. The federated database approach can be tightly- or loosely- coupled. More than one federated schema is created by users or administrators. The multidatabase approach is more loosely-coupled by providing a multi-database manipulation language as a query tool to communicate with component databases. However, each approach has some limitations, for example, the global schema and multidatabase approaches cannot be served by legacy file processing systems, the global schema and federated schemas have to be recreated in dynamic systems when component schemas changed, and so on.

This research investigates the design of an approach to logically integrate database and legacy file processing systems and to resolve the three previously classified heterogeneities. The integration and conflict resolution processes should be transparent to users when they issue the queries. One of the major concerns is the component schema evolution should not affect the integration or lead to a large number of consequent modifications. The research finally introduces the MeDInt architecture based on the mediation approach as a solution to logically integrating heterogeneous data sources. It is the middle layer between clients and multiple data sources. It encompasses three major components: the MeDInt Mediator, MDM, and wrappers. The MeDInt architecture can be explained based on the conceptual level of the ANSI/SPARC architecture.

The MeDInt Mediator is in-between the clients and the wrappers. It has been designed to overcome Schema and Semantic Heterogeneity issues. It functions as an agent homogenising conflicts in both directions. In the client-to-source direction, it decomposes user queries according to the schemas and semantic contexts of component data sources. In the source-to-client direction, it homogenises results which are schematic and semantic differences to have the same structure and context as the pre-defined template. The MeDInt has six components. The Registering Processor (RP) captures component data source, object, attribute and constraint information to the MeDInt MetaData (MMD). MMD consists of the Object Mapping MetaData (OMMetaData), the Attribute Mapping MetaData (AMMetaData), the Thesaurus MetaData (TSMetaData), and the Conversion MetaData (CVMetaData). The Query Transformation Agent (QTA) decomposes and transforms the query to subqueries in the same context as the target data sources. The Conflict Resolution Agent (CRA) resolves the conflicts by homogenising query results corresponding to the pre-defined template. The Consolidation Processor (CP) merges conflict-resolved results from multiple data sources. The Rendering Agent (RA) finally generates the integrated results to display to users.

MDM is developed to be a common data model used in the MeDInt Mediator for solving Data Model Heterogeneities. MDM characteristics are derived from the object data model. However, it adds the third dimension to the two dimensions of the relation data model to represent semantic contexts. Therefore, it is not only a general data model which just describes the structure of data sources, but it is also capable of depicting and representing heterogeneous data models schematically and semantically. MDM consists of the Mediated Data Definition Language (MDDL), the Mediated Query Language (MQL), and the Mediated Data Representation Structure (MDRS). The Mediated Data Definition Language (MDDL) is able to express schemas of different data models semantically. The Mediated Query language (MQL) is a semantic query language by which users can specify the query with the context if the data in component sources are represented in different contexts. The Mediated Data Representation Structure (MDRS) presents data with its contexts in order to be consolidated correctly.

Wrappers overcome Data Model Heterogeneities including different data definition language and data manipulation language issues. They function as translators interpreting different schemas, queries, and data from/to MDM. Component schemas are translated by Schema Translation Processors (STPs). User queries are translated by Query Translation Processors (QTPs). Results are translated by Data Translation Processors (DTPs). In this research, wrappers are provided for relational data models, object data models, and legacy file systems. Each of them includes an STP, a QTP, and a DTP.

In summary, Data Model Heterogeneities covering different data definition languages and data manipulation languages can be overcome by the Mediated Data Model (MDM) incorporating wrappers. Schema Heterogeneities can be resolved with the assistance of mapping information and constraint information defined in OMMetaData and AMMetaData. Semantic Heterogeneities are resolved by TSMetaData, CVMetaData, and the extended dimension of MDM.

On resolving the Schematic heterogeneities, one of the strengths of the MeDInt architecture is that on the integration process, it neither tries to force component schemas to create a global schema, nor integrates them directly, but only query results are consolidated. This does not violate original schemas. Furthermore, this avoids pre- and full-integration and therefore can solve the problem of schema changing in dynamic systems. In addition to the semantic conflict resolution process, the Semantic Heterogeneities are not solved directly, but each result from the component data sources will be transformed to have the same format as the pre-defined template.

The MeDInt architecture can be described as partial automation. Conflict resolution processes are transparent to users. Only the pre-registered process needs to be done at the beginning or when a new data source is added to the integration system. This task is done by RP in cooperation with MMD. These help users in minimising the complexity of the query processes by which the users do not have to find out where data sources are, what conflicts exist, and how to resolve them.

Compared to other dynamic integration systems, in terms of minimisation, this method is applied to get only query-associated object schema definitions in order to decompose and transform a query. This shows efficient performance especially in medium- or large-sized organisations which involve a number of data sources and/or a large number of entities, because most of the queries just require information from a small portion of the entire information of an organisation. However, for small-sized organisations, the method can be changed to get all object schemas once which is less complicated and is a subset of this architecture.

In relation to usability, MQL, an extension of SQL which is familiar to users, allows users to specify their own queries. The semantic contexts can be specified on the projection and restriction parts of MQL. In terms of scalability and flexibility, when a new data source is added to the integration system and uses the same data model as the pre-registered data sources, only the registering process is required. However, if a new data source with data model heterogeneities is added, a new wrapper is also required. The integration system therefore requires only minimised modifications with the addition or removal of data sources.

MMD is implemented using the eXtensible Markup Language (XML) which is a W3C's standard of representing and exchanging structured data. Two examples of integration systems in Chapter 8 and Appendix J were tested and evaluated. They show and prove the validity and effectiveness of the MeDInt architecture.

From the specified research goals which focus on investigating an effective approach to integrating heterogeneous systems, each goal has been achieved:

- Addressing conflicts among heterogeneous database systems;
- Providing conflict resolution;
- Providing the appropriate architecture for achieving the interoperability or logically integrating of multiple data sources by which schema evolution will not affect the integration;
- This research covers legacy file processing systems, the relational data model and the object-oriented data model.

TABLE 9.1 COMPARISON OF MEDINT WITH OTHER INTEGRATION APPROACHES

| | Global Schema Approach | Federated Database Approach | Multi-database Language Approach | MeDInt |
|---|---|---|---|---|
| Serving schema Evolution | No.<br>Pre-created global schema requires to be recreated | No.<br>Pre-created federated schemas require to be recreated | Yes | Yes |
| Integration responsibility | DBA | DBA or users. Depend on tightly or loosely approach | Users | Automation |
| Conflict resolution responsibility | DBA | DBA or users. Depend on tightly or loosely approach | Users | Automation |
| Schema integration process | Complicate.<br>Especially when many data sources are involved. | Complicate. | No. | Automation |
| Semantic integration process | Complicate.<br>Have to be done together with schema integration process | Complicate. | Complicate.<br>Users need to understand all component data sources thoroughly. | Automation |
| Structural Integration | Yes.<br>A global schema is created. | Yes.<br>Federated schemas are created. | No | No.<br>Only results are consolidated. |
| Transparent to users | Yes | Yes/No.<br>Depend on tightly or loosely approach | No | Yes |
| Scalability | No | | | Yes |
| Support legacy file systems | No | No | No | Yes |

Finally, Table 9.1 shows the comparison of MeDInt with other integration approaches. The MeDInt is unique in serving dynamic systems whose component schemas could be changed dramatically. It is a partial automated integration by which only pre-registration information is required. Neither database administrators or users are responsible for the integration process and the conflict resolution process. Such complex processes are transparent to users. In terms of scalability, only a wrapper is required to be developed when a new data sources from a different data model is added to the integration system. Furthermore, legacy file processing systems can be interoperated in the MeDInt architecture.

TABLE 9.2 COMPARISON OF MEDINT WITH OTHER MODELS OF THE MEDIATION APPROACH

| | TSIMMIS | Context Mediator | AURORA | MeDInt |
|---|---|---|---|---|
| Techniques employ | Mediator, Global Schema, Object Exchange | Mediation, Conversion, Shared Ontologies | Mediation, Homogenisation, Integrated view, Wrapper | Mediator, Wrapper, Semantically-rich data model, MetaData |
| Mediation technique | Mediate the differences between the integrated view and the underlying views | Mediate the differences using conversion | Mediate the relation using transformation technique | Mediate the differences using the translation and semantic representation techniques. |
| Integration technique | Generate the routines for combining information by reformulating queries | Terminology Mapping | Creating an integrated view | Translating the queries into the underlying context |
| Data Modelling | Object Exchange Model – Hierarchy representation | Information and its data environment | N/A | MDRS as a data model to represent data and its contexts |
| Query Issuing | Users issue query based on the context of the mediated global schema | Users can define query by their own context | User issue the query based on the integrated view | Users can define query by their own context |
| Data Model Heterogeneity | Yes | N/A | N/A | Yes |
| Schematic Heterogeneity | Yes | N/A | Yes | Yes |
| Semantic Heterogeneity | N/A | Yes | N/A | Yes |
| Support Static/Dynamic Integration Environment | Static | Dynamic | Static | Dynamic |

The comparison of the MeDInt architecture to other mediation architecture is shown in Table 9.2

## 9.2 Thesis Contribution

The contributions of the work presented from this research are:

- presenting a transparent data integration framework based on the mediation and wrapper approach to homogenise the heterogeneities and to interoperate database and legacy systems;
- introducing a semantically-rich data model, MDM, which is capable of describing the Schematic and Semantic Heterogeneities of multiple data models;
- finding the shared characteristics of disparate data sources and giving these integration tasks to the MeDInt Mediator, while the unshared characteristics of data sources are pushed to wrappers for efficiency;
- initiating the idea of design a database management system for which database administrators can determine the data semantic context freely. This performs well especially in medium- or large-sized organisations in both keeping tracks of the large amount of information to be meaningful and interoperating with other data sources when needed.

## 9.3 Limitations

1. This architecture focuses on read-only access to the integration.

2. Only SQL and OQL were considered in query translation as representative of relational and object-oriented query languages respectively. However, for other query languages, the appropriate algorithm can be developed using the same concept.

## 9.4 Future directions

This section provides some recommendations for future research.

1. One of the weaknesses of no pre-integrated schema is that it requires fetching the component schemas during the query decomposition process. Therefore,

the research can be extended to cover the query performance with the aim of enhancing the performance of the entire system.

2. The validity of the conflict resolution process still depends on human to define the correspondences. To enhance the mapping automation, a rule-based system can be applied to schematic and semantic mappings to reduce the manpower required and human errors in the manual mapping process. This has the benefit of only re-defining some incorrect cases.

3. Dynamic conversions can be extended, for example the currency conversion factor can use current information from the Internet to reduce the time spent on maintenance.

4. The interception of different legacy systems could be investigated to create a template from their common points for generating wrappers in order to avoid creating everything from scratch.

5. Because this research focuses on read-only access to the integration, the architecture can be extended to read-write access with careful consideration of the consistency aspect by updating to master data sources and appropriately propagating to replicating data sources.

6. In terms of increasing user friendliness, a graphical user interface could be developed to draw component schemas and contexts to simplify the query specification.

## *9.5 Conclusion*

Generally, multiple and heterogeneous data sources are used to serve an organisation for different operational purposes. Depending on the management perspectives, related information should be interoperated to provide the unique concept to enhance decision making. To do this, some critical problems occur, for example, how to integrate data sources which have different data models, how to solve the problem when the structure of data sources is designed differently, how to solve the problem of different terminologies or different contexts. From the existing integration approaches, a number of obstacles have been found, for example, the effect from component schema evolution. The research questions have been directed towards meeting the integration system requirements.

MeDInt is a mediation-wrapper approach presented as a framework to interoperate heterogeneous data sources. It has been designed based on the functional, divide and conquer, top-down approach. The MeDInt Mediator incorporates wrappers and MDM, a semantic data model, to accomplish the integration requirements and resolve the heterogeneities which are categorised in this research into Data Model Heterogeneities, Schematic Heterogeneities, and Semantic Heterogeneities. By the design, the shared-characteristics of the integration processes are assigned to the MeDInt Mediator while the unshared-characteristics which are the differences in data models are assigned to wrappers. The MeDInt Mediator deals with homogenising tasks including getting component schema definitions into MDDL, decomposing and transforming user queries into MQL subqueries with contexts corresponding to the data source contexts, applying MDRS results to the pre-defined template to resolve Schematic and Semantic Heterogeneities, and finally consolidating conflict-resolved results. These show that the MeDInt Mediator principally functions as a conflict-broker of all data sources resolving the three previously-classified heterogeneities with the assistance of wrappers which are designed to be translators. Each of them translates schema definitions, query languages, and results between the data model of a data source and MDM. This reduces the complexity of dealing with several data models at the same time during the integration process by taking the advantage of using a unique data model, MDM. It is capable of describing the component data models schematically and semantically through the extended third-dimension which is responsible for capturing semantic contexts.

The information systems including object-oriented data sources, relational data sources, and legacy file processing data sources have been tested and evaluated. The results show the validity and effectiveness of the approach. The complex processes of query decomposition, query transformation, query translation, data translation, conflict resolution, and data consolidation have been made transparent to users. The component schema definitions are gathered after users issue the queries, thus the problem of schema evolution can be solved. The query therefore gets the latest component schema update. In addition, no schema is integrated, but only results are consolidated. Human interaction is required only in the data source registration phase when a new data source of a different data model is added to the integration system.

# REFERENCES

Abdalla, K. (1998). *A new approach to the integration of heterogeneous databases and information systems.* Unpublished doctoral dissertation, University of Miami, Florida.

Aronica, R. C., & Rimel, D. E. (1996). Wrapper your legacy systems. *Datamation, 42*(12), 83-88.

Batini, C., Lenzerini, M., & Navathe, S. B. (1986). A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys, 18*(4), 323-364.

Ben-Natan, R. (1995). *CORBA: a guide to common object request broker architecture.* New York, USA: McGraw-Hill.

Benyon, D. (1997). *Information and data modelling* (2nd ed.). England: McGraw-Hill.

Bertino, E., Catania, B., & Zarri, G. P. (2001). *Intelligent database systems.* Britain: ACM Press.

Blaha, M., & Premerlani, W. (1998). *Object-oriented modeling and design for database applications.* New Jersey, USA: Prentice-Hall.

Bolloju, N. (1996). Semantic query transformation: an approach to achieve semantic interoperability in homogeneous application domains. In T.-Y. Cheung, J. Fong & B. Siu (Eds.), *Database Reengineering and Interoperability.* New York, USA: Plenum Press.

Bouguettaya, A., Benatallah, B., & Elmagarmid, A. (1999). An overview of multidatabase systems: past and present. In A. Elmagarmid, M. Rusinkiewicz & A. Sheth (Eds.), *Management of heterogeneous and autonomous database systems* (pp. 1-32). San Francisco, California, USA: Morgan Kaufmann Publishers.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., & Maler, E. (2000, August 14). *Extensible Markup Language (XML) 1.0.* Retrieved August 15, 2000, from http://www.w3.org/TR/2000/WD-xml-2e-20000814

Bright, M. W., Hurson, A. R., & Pakzad, S. (1994). Automated resolution of semantic heterogeneity in multidatabases. *ACM Transactions on Database Systems, 19*(2), 212-253.

Bright, M. W., Hurson, A. R., & Pakzad, S. H. (1992). A taxonomy and current issues in multidatabase systems. *Computer, 25*(3), 50-59.

Cattel, R. G. G., & Barry, D. K. (Eds.). (2000). *The object data standard: ODMG 3.0.* California, USA: Academic Press.

Chang, Y.-H., & Raschid, L. (1996). Using parameterized canonical representations to resolve conflicts and achieve interoperability between relational and object databases. In T.-Y. Cheung, J. Fong & B. Siu (Eds.), *Database Reengineering and Interoperability*. New York, USA: Plenum Press.

Chirathamjaree, C., & Mukviboonchai, S. (2002a, 8-9 August). *A Mediated Data Model for Heterogeneous Data Integration.* Paper presented at the 2nd Annual International Conference on Computer and Information Science (ICIS '02), Seoul, Korea.

Chirathamjaree, C., & Mukviboonchai, S. (2002b, 28-31 October). *The Mediated Integration Architecture for Heterogeneous Data Integration.* Paper presented at the 17th IEEE Region 10 International Conference on Computers, Communications, Control and Power Engineering (IEEE TENCON'02), Beijing, CHINA.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM, 13*(6), 377-387.

Codd, E. F. (1979). Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems, 4*(4), 397-434.

ComputerUser.com Inc. (2000). *High-tech dictionary.* Retrieved May 11, 2000, from http://www.currents.net/resources/dictionary/noframes/index.html

Conrad, S., Hasselbring, W., Hohenstien, U., Kutsche, R., Roantree, M., Saake, G., et al. (1999). Engineering Federated Information Systems (EFIS '99 Workshop Report). *SIGMOD Record, 28*(3).

Critchlow, T. (1997). *Schema coercion: using database meta-information to facilitate data transfer.* Unpublished doctoral dissertation, University of Utah.

Critchlow, T., Ganesh, M., & Musick, R. (1998, August). *Meta-data based mediator generation.* Paper presented at the Proceedings of the Third IFCIS Conference on Cooperative Information Systems (CoopIS'98), New York, USA.

CrossAccess Corporation. (2001). *Completing the enterprise integration strategy: data integration for e-business in search of speed, scale and scope* (white paper). Santa, Clara, California, USA.

Date, C. J. (1990). *An introduction to database systems* (5th ed. Vol. 1). USA: Addison-Wesley Publishing.

Distributed Management Group. (n.d.). *Distributed object computing with CORBA*. Retrieved March 22, 2000, from http://www.infosys.tuwien.ac.at/Research/Corba/

Domenig, R., & Dittrich, K. R. (2000, November). *A query based approach for integrating heterogeneous data sources*. Paper presented at the Proceedings of the Ninth International Conference on Information and Knowledge Management (CIKM'2000), Washington DC, USA.

Goh, C. H., Bressan, S., Madnick, S., & Siegel, M. (1999). Context interchange: new features and formalisms for the intelligent integration. *ACM Transactions on Information Systems, 17*(3), 270-293.

Goh, C. H., Madnick, S. E., & Siegal, M. D. (1994, 29 November - 2 December). *Context interchange: overcoming the challenges of large-scale interoperable database systems in a dynamic environment*. Paper presented at the Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), Gaithersburg, Maryland, USA.

Goldfarb, C. F., & Prescod, P. (2000). *The XML handbook* (2nd ed.). New Jersey: Prentice-Hall.

Gray, P. M. D., Kulkarni, K. G., & Paton, N. W. (1992). *Object-oriented databases: a semantic data model approach*. UK: Prentice-Hall International (UK).

Hammer, J. (1999). *The information integration wizard (IWiz) project* (report on work in progress No. TR99-019): Department of Computer and Information Science & Engineering, University of Florida.

Hammer, M., & McLeod, D. (1981). Database description with SDM: a semantic database model. *ACM Transactions on Database Systems, 6*(3), 351-386.

Heiler, S., Miller, R. J., & Ventrone, V. (1996, April). *Using metadata to address problems of semantic interoperability in large object systems*. Paper presented at the Proceedings of the First IEEE Metadata Conference, Silver Spring, Maryland, USA.

Heimbigner, D., & Mcleod, D. (1989). A federated architecture for information management. In A. Gupta (Ed.), *Integration of information systems: bridging heterogeneous databases* (pp. 46-71). New York, USA: IEEE Press.

Hirschheim, R., Klein, H. K., & Lyytinen, D. (1995). *Information systems development and data modelling: conceptual and philosophical foundations*. Britain: Cambridge University Press.

Holowczak, R. D., & Li, W. S. (1996, April). *A survey on attribute correspondence and heterogeneity metadata representation*. Paper presented at the Proceedings of the First IEEE Metadata Conference, Silver Spring, Maryland, USA.

Howe, D. (1999). *The free on-line dictionary of computing*. Retrieved May 10, 2000, from http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?IMS

Hughes, J. G. (1991). *Object-oriented databases*. New York, USA: Prentice-Hall.

Hurson, A. R., & Bright, M. W. (1996). Object-oriented multidatabase systems. In O. A. Bukhres & A. K. Elmagrarmid (Eds.), *Object-oriented multidatabase systems: a solution for advanced application*. New Jersey, USA: Prentice-Hall.

Internet.com Corp. (2000, Aug. 11, 1997). *Webopedia: the number one encyclopedia dedicated to computer technology*. Retrieved 10 May, 2000, from http://webopedia.internet.com/TERM/e/enterprise.html

ISO/IEC. (1996). *Information technology - Syntactic Metalanguage - Extended BNF, ISO/IEC 14977*, from http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

ISO/IEC/TC JTC 1. (2002, 6 September). *Information technology - Metadata registries - part 1: framework*. Retrieved October 8, 2002, from http://www.sdct.itl.nist.gov/~ftp/18/11179/11179-1.htm

Jakobovits, R. (1997). *Integrating autonomous heterogeneous data sources* (report No. TR-97-12-05): Department of Computer Science Engineering, University of Washington.

Kapitskaia, O., Tomasic, A., & Valduriez, P. (1997). *Dealing with discrepancies in wrapper functionality* (Technical Report No. RR-3138): INRIA.

Kim, W. (1995). *Modern database systems: the object model, interoperability, and beyond*. New York, USA: ACM Press.

Kim, W., Choi, I., Gala, I., & Scheevel, M. (1993). On resolving schematic heterogeneity in multidatabase systems. *Journal of Distributed and Parallel Database, 1*(3), 251-279.

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

Kroenke, D. M. (2002). *Database processing: fundamentals, design & implementation* (8th ed.). New Jersey, USA: Prentice Hall.

Li, C., Yerneni, R., Vassalos, V., Garcia-Molina, H., Papakonstantinou, Y., Ullman, J., et al. (1998, June). *Capability based mediation in TSIMMIS*. Paper presented at the Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'98), Seattle, Washington, USA.

Lu, H. (1998). *A data mining approach for resolving conflicts during data integration*. Retrieved October 18, 1999, from http://www.comp.polyu.edu.hk/News/Seminars/sem980917.html

Lu, H., Fan, W., Goh, C. H., Madnick, S. E., & Cheung, D. W. (1997, October). *Discovering and reconciling semantic conflicts: a data mining perspective.* Paper presented at the IFIP Working Conference on Data Semantics (DS-7), Switzerland.

Matena, V., & Hapner, M. (1999, December 17). *Enterprise JavaBeans$^{TM}$ specification, v1.1.* Retrieved March 28, 2000, from http://www.javasoft.com/products/ejb/docs.html

McBrien, P., & Poulovassilis, A. (2001, June). *A semantic approach to integrating XML and structured data sources.* Paper presented at the Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE 2001), Interlaken, Switzerland.

Miller, R. J. (1998, June). *Using schematically heterogeneous structures.* Paper presented at the Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'98), Seattle, Washington, USA.

Morgenstern, M. (1997, September). *Metadata for heterogeneous databases.* Paper presented at the Second IEEE Metadata Conference, Silver Springs, Maryland, USA.

Morrison, M., Boumphrey, F., & Brownell, D. (2000). *XML unleashed.* Indiana, USA: Sams Publishing.

Mowbray, T. J., & Zahavi, R. (1995). *The essential CORBA.* Canada: John Wiley & Sons.

Mukviboonchai, S., & Chirathamjaree, C. (2001a, 1-5 July). *XMInt: an mediated integration model.* Paper presented at the Eleventh Annual International Symposium of the International Council On Systems Engineering (INCOSE 2001), Melbourne, Australia.

Mukviboonchai, S., & Chirathamjaree, C. (2001b, 22-25 July). *An XML based approach for the integration of database and legacy systems.* Paper presented at the 5th World Multi-Conference on Systemics, Cybernetics, and Informatics (SCI2001), Orlando, FL.

NCITS. (1999). *American national standard dictionary for information technology (ANSDIT).* Retrieved March 22, 2000, from http://www.x3.org/tc home/k5htm/WD.htm

Neild, T. H. (1999). *The virtual data integrator: an object-oriented mediator for heterogeneous database integration.* Unpublished doctoral dissertation, Northwestern University.

Newton, J. (1996, April). *Application of metadata standards.* Paper presented at the Proceedings of the First IEEE Metadata Conference, Silver Spring, Maryland, USA.

OMG. (2001, February). *The common object request broker: architecture and specification.* Retrieved June 5, 2001, from ftp://ftp.omg.org/pub/docs/formal/01-02-01.pdf

Papakonstantinou, Y., Garcia-Molina, H., & Widom, J. (1995, March). *Object exchange across heterogeneous information sources.* Paper presented at the Proceedings of the Eleventh International Conference on Data Engineering (ICDE'95), Taipei, Taiwan.

Phijaisanit, W. (1997). *Dynamic meta-data support for information integration and sharing across heterogeneous databases (federated database).* Unpublished doctoral dissertation, George Mason University.

Potter, W. D., Trueblood, R. P., & Eastman, C. M. (1989). Hyper-semantic data modeling. *Data & Knowledge Engineering, 4*, 69-90.

Rao, B. R. (1994). *Object-oriented databases: technology, applications, and products.* Singapore: McGraw-Hill.

Reddy, M. P., Prasad, B. E., & Reddy, P. G. (1989). Query processing in heterogeneous distribute database management systems. In A. Gupta (Ed.), *Integration of information systems: bridging heterogeneous databases.* New York, USA: IEEE Press

Roth, M. T., Arya, M., Haas, L., Carey, M., Cody, W., Fagin, R., et al. (1996, June). *The Garlic project.* Paper presented at the Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD'96), Montreal, Quebec, Canada.

Roth, M. T., & Schwarz, P. (1997, August). *A wrapper architecture for legacy data sources.* Paper presented at the Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97), Athens, Greece.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). *Object-oriented modeling and design.* New Jersey, USA: Prentice Hall.

Scallan, T. (1999). *Assuring reliability of enterprise JavaBean applications.* Retrieved March 15, 2000, from www.segue.com

Schönhoff, M., Strässler, M., & Dittrich, K. R. (1997, June). *Data integration in engineering environments.* Paper presented at the Proceedings of the International Conference on Advanced information Systems Engineering (CAiSE'97) Workshop, Barcelona, Spain.

Sciore, E., Siegal, M., & Rosenthal, A. (1994). Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Transactions on Database Systems, 19*(2), 254-290.

Scowen, R. S. (1998, 17 September). *Extended BNF - a generic base standard*, from http://www.cl.cam.ac.uk/~mgk25/iso-14977-paper.pdf

Segue Software. (n.d.). *A CORBA primer*. Retrieved March 14, 2000, from www.omg.org/library/whitepapers.html

Seligman, L., & Rosenthal, A. (1996, April). *A metadata resource to promote data integration*. Paper presented at the Proceedings of the First IEEE Metadata Conference, Silver Spring, Maryland, USA.

Sheth, A. P., & Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys, 22*(3).

Shipman, D. W. (1981). The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems, 6*(1), 140-173.

Srinivasan, U. (1997). *A framework for conceptual integration of heterogeneous databases.* Unpublished doctoral dissertation, University of New South Wales, Sydney, Australia.

Stallings, W. (2001). *Operating systems: internals and design principles* (4th ed.). New Jersey, USA: Prentice-Hall.

Strässler, M., & Schönhoff, M. (1998, December). *Integrating engineering databases: how does the application domain influence the FDBMS architecture?* Paper presented at the Föderierte Datenbanken Workshop, Magdeburg, Denmark.

Thomas, A. (1998, December). *Enterprise JavaBeans$^{TM}$ technology: server component model for the Java$^{TM}$ platform.* Retrieved February 25, 2000, from http://www.javasoft.com/products/ejb/white paper.html

Tomasic, A., Raschid, L., & Valduriez, P. (1995, May). *Scaling heterogeneous databases and the design of Disco.* Paper presented at the Proceedings of the 16th International Conference on Distributed Computing Systems, Hong Kong.

Tun, Z. Z., Goodchild, A., Bird, L., & Sue, H. (1999). *Introduction to XML Schema.* Retrieved October 30, 2000, from http://www.dstc.edu.au/Research/Projects/Titanium/papers/XMLSchema/Intro-to-XMLSchema.html

Vogel, A., & Rangarao, M. (1999). *Programming with enterprise JavaBeans, JTS and OTS: building distributed transactions with Java and C++*. Canada: John Wileys & Sons, Inc.

Weiderhold, G. (1995). *Mediation and software maintenance.* Paper presented at the OO-ER Conference.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer Magazine, 25*(3).

Wiederhold, G., & Genesereth, M. (1997). The conceptual basis for mediation services. *IEEE Expert, 12*(5), 38-47.

Woelk, D., Bohrer, B., Brice, R., Huhns, M., Jacobs, N., Ksieyzk, T., et al. (n.d.). *Carnot.* Retrieved April 10, 2000, from http://www.mcc.com/projects/infosleuth/archives/carnot/

*XML query uses cases.* (2002, 16 August). Retrieved 23 September, 2002, from http://www.w3.org/TR/xmlquery-use-cases

*XQuery 1.0: an XML query language.* (2002, 16 August). Retrieved 23 September, 2002, from http://www.w3c.org/TR/xquery

Yan, L. L., Ozsu, M. T., & Liu, L. (1997, June). *Accessing heterogeneous data through homogenization and integration mediators.* Paper presented at the Second IFCIS Conference on Cooperative Information Systems (CoopIS-97), Charleston, South Carolina, USA.

Yu, T. F. (1997). *Information modeling and mediation languages and techniques for information sharing among heterogeneous information systems.* Unpublished doctoral dissertation, University of Florida.

Zhou, G., Hull, R., & King, R. (1996). Generating data integration mediators that use materialization. *Journal of Intelligent Information Systems, 6*(2/3), 199-221.

Zhou, G., Hull, R., King, R., & Franchitte, J. C. (1995). Supporting data integration and warehousing using H2O. *Data Engineering Bulletin, 18*(2).

# APPENDICES

# Appendix A – Glossary

**Agent** is a self-contained program capable of controlling its own decision making and acting, based on its perception of its environment, in pursuit of one or more objectives (Bertino et al., 2001).

**Aggregation** is the process of collecting together a number of characteristics of something and treating it as a single thing (Benyon, 1997).

**Classification** is the process of recognising that various objects share certain characteristics and can be treated as a single thing (Benyon, 1997).

**Data integration** is the method of accessing multiple data sources and receiving only one unified result to solve the problem of island of information.

**Date Model Heterogeneity** occur when data in component data sources to be interoperated are in different data models.

**Design autonomy** refers to data sources are designed without awareness of the existing related data sources. This leads to heterogeneity problem when data integration is required. .

**Directly-associated objects** are objects that QTA can determine instantly from information from the user-requested query. The schemas of these objects are required to decomposing and transforming the query.

**Extended Backus-Naur Form (EBNF)** is a syntactic metalanguage which presents by a notation for defining the linear sequence syntax of a language by use of a number of rules (1996; Scowen, 1998).

**Generalisation** is a relationship that an object class is defined as a superset of other objects.

**Heterogeneity** is the problem when integrating heterogeneous data sources. It has been defined in this study into three categories: Data Model Heterogeneity, Schematic Heterogeneity, and Semantic Heterogeneity.

**Interoperability** is the capability that databases, software and hardware can communicate, execute programs, exchange services, or transfer data among various systems (NCITS, 1999).

**Legacy system** is a critical application system, which has served an organisation for several years. Although the system is not compatible and hard to modify, it is still used because an organisation has invested considerably time and money and cost of replacing is (ComputerUser.com Inc., 2000; Howe, 1999; Internet.com Corp, 2000).

**Mediator** is a dynamic interface between clients and databases. It provides communication needed to transform data to information (Wiederhold, 1992).

**Metadata** is the description of the structure of data (Kroenke, 2002).

**Middleware** is a set of drivers, APIs, or other software that improves connectivity between a client application and a server (Stallings, 2001).

**Schema** is a description of the structure of a database. Such description, generally stored in a data dictionary, is relevant to the level of (Internet.com Corp, 2000; NCITS, 1999).

**Schematic Heterogeneities** are conflicts which results from the use of different schemas or structures in heterogeneous database systems.

**Schema evolution** is the process of changing the structure or the behaviour of persistent classes including creating, dropping, renaming, changing attributes and methods in the classes (Rao, 1994).

**Semantics** are the relationships of characters or groups of characters using as symbols to their meanings (NCITS, 1999).

**Semantic Heterogeneities** are conflicts which occur when data which have the same meaning are represented differently by different database systems.

**Specialisation** is a relationship that an object is defined as a subset of a general object class.

**Structural view** focuses on the main objects which are in the system and how those objects are related (Benyon, 1997).

**Transitively-associated objects** are objects relating to the query that QTA determines further from directly-associated object schema definitions that their schemas are required to decomposing and transforming the query.

**Wrapper** is an interface between the MeDInt mediator and data sources translating schema definitions, query languages, and data.

## Appendix B – List of Acronyms

| | |
|---|---|
| **AMMetaData** | The Attribute Mapping MetaData |
| **CORBA** | Common Object Request Broker Architecture |
| **CP** | The Consolidation Processor |
| **CRA** | The Conflict Resolution Agent |
| **CVMetaData** | The Conversion MetaData |
| **DBMS** | Database Management System |
| **DDL** | Data Definition Language |
| **DML** | Data Manipulation Language |
| **DSMetaData** | The Data Sources MetaData |
| **DTD** | Document Type Definition |
| **DTP** | The Data Translation Processor |
| **EJB** | Enterprise JavaBeans |
| **FDBS** | Federated Database System |
| **HTML** | Hyper Text Markup Language |
| **IDL** | Interface Definition Language |
| **ISO** | International Standard Organisation |
| **JTS** | Java Transaction Service |
| **MeDInt** | The Mediated Data Integration |
| **MDDL** | The Mediated Data Definition Language |
| **MDM** | The Mediated Data Model |
| **MDRS** | The Mediated Data Representation Structure |
| **MMD** | The Mediated MetaData |
| **MQL** | The Mediated Query Language |
| **ODMG** | Object Database Management Group |
| **ODL** | Object Definition Language |
| **OEM** | Object Exchange Model |
| **OIF** | Object Interchange Format Language |
| **OM** | Object Model |
| **OMG** | Object Management Group |
| **OMMetaData** | The Object Mapping MetaData |

| | |
|---|---|
| **OMT** | Object Modelling Technique |
| **OQL** | Object Query Language |
| **ORB** | Object Request Broker |
| **ORDBMS** | Object Relational Database Management System |
| **QTA** | The Query Transformation Agent |
| **QTP** | The Query Translation Processor |
| **RA** | The Rendering Agent |
| **RDBMS** | Relational Database Management System |
| **RMDRS** | Conflict-Resolved MDRS |
| **RP** | The Registering Processor |
| **SGML** | Standard Generalised Markup Language |
| **SQL** | Structured Query Language |
| **STP** | The Schema Translation Processor |
| **TSMetaData** | The Thesaurus MetaData |
| **W3C** | World Wide Web Consortium |
| XML | Extensible Markup Language |

# Appendix C – Symbols used in EBNF

| Symbol | Indicates |
|---|---|
| = | Defining-symbol |
| ; | Terminator-symbol |
| * | Repetition-symbol |
| \| | Definition-separator-symbol |
| , | Concatenate-symbol |
| - | Except-symbol |
| { } | Repeated sequence |
| [ ] | Optional-sequence |
| ( ) | Grouped sequence |
| ' ' | Quote-symbol |

# Appendix D – MDM Reserved Words

| | | |
|---|---|---|
| attribute | and | character |
| condition | date | float |
| from | in | integer |
| key | operation | or |
| relationship | select | string |
| subtype | user_defined | = |
| > | < | >= |
| <= | <> | |

# Appendix E – Data Definition Language in Relational Data Model

## *UnivDB*

```
CREATE TABLE Author
    (   call_no     CHAR(10)  NOT NULL,
        lname       CHAR(30)  NOT NULL,
        fname       CHAR(50)  NOT NULL,
        PRIMARY KEY (call_no, lname, fname)
        FOREIGN KEY (call_no) REFERENCES Book);

CREATE TABLE Book
    (   call_no     CHAR(10)  NOT NULL,
        name        CHAR(50)  NOT NULL,
        isbn        CHAR(15),
        PRIMARY KEY (call_no));

CREATE TABLE LoanRec
    (   call_no     CHAR(10)  NOT NULL,
        id          CHAR(7)   NOT NULL,
        from        DATE      NOT NULL,
        to          DATE      NOT NULL,
        PRIMARY KEY (call_no)
        FOREIGN KEY (call_no) REFERENCES Book);

CREATE TABLE Staff
    (   id          CHAR(7)   NOT NULL,
        fname       CHAR(30)  NOT NULL,
        lname       CHAR(30)  NOT NULL,
        address     CHAR(50),
        tel_no      CHAR(10),
        sex         CHAR(1),
        dob         DATE,
        salary      DECIMAL(6,2),
        type        CHAR(1)   NOT NULL,
        PRIMARY KEY (id));

CREATE TABLE Student
    (   id          CHAR(7)   NOT NULL,
        fname       CHAR(30)  NOT NULL,
        lname       CHAR(30)  NOT NULL,
        address     CHAR(50),
        tel_no      CHAR(10),
        sex         CHAR(1),
        dob         DATE,
        level       CHAR(1)   NOT NULL,
        PRIMARY KEY (id));

CREATE TABLE Unit
    (   id          CHAR(7)   NOT NULL,
        name        CHAR(30)  NOT NULL,
        PRIMARY KEY (id));

CREATE TABLE Prerequisite
    (   unit_id         CHAR(7)   NOT NULL,
        unit_pre        CHAR(7)   NOT NULL,
        PRIMARY KEY (unit_id, unit_pre)
        FOREIGN KEY (id) REFERENCES Unit);
```

```
CREATE TABLE Lecture
    (   staff_id        CHAR(7)    NOT NULL,
        unit_id         CHAR(7)    NOT NULL,
        PRIMARY KEY (staff_id, unit_id)
        FOREIGN KEY (id) REFERENCES Staff,
        FOREIGN KEY (id) REFERENCES Unit);

CREATE TABLE EnrolRec
    (   student_id      CHAR(7)    NOT NULL,
        unit_id         CHAR(7)    NOT NULL,
        PRIMARY KEY (student_id, unit_id)
        FOREIGN KEY (id) REFERENCES Student,
        FOREIGN KEY (id) REFERENCES Unit);
```

# Appendix F – Data Definition Language in Object Data Model

## *CampusDB*

```
Interface Person {
    attribute       string          id;
    attribute       struct<string fname, string lname>  name;
    attribute       string          address;
    attribute       string          tel_no;
    attribute       string          sex;
    attribute       date            dob;
    relationship    Book            borrow
        inverse     Book::loanby}

Interface Staff:Person {
    attribute       float           salary;}

Interface Lecturer:Staff (key id) {
    relationship    set<Unit>       lecture
        inverse     Unit::lecturedby;}

Interface Student:Person (key:id){
    attribute       <undergrad, postgrad> level;
    relationship    set<Unit>       enrol
        inverse     Unit::enrolledby;}

Interface Book (key call_no) {
    attribute       string          call_no;
    attribute       string          name;
    attribute       set<struct<string fname; string lname>>     author;
    attribute       string          isbn;
    relationship    Person          loanby
        inverse     Person::borrow;}

Interface Course (key_code) {
    attribute       string          code;
    attribute       string          name;
    relationship    Lecturer        lecturedby
        inverse     Lecturer::lecture;
    relationship    set<Student>    enrolledby
        inverse     Student::enrol;
    relationship    set<Course>     hasprerequisite;}
```

# Appendix G – Schemas Representation by MDDL

## The Mediated Data Definition Language (MDDL) – *CampusDB*

```
Set of Objects  =   {Person, Staff, Lecturer, Student, Book, Course}

Person          =   {
            attribute
                id              string
                name            struct(fname string, lname string)
                address         string
                tel_no          string
                sex             character
                dob             date;
            relationship
                Borrow          set(Book)           Book.LoanBy;
            operation
                age();
            }

Staff           =   {
            subtype
                Person;
            attribute
                salary          float;
            key
                id;
            }

Lecturer        =   {
            subtype
                Staff;
            relationship
                Lecture         set(Course)         Course.LecturedBy;
            key
                id;
            }

Student =       {
            subtype
                Person;
            attribute
                level           {undergrad, postgrad};
            relationship
                Enrol           set(Course)         Course.EnrolledBy;
            key
                id;
            }
```

```
Book        =   {
            attribute
                call_no     string
                name        string
                author      set(struct(fname string, lname string))
                isbn        string;
            relationship
                LoanBy      Person                Person.Borrow;
            operation
                new_book()
                loan_book(in Person.id);
            key
                call_no;
            }

Course      =   {
            attribute
                code        string
                name        string;
            relationship
                LecturedBy  Staff          Lecturer.Lecture
                EntolledBy  Student        Student.Enrol
                HasPrerequisite set(Course);
            operation
                new_Course()
                student_enrol(in Student.id);
            key
                code;
            }
```

## The Mediated Data Definition Language (MDDL) – *UnivDB*

```
Set of Objects  =    {Staff, Student, Book, Unit, Prerequisite,
                     Lecture, EnrolRec, LoanRec, Author}

Staff        =   {
            attribute
                id           string
                fname        string
                lname        string
                address      string
                tel_no       string
                sex          character
                dob          date
                salary       float
                type         character;
            relationship
                id           LoanRec.id
                id           Lecture.staff_id;
            key
                id;
            }

Student =   {
            attribute
                id           string
                fname        string
                lname        string
                address      string
                tel_no       string
                sex          character
                dob          date
                level        character;
            relationship
                id           LoanRec.id
                id           EnrolRec.student_id;
            key
                id;
            }

Book         =   {
            attribute
                call_no      string
                name         string
                isbn         string;
            relationship
                call_no      Author.call_no
                call_no      LoanRec.call_no;
            key
                call_no;
            }
```

```
Unit        =   {
            attribute
                id          string
                name        string;
            relationship
                id          Lecture.staff_id
                id          EnrolRec.student_id
                id          Prerequisite::unit_id;
            key
                id;
            }

Prerequisite    =   {
            attribute
                unit_id     string
                unit_pre    string;
            relationship
                unit_id     Unit.id;
            key
                unit_id + unit_pre;
            }

Lecture     =   {
            attribute
                staff_id    string
                unit_id     string;
            relationship
                staff_id    Staff.id
                unit_id     Unit.id;
            key
                staff_id + unit_id;
            }

EnrolRec        =   {
            attribute
                student_id  string
                unit_id     string;
            relationship
                student_id  Student.id
                unit_id     Unit.id;
            key
                student_id + unit_id;
            }

LoanRec     =   {
            attribute
                student_id  string
                unit_id     string;
            relationship
                student_id  Student.id
                unit_id     Unit.id;
            key
                student_id + unit_id;
            }
```

```
Author      =   {
           attribute
               call_no      string
               lname        string
               fname        string;
           relationship
               call_no      Book.call_no;
           key
               call_no + lname + fname;
           }
```

# Appendix H – MDDL implementation by XML

## CampusDB

```
<?xml version="1.0" standalone="no"?>
<DataSource id="00010000000" name="CampusDB">
   <ObjectType id="000100010000" name="Person">
      <Attribute>
         <id id="000100010001" datatype="string"/>
         <name id="000100010002" datatype="user_defined">
            <fname id="000100010003" datatype="string"/>
            <lname id="000100010004" datatype="string"/>
         </name>
         <address id="000100010005" datatype="string"/>
         <tel_no id="000100010006" datatype="string"/>
         <sex id="000100010007" datatype="char"/>
         <dob id="000100010008" datatype="date"/>
      </Attribute>
      <Relationship>
         <borrow id="000100010009" datatype="Book">
            <inverse>Book.Loanby</inverse>
         </borrow>
      </Relationship>
      <Operation>
         <age id="000100010010">
            <datatype>integer</datatype>
         </age>
      </Operation>
   </ObjectType>
   <ObjectType id="000100020000" name="Staff">
      <Subtype>Person</Subtype>
      <Attribute>
         <salary id="000100020001" datatype="float" period="yearly"
currency="USD"/>
      </Attribute>
      <Key>Person.id</Key>
   </ObjectType>
   <ObjectType id="000100030000" name="Lecturer">
      <Subtype>Staff</Subtype>
      <Relationship>
         <lecture id="000100030001" datatype="Course">
            <inverse>Course.LecturedBy</inverse>
         </lecture>
      </Relationship>
      <Key>Person.id</Key>
   </ObjectType>
   <ObjectType id="000100040000" name="Student">
      <Subtype>Person</Subtype>
      <Attribute>
         <Level id="000100040001" datatype="{undergrad|postgrad}"/>
      </Attribute>
      <Relationship>
         <enrol id="000100040002" datatype="Course">
            <inverse>Course.EnrolledBy</inverse>
         </enrol>
      </Relationship>
      <Key>Person.id</Key>
   </ObjectType>
   <ObjectType id="000100050000" name="Book">
      <Attribute>
         <call_no id="000100050001" datatype="string"/>
         <name id="000100050002" datatype="string"/>
         <author id="000100050003" datatype="user_defined">
            <fname id="000100050004" datatype="string"/>
            <lname id="000100050005" datatype="string"/>
         </author>
         <isbn id="000100050006" datatype="string"/>
      </Attribute>
      <Relationship>
         <loanby id="000100050007" datatype="Person">
            <inverse>Person.borrow</inverse>
         </loanby>
      </Relationship>
      <Operation>
         <new_book id="000100050008">
            <datatype>book</datatype>
         </new_book>
         <loan_book id="000100050009">
            <in>person</in>
         </loan_book>
      </Operation>
      <Key>call_no</Key>
   </ObjectType>
   <ObjectType id="000100060000" name="Course">
      <Attribute>
         <code id="000100060001" datatype="string"/>
         <name id="000100060002" datatype="string"/>
      </Attribute>
      <Relationship>
         <lecturedby id="000100060003" datatype="Lecturer">
            <inverse>Lecturer.Lecture</inverse>
         </lecturedby>
         <enrolledby id="000100060004" datatype="Student">
            <inverse>Student.enrol</inverse>
         </enrolledby>
         <hasprerequisite id="000100060005" datatype="set(Course)"/>
      </Relationship>
      <Operation>
         <new_course id="000100060006">
            <datatype>Course</datatype>
         </new_course>
         <student_enrol id="000100060007">
            <in>student</in>
         </student_enrol>
      </Operation>
      <Key>code</Key>
   </ObjectType>
</DataSource>
```

## *UnivDB*

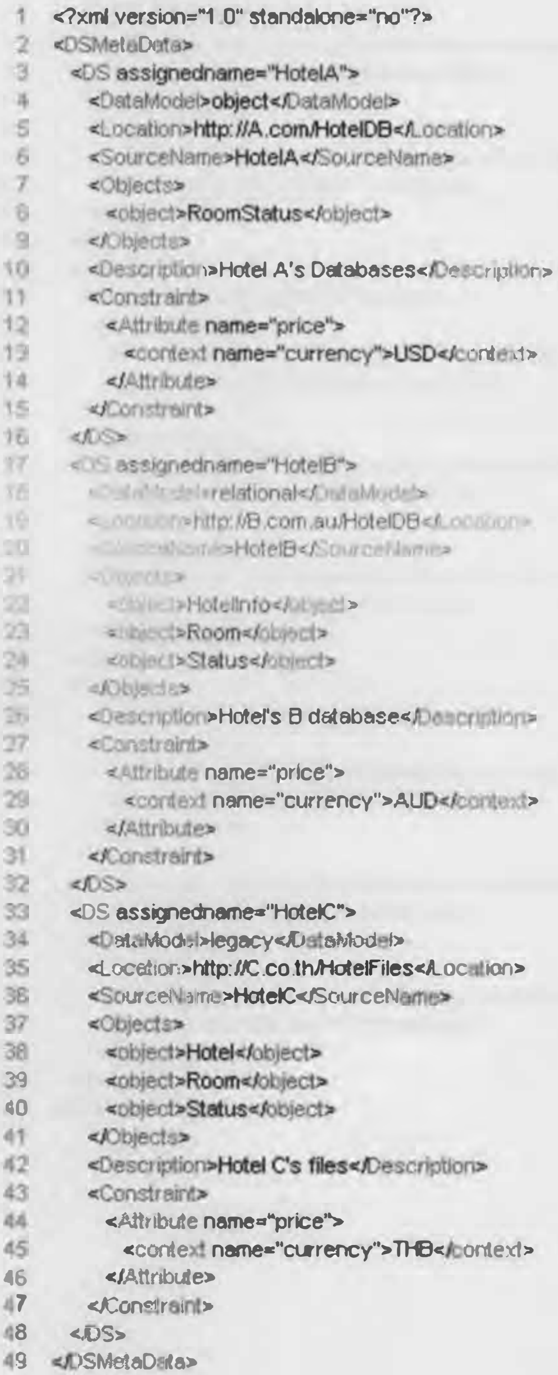
```
<?xml version="1.0" standalone="no"?>
<DataSource id="00020000000" name="UnivDB">
   <ObjectType id="000200010000" name="Staff">
      <Attribute>
         <id id="000200010001" datatype="string"/>
         <fname id="000200010002" datatype="string"/>
         <lname id="000200010003" datatype="string"/>
         <address id="000200010004" datatype="string"/>
         <tel_no id="000200010005" datatype="string"/>
         <sex id="000200010006" datatype="char"/>
         <dob id="000200010007" datatype="date"/>
         <salary id="000200010008" datatype="float"/>
         <type id="000200010009" datatype="char"/>
      </Attribute>
      <Relationship>
         <id id="000200010010">
            <inverse>loanrec.id</inverse>
         </id>
         <id id="000200010011">
            <inverse>lecture.staff_id</inverse>
         </id>
      </Relationship>
      <Key>id</Key>
   </ObjectType>
   <ObjectType id="000200020000" name="Student">
      <Attribute>
         <id id="000200020001" datatype="string"/>
         <fname id="000200020002" datatype="string"/>
         <lname id="000200020003" datatype="string"/>
         <address id="000200020004" datatype="string"/>
         <tel_no id="000200020005" datatype="string"/>
         <sex id="000200020006" datatype="char"/>
         <dob id="000200020007" datatype="date"/>
         <level id="000200020008" datatype="char"/>
      </Attribute>
      <Relationship>
         <id id="00020001009">
            <inverse>loanrec.id</inverse>
         </id>
         <id id="000200010010">
            <inverse>enrolrec.student_id</inverse>
         </id>
      </Relationship>
      <Key>id</Key>
   </ObjectType>
   <ObjectType id="000200030000" name="Book">
      <Attribute>
         <call_no id="000200030001" datatype="string"/>
         <name id="000200030002" datatype="string"/>
         <isbn id="000200030003" datatype="string"/>
      </Attribute>
      <Relationship>
         <call_no id="000200030004">
            <inverse>Author.call_no</inverse>
         </call_no>
         <call_no id="000200030005">
         <inverse>LoanRec.call_no</inverse>
      </call_no>
   </Relationship>
   <Key>call_no</Key>
</ObjectType>
<ObjectType id="000200040000" name="Unit">
   <Attribute>
      <id id="000200040001" datatype="string"/>
      <name id="000200040002" datatype="string"/>
   </Attribute>
   <Relationship>
      <id id="000200040003">
         <inverse>Lecture.staff_id</inverse>
      </id>
      <id id="000200040004">
         <inverse>EnrolRec.student_id</inverse>
      </id>
      <id id="000200040005">
         <inverse>Prerequisite.unit_id</inverse>
      </id>
   </Relationship>
   <Key>id</Key>
</ObjectType>
<ObjectType id="000200050000" name="Prerequisite">
   <Attribute>
      <unit_id id="000200050001" datatype="string"/>
      <unit_pre id="000200050002" datatype="string"/>
   </Attribute>
   <Relationship>
      <unit_id id="000200050003">
         <inverse>Unit.id</inverse>
      </unit_id>
   </Relationship>
   <Key>unit_id+unit_pre</Key>
</ObjectType>
<ObjectType id="000200060000" name="Lecture">
   <Attribute>
      <staff_id id="000200060001" datatype="string"/>
      <unit_id id="000200060002" datatype="string"/>
   </Attribute>
   <Relationship>
      <staff_id id="000200060003">
         <inverse>Staff.id</inverse>
      </staff_id>
      <unit_id id="000200060004">
         <inverse>Unit.id</inverse>
      </unit_id>
   </Relationship>
   <Key>staff_id+unit_id</Key>
</ObjectType>
<ObjectType id="000200070000" name="EnrolRec">
   <Attribute>
      <student_id id="000200070001" datatype="string"/>
      <unit_id id="000200070002" datatype="string"/>
   </Attribute>
   <Relationship>
      <student_id id="000200070003">
         <inverse>Student.id</inverse>
      </student_id>
```

```
            <unit_id id="000200070004">
               <inverse>Unit.id</inverse>
            </unit_id>
         </Relationship>
         <Key>student_id+unit_id</Key>
      </ObjectType>
      <ObjectType id="000200080000" name="LoanRec">
         <Attribute>
            <student_id id="000200080001" datatype="string"/>
            <unit_id id="000200080002" datatype="string"/>
         </Attribute>
         <Relationship>
            <student_id id="000200080003">
               <inverse>Student.id</inverse>
            </student_id>
            <unit_id id="000200080004">
               <inverse>Unit.id</inverse>
            </unit_id>
         </Relationship>
         <Key>student_id+unit_id</Key>
      </ObjectType>
      <ObjectType id="000200090000" name="Author">
         <Attribute>
            <call_no id="000200090001" datatype="string"/>
            <lname id="000200090002" datatype="string"/>
            <fname id="000200090003" datatype="string"/>
         </Attribute>
         <Relationship>
            <call_no id="000200090006">
               <inverse>Book.call_no</inverse>
            </call_no>
         </Relationship>
         <Key>call_no+lname+fname</Key>
      </ObjectType>
   </DataSource>
```

# Appendix I – MMD Representations in XML

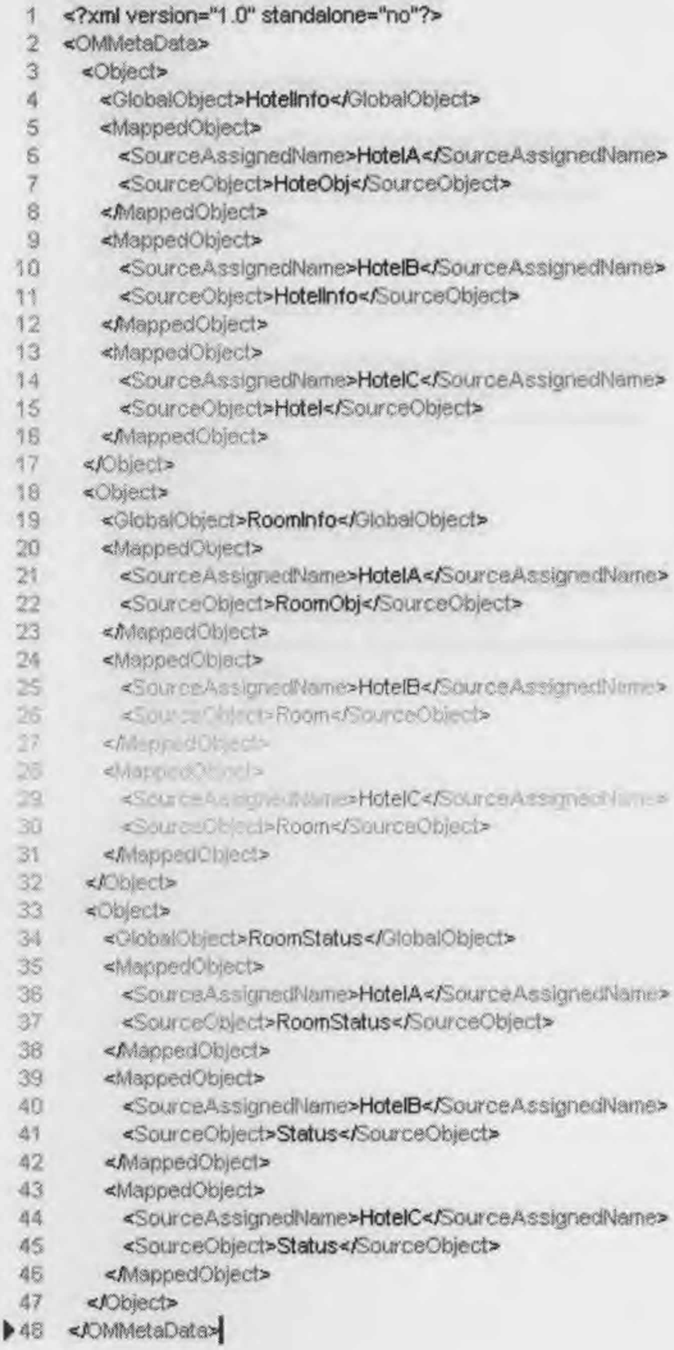
## Test Problem 1 – Hotel Chain Information System

### DataSource MetaData (DSMetaData)



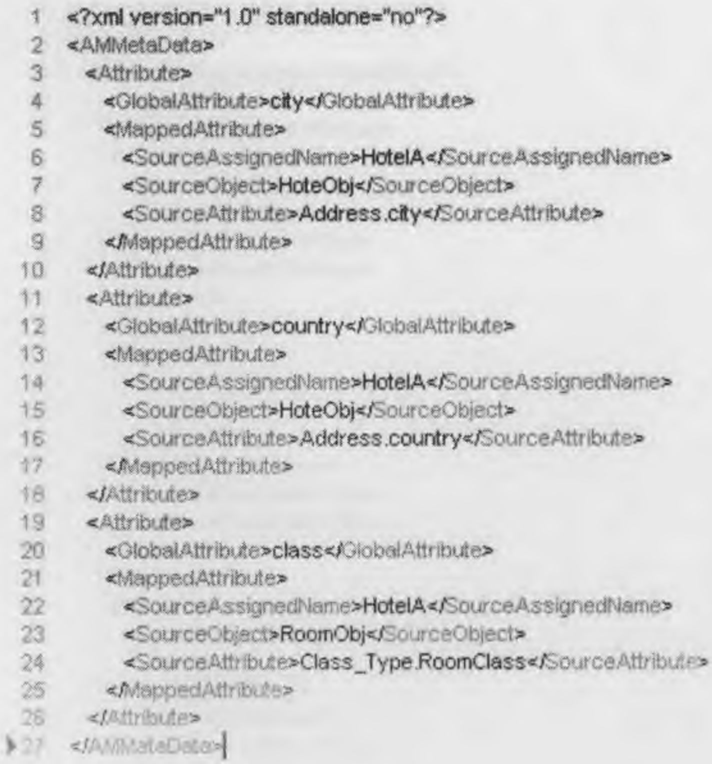
```
<?xml version="1.0" standalone="no"?>
<DSMetaData>
  <DS assignedname="HotelA">
    <DataModel>object</DataModel>
    <Location>http://A.com/HotelDB</Location>
    <SourceName>HotelA</SourceName>
    <Objects>
      <object>RoomStatus</object>
    </Objects>
    <Description>Hotel A's Databases</Description>
    <Constraint>
      <Attribute name="price">
        <context name="currency">USD</context>
      </Attribute>
    </Constraint>
  </DS>
  <DS assignedname="HotelB">
    <DataModel>relational</DataModel>
    <Location>http://B.com.au/HotelDB</Location>
    <SourceName>HotelB</SourceName>
    <Objects>
      <object>HotelInfo</object>
      <object>Room</object>
      <object>Status</object>
    </Objects>
    <Description>Hotel's B database</Description>
    <Constraint>
      <Attribute name="price">
        <context name="currency">AUD</context>
      </Attribute>
    </Constraint>
  </DS>
  <DS assignedname="HotelC">
    <DataModel>legacy</DataModel>
    <Location>http://C.co.th/HotelFiles</Location>
    <SourceName>HotelC</SourceName>
    <Objects>
      <object>Hotel</object>
      <object>Room</object>
      <object>Status</object>
    </Objects>
    <Description>Hotel C's files</Description>
    <Constraint>
      <Attribute name="price">
        <context name="currency">THB</context>
      </Attribute>
    </Constraint>
  </DS>
</DSMetaData>
```

## Object Mapping MetaData (OMMetaData)



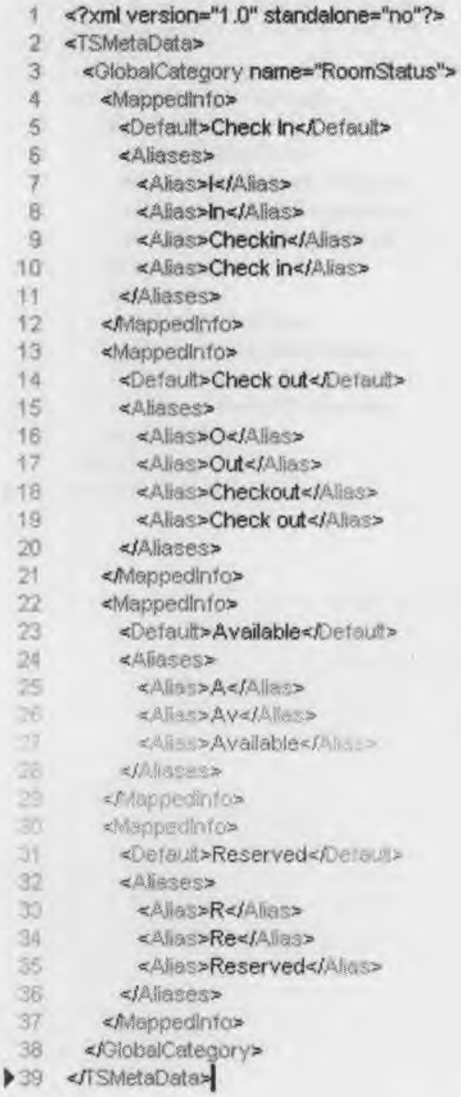
```
<?xml version="1.0" standalone="no"?>
<OMMetaData>
  <Object>
    <GlobalObject>HotelInfo</GlobalObject>
    <MappedObject>
      <SourceAssignedName>HotelA</SourceAssignedName>
      <SourceObject>HoteObj</SourceObject>
    </MappedObject>
    <MappedObject>
      <SourceAssignedName>HotelB</SourceAssignedName>
      <SourceObject>HotelInfo</SourceObject>
    </MappedObject>
    <MappedObject>
      <SourceAssignedName>HotelC</SourceAssignedName>
      <SourceObject>Hotel</SourceObject>
    </MappedObject>
  </Object>
  <Object>
    <GlobalObject>RoomInfo</GlobalObject>
    <MappedObject>
      <SourceAssignedName>HotelA</SourceAssignedName>
      <SourceObject>RoomObj</SourceObject>
    </MappedObject>
    <MappedObject>
      <SourceAssignedName>HotelB</SourceAssignedName>
      <SourceObject>Room</SourceObject>
    </MappedObject>
    <MappedObject>
      <SourceAssignedName>HotelC</SourceAssignedName>
      <SourceObject>Room</SourceObject>
    </MappedObject>
  </Object>
  <Object>
    <GlobalObject>RoomStatus</GlobalObject>
    <MappedObject>
      <SourceAssignedName>HotelA</SourceAssignedName>
      <SourceObject>RoomStatus</SourceObject>
    </MappedObject>
    <MappedObject>
      <SourceAssignedName>HotelB</SourceAssignedName>
      <SourceObject>Status</SourceObject>
    </MappedObject>
    <MappedObject>
      <SourceAssignedName>HotelC</SourceAssignedName>
      <SourceObject>Status</SourceObject>
    </MappedObject>
  </Object>
</OMMetaData>
```

## Attribute Mapping MetaData (AMMetaData)



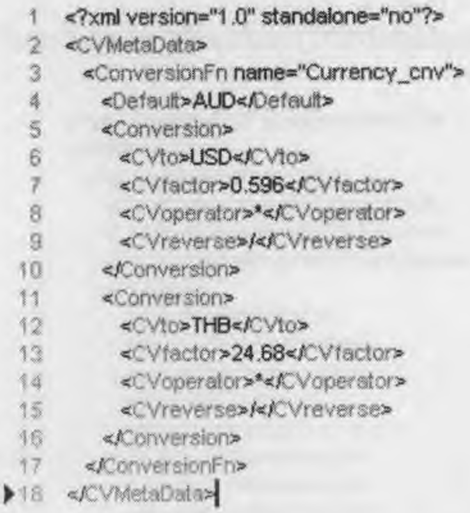
```
<?xml version="1.0" standalone="no"?>
<AMMetaData>
  <Attribute>
    <GlobalAttribute>city</GlobalAttribute>
    <MappedAttribute>
      <SourceAssignedName>HotelA</SourceAssignedName>
      <SourceObject>HoteObj</SourceObject>
      <SourceAttribute>Address.city</SourceAttribute>
    </MappedAttribute>
  </Attribute>
  <Attribute>
    <GlobalAttribute>country</GlobalAttribute>
    <MappedAttribute>
      <SourceAssignedName>HotelA</SourceAssignedName>
      <SourceObject>HoteObj</SourceObject>
      <SourceAttribute>Address.country</SourceAttribute>
    </MappedAttribute>
  </Attribute>
  <Attribute>
    <GlobalAttribute>class</GlobalAttribute>
    <MappedAttribute>
      <SourceAssignedName>HotelA</SourceAssignedName>
      <SourceObject>RoomObj</SourceObject>
      <SourceAttribute>Class_Type.RoomClass</SourceAttribute>
    </MappedAttribute>
  </Attribute>
</AMMetaData>
```

## Thesaurus MetaData (TSMetaData)



```
<?xml version="1.0" standalone="no"?>
<TSMetaData>
  <GlobalCategory name="RoomStatus">
    <MappedInfo>
      <Default>Check In</Default>
      <Aliases>
        <Alias>I</Alias>
        <Alias>In</Alias>
        <Alias>Checkin</Alias>
        <Alias>Check in</Alias>
      </Aliases>
    </MappedInfo>
    <MappedInfo>
      <Default>Check out</Default>
      <Aliases>
        <Alias>O</Alias>
        <Alias>Out</Alias>
        <Alias>Checkout</Alias>
        <Alias>Check out</Alias>
      </Aliases>
    </MappedInfo>
    <MappedInfo>
      <Default>Available</Default>
      <Aliases>
        <Alias>A</Alias>
        <Alias>Av</Alias>
        <Alias>Available</Alias>
      </Aliases>
    </MappedInfo>
    <MappedInfo>
      <Default>Reserved</Default>
      <Aliases>
        <Alias>R</Alias>
        <Alias>Re</Alias>
        <Alias>Reserved</Alias>
      </Aliases>
    </MappedInfo>
  </GlobalCategory>
</TSMetaData>
```

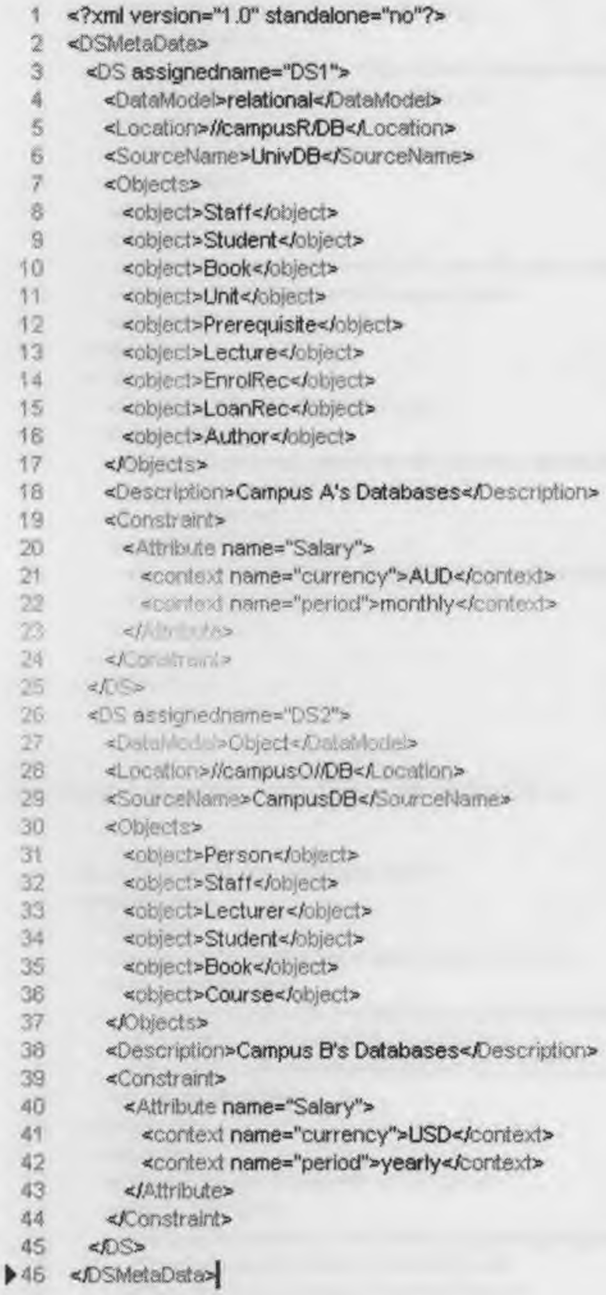
## Conversion MetaData (CVMetaData)



```
<?xml version="1.0" standalone="no"?>
<CVMetaData>
  <ConversionFn name="Currency_cnv">
    <Default>AUD</Default>
    <Conversion>
      <CVto>USD</CVto>
      <CVfactor>0.596</CVfactor>
      <CVoperator>*</CVoperator>
      <CVreverse>/</CVreverse>
    </Conversion>
    <Conversion>
      <CVto>THB</CVto>
      <CVfactor>24.68</CVfactor>
      <CVoperator>*</CVoperator>
      <CVreverse>/</CVreverse>
    </Conversion>
  </ConversionFn>
</CVMetaData>
```

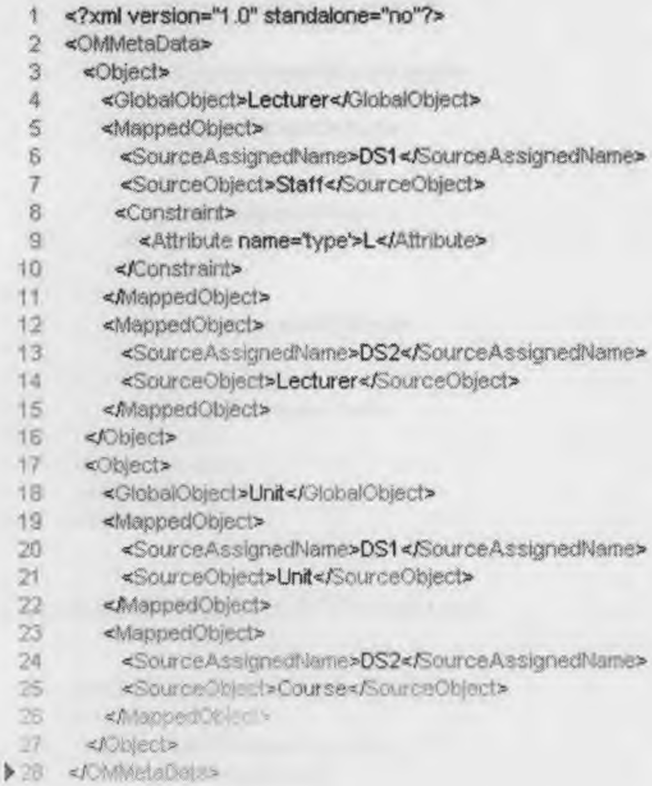
## Test Problem 2 – University Information System

### Data Source MetaData (DSMetaData)



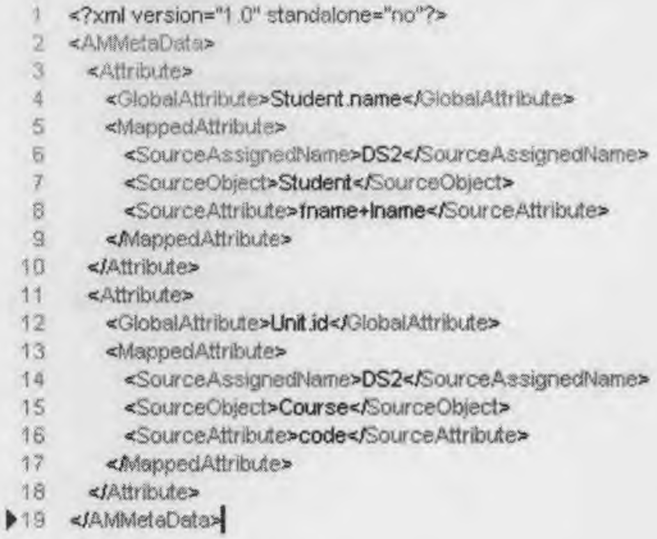
```
<?xml version="1.0" standalone="no"?>
<DSMetaData>
  <DS assignedname="DS1">
    <DataModel>relational</DataModel>
    <Location>//campusR/DB</Location>
    <SourceName>UnivDB</SourceName>
    <Objects>
      <object>Staff</object>
      <object>Student</object>
      <object>Book</object>
      <object>Unit</object>
      <object>Prerequisite</object>
      <object>Lecture</object>
      <object>EnrolRec</object>
      <object>LoanRec</object>
      <object>Author</object>
    </Objects>
    <Description>Campus A's Databases</Description>
    <Constraint>
      <Attribute name="Salary">
        <context name="currency">AUD</context>
        <context name="period">monthly</context>
      </Attribute>
    </Constraint>
  </DS>
  <DS assignedname="DS2">
    <DataModel>Object</DataModel>
    <Location>//campusO//DB</Location>
    <SourceName>CampusDB</SourceName>
    <Objects>
      <object>Person</object>
      <object>Staff</object>
      <object>Lecturer</object>
      <object>Student</object>
      <object>Book</object>
      <object>Course</object>
    </Objects>
    <Description>Campus B's Databases</Description>
    <Constraint>
      <Attribute name="Salary">
        <context name="currency">USD</context>
        <context name="period">yearly</context>
      </Attribute>
    </Constraint>
  </DS>
</DSMetaData>
```

## Object Mapping MetaData (OMMetaData)



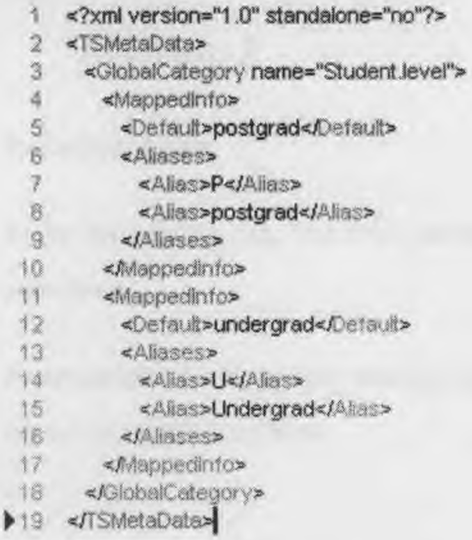
```
<?xml version="1.0" standalone="no"?>
<OMMetaData>
  <Object>
    <GlobalObject>Lecturer</GlobalObject>
    <MappedObject>
      <SourceAssignedName>DS1</SourceAssignedName>
      <SourceObject>Staff</SourceObject>
      <Constraint>
        <Attribute name='type'>L</Attribute>
      </Constraint>
    </MappedObject>
    <MappedObject>
      <SourceAssignedName>DS2</SourceAssignedName>
      <SourceObject>Lecturer</SourceObject>
    </MappedObject>
  </Object>
  <Object>
    <GlobalObject>Unit</GlobalObject>
    <MappedObject>
      <SourceAssignedName>DS1</SourceAssignedName>
      <SourceObject>Unit</SourceObject>
    </MappedObject>
    <MappedObject>
      <SourceAssignedName>DS2</SourceAssignedName>
      <SourceObject>Course</SourceObject>
    </MappedObject>
  </Object>
</OMMetaData>
```

## Attribute Mapping MetaData (AMMetaData)



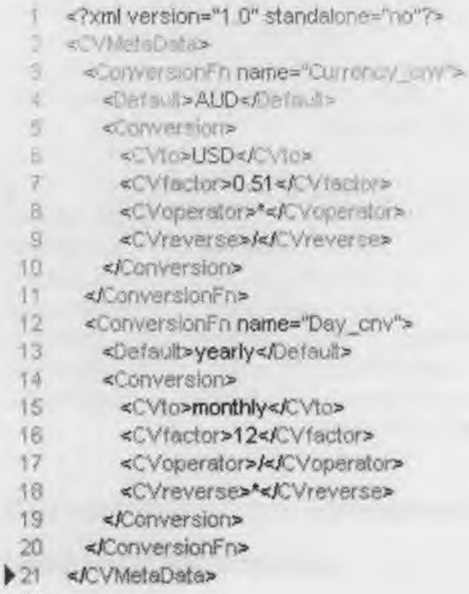
```
<?xml version="1.0" standalone="no"?>
<AMMetaData>
  <Attribute>
    <GlobalAttribute>Student.name</GlobalAttribute>
    <MappedAttribute>
      <SourceAssignedName>DS2</SourceAssignedName>
      <SourceObject>Student</SourceObject>
      <SourceAttribute>fname+lname</SourceAttribute>
    </MappedAttribute>
  </Attribute>
  <Attribute>
    <GlobalAttribute>Unit.id</GlobalAttribute>
    <MappedAttribute>
      <SourceAssignedName>DS2</SourceAssignedName>
      <SourceObject>Course</SourceObject>
      <SourceAttribute>code</SourceAttribute>
    </MappedAttribute>
  </Attribute>
</AMMetaData>
```

## Thesaurus MetaData (TSMetaData)



```
<?xml version="1.0" standalone="no"?>
<TSMetaData>
  <GlobalCategory name="Student.level">
    <MappedInfo>
      <Default>postgrad</Default>
      <Aliases>
        <Alias>P</Alias>
        <Alias>postgrad</Alias>
      </Aliases>
    </MappedInfo>
    <MappedInfo>
      <Default>undergrad</Default>
      <Aliases>
        <Alias>U</Alias>
        <Alias>Undergrad</Alias>
      </Aliases>
    </MappedInfo>
  </GlobalCategory>
</TSMetaData>
```

## Conversion MetaData (CVMetaData)



```
<?xml version="1.0" standalone="no"?>
<CVMetaData>
  <ConversionFn name="Currency_cnv">
    <Default>AUD</Default>
    <Conversion>
      <CVto>USD</CVto>
      <CVfactor>0.51</CVfactor>
      <CVoperator>*</CVoperator>
      <CVreverse>/</CVreverse>
    </Conversion>
  </ConversionFn>
  <ConversionFn name="Day_cnv">
    <Default>yearly</Default>
    <Conversion>
      <CVto>monthly</CVto>
      <CVfactor>12</CVfactor>
      <CVoperator>/</CVoperator>
      <CVreverse>*</CVreverse>
    </Conversion>
  </ConversionFn>
</CVMetaData>
```

# *Appendix J – The MeDInt Integration Process of Test Problem 2*

## Initialising step

In the initialising step, the five prerequisites are prepared before any integration processes.

**Prerequisite 1** - New data sources have to be registered in the Data Source MetaData (DSMetaData).

```
{
AssignedName  DS1;
DataModel     relational;
Location      //campusR/DB;
SourceName    UnivDB;
Objects       Staff, Student, Book, Unit, Prerequisite, Lecture, EnrolRec,
LoanRec,
                   Author;
Description   Campus A's database;
Constraint    Salary    (Currency = "AUD",
                        Period= "monthly");
}
{
AssignedName  DS2;
DataModel     Object;
Location      //campusO//DB;
SourceName    CampusDB;
Objects       Person, Staff, Lecturer, Student, Book, Course;
Description   Campus B's database;
Constraint    Salary    (Currency = "USD",
                        Period = "yearly");
}
```

**Prerequisite 2** - Entity equivalences have to be indicated in the Object Mapping MetaData (OMMetaData).

```
{
GlobalObject       Lecturer
MappedObject       SourceAssignedName      DS1
                   SourceObject            Staff
                   Constraint              {attribute     type="L"
                                           }
MappedObject       SourceAssignedName      DS2
                   SourceObject            Lecturer
}
```

```
{
GlobalObject       Unit
MappedObject       SourceAssignedName      DS1
                   SourceObject            Unit
MappedObject       SourceAssignedName      DS2
                   SourceObject            Course
}
```

**Prerequisite 3** - Attribute equivalences have to be indicated in the Attribute Mapping MetaData (AMMetaData).

```
{
GlobalAttribute    Student.Name
MappedAttribute    SourceAssignedName      DS2
                   SourceObject            Student
                   SourceAttribute         fname+lname
}
{
GlobalAttribute    Unit.id
MappedAttribute    SourceAssignedName      DS2
                   SourceObject            Course
                   SourceAttribute         code
}
```

**Prerequisite 4** - Data equivalences have to be defined in the Thesaurus MetaData (TSMetaData).

```
{
GlobalCategory     Student.level
MappedInfo         Default                 postgrad
                   Aliases
                        {
                             Alias                 P
                             Alias                 postgrad
                        }
MappedInfo         Default                 undergrad
                   Aliases
                        {
                             Alias                 U
                             Alias                 Undergrad
                        }
}
```

**Prerequisite 5** - Conversion factors of different units have to be specified in the Conversion MetaData (CVMetaData).

```
{Currency_cnv      Default            AUD
                   {
                        CVto          USD
                        CVfactor      0.51
                        CVoperator    *
                        CVreverse     /
                   }
}
```

```
{Day_cnv        Default             yearly
                    {
                        CVto            monthly
                        CVfactor        12
                        CVoperator      /
                        CVreverse       *
                    }
                    {
                        CVto            daily
                        CVfactor        365
                        CVoperator      /
                        CVreverse       *
                    }
}
```

## Query 1

The first query example is a request for the id and name of postgraduate students who enrol 'CSP1143' from both *DS1* and *DS2*.

```
SELECT          Student.id, Student.name
FROM            Student, Unit
IN              DS1, DS2
CONDITION       Unit.id = 'CSP1143' and
                Student.level="postgrad";
```

The major task of the MeDInt Mediator after getting a query from a client is to decompose the query to subqueries and to distribute subqueries to associated wrappers. This task is assigned to QTA. Before doing this, QTA has to fetch object schema definitions which are related to the query.

### The Process of Fetching Object Schema Definitions

Following the algorithm stated in the Process *FetchDef(Đ, Ö)* (See Chapter 6), from the query, DSMetaData, and OMMetaData, QTA realises that the required object schema  are as shown in Table J.1.

TABLE J.1 REQUIRED OBJECT SCHEMA DEFINITIONS OF THE QUERY 1 OF TEST PROBLEM 2

| DS1 | DS2 |
|---|---|
| Student | Student |
| Unit | Course |

QTA send requests for the MDDLs of those object to STPs of associated wrappers.

### Schema Translation Processes

The STPs, by RSchmTransl(Ši,Ŏj), OSchmTransl(Ši,Ŏj), and LSchmTransl(Ši,Ŏj) processes (See Chapter 7), translate the disparate object schema definitions into MDDLs.

From DS1

```
Student     =   {
            attribute
                id          string;
                fname       string;
                lname       string;
                address     string;
                tel_no      string;
                sex         character;
                dob         date;
                level       character;
            relationship
                id          LoanRec.id;
                id          EnrolRec.student_id;
            key
                id;
            }
Unit        =   {
            attribute
                id          string;
                name        string;
            relationship
                id          Lecture.staff_id;
                id          EnrolRec.student_id;
                id          Prerequisite::unit_id;
            key
                id;
            }
```

From DS2

```
Student     =   {
            subtype
                Person;
            attribute
                level       {undergrad, postgrad};
            relationship
                Enrol       set(Course)   Course.EnrolledBy;
            key
                id;
            }
Course      =   {
            attribute
                code        string;
                name        string;
            relationship
                LecturedBy  Staff         Lecturer.Lecture;
                EntolledBy  Student       Student.Enrol;
                HasPrerequisite set(Course);
            operation
                new_Course();
                student_enrol(in Student.id);
            key
                id;
            }
```

From the above MDDLs, the *FetchDef(Đ, Ö)* process also recognises that there are further data definitions (*EnrolRec and Person*) required because of the relationship (between *Unit* and *Student*) and because of the generalisation (*Student* is a subtype of *Person*). Then, QTA sends requests to RWrap and OWrap.

```
Person     =   {
            attribute
                id          string;
                name        struct(fname string, lname string);
                address     string;
                tel_no      string;
                sex         character;
                dob         date;
            relationship
                Borrow      set(Book)           Book.LoanBy;
            operation
                age();
            }

EnrolRec   =   {
            attribute
                student_id  string;
                unit_id     string;
            relationship
                student_id  Student.id;
                unit_id     Unit.id;
            key
                student_id + unit_id;
```

### Query Decomposing Process

Now, QTA has enough object schema definitions for decomposing the query by the *Qtransform(Ä, Đ, Ö, Ç)* process (See Chapter 6)).

All the object and attribute identifiers defined on the users' query are global identifiers which can be mapped to local identifiers with the assistance of information in OMMetaData and AMMetaData. From TSMetaData and CVMetaData, attribute values and contexts will be converted to the corresponding source values and contexts.

MQL to DS1

```
SELECT        Student.id, Student.fname, Student.lname
FROM          Student, Unit, Enrolrec
IN            DS1
CONDITION     Unit.id = 'CSP1143' and
              Student.level ="P" and
              Student.id = EnrolRec.student_id and
              Unit.id = Enrolrec.unit_id;
```

The relationship conditions therefore between *Student* and *Unit* have been created.

MQL to DS2

```
SELECT        Student.ed, Student.name
FROM          Student, Course
IN            DS2
CONDITION     Course.code ='CSP1143' and
              Student.level ="postgrad";
```

### Creating a Pre-defined Template Process

QTA also prepares a template in MDRS format.

```
(Student.id, Student.name)
```

### Query Translation Processes

Each subquery will be sent to the QTP of its associated wrapper for query translation which is performed by the SQLGen(χ) or OQLGen(χ) processes (See Chapter 7).

SQL to DS1

```
SELECT        Student.id, Student.fname, Student.lname
FROM          Student, Unit, EnrolRec
WHERE         Unit.id = 'CSP1143' and
              Student.level ="P" and
              Student.id = EnrolRec.student_id and
              Unit.id = EnrolRec.student_id;
```

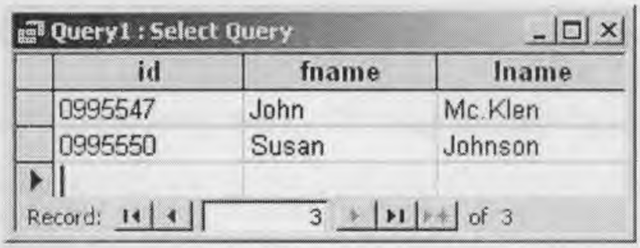
OQL to DS2

```
SELECT        Student.id, Student.name
FROM          Student, Course
WHERE         Course.code = 'CSP1143' and
              Student.level ="postgrad";;
```

### Data Translation Processes

The subqueries above will be performed by the query processing of the local database management systems. Then, the query results will be returned to wrappers. The DTPs will translate any query results which are in disparate models to MDRS:

### DS1



Query1 : Select Query

| id | fname | lname |
|---|---|---|
| 0995547 | John | Mc.Klen |
| 0995550 | Susan | Johnson |

which are in this format

{("0995547","John","Mc.Klen"),("0995550","Susan","Johnson")}

### DS2

{("0995152", ("Jame", "Carter")), ("0994521", ("Catherine","Foster"))}

However, the results still cannot be integrated because they are still in different contexts.

### Applying MDRS Results to the Pre-defined Template Process

The result from DS1 still needs the conflict resolving process *ApplTemp(ρ, τ, θ)* performed by CRA to apply it to the predefined template.

**(Student.id, Student.name)**

### DS1

{("0995547","John","Mc.Klen"),("0995550","Susan","Johnson")}

CRA checks the attribute of each result against the structure and contexts of that attribute in the predefined template. If the structure or the contexts are not the same, then conversion functions will be called from CVMetaData to convert the data to a structure and context corresponding to the predefined template. Then, its semantic aliases will be check to convert any Naming conflicts in the semantic level.

The first result from DS1 needs modification. Attribute mapping information from AMMetaData is required so that the global attribute *student.name* can be mapped to *student.fname* concatenating to *student.lname*. Thus, MDRS from DS1 has to be applied to:

```
(Student.id, Student.fname+Student.lname)
{(“0995547”,”John Mc.Klen”), (“0995550”,”Susan Johnson”)}
```

**Integrating the Mediated Data Representation Structure Process**

Now all query results can be integrated by CP using the union operator.

```
{(“0995547”,”John Mc.Klen”),
(“0995550”,”Susan Johnson”),
(“0995152”, “Jame Carter”),
(“0994521”, “Catherine Foster”)}
```

**Generating the Integrated Result Process**

Finally, RA can present the integrated query result to the user as shown in Table J.2.

TABLE J.2 INTEGRATED RESULT OF THE QUERY 1 OF TEST PROBLEM 2

| Student.id | Student.name |
|---|---|
| 0995547 | John Mc.Klen |
| 0995550 | Susan Johnson |
| 0995152 | Jame Carter |
| 0994521 | Catherine Foster |

**<u>Query 2</u>**

A user may want to get the id and yearly salary of staff who earns less than 50,000 AUD$ from *UnivDB* and *CampusDB*. This query initiates conflicts which are different from the first query.

```
Select       Staff.id, Staff.salary(currency="AUD", period="yearly")
From         Staff
In           DS1, DS2
Condition    Staff.salary(currency="AUD", period="yearly") < 50000;
```

In this query example, a Scaling conflict is added. The submitted query needs yearly salary information from *UnivDB* and *CampusDB* in Australian dollars, but in the data sources registered information in DSMetaData, the currency using in *CampusDB* is

US dollars and salary is quoted on a monthly basis in *UnivDB*. Therefore, the condition in the query submitted to *CampusDB* has to be converted to US dollars and then after getting the result from *CampusDB*, again the result in US dollars has to be converted back into Australian dollars. Moreover, the query submitted to UnivDB has to be transformed into a monthly basis to compare to data in the source, and the result has to be converted back into a yearly basis by the query requested.

The five prerequisites have been defined at the beginning step, so only maintenance will be required when there is schema modification made to the local data sources.

### The Process of Fetching Object Schema Definitions

Following the algorithm stated in Process *FetchDef(Đ, Ö)* (See Chapter 6), from the query, DSMetaData, and OMMetaData, QTA recognises that required object schema required are as shown in Table J.3.

TABLE J.3 REQUIRED OBJECT SCHEMA DIFINITIONS OF THE QUERY 2 OF TEST SAMPLE 2

| DS1 | DS2 |
|---|---|
| Staff | Staff |

QTA sends request for the MDDLs of those objects to the STPs of the associated wrappers.

### Schema Translation Processes

The STPs, by the RSchmTransl(Ši,Ŏj), OSchmTransl(Ši,Ŏj), and LSchmTransl(Ši,Ŏj) processes (See Chapter 7), translate the disparate object schema definitions into MDDLs.

From DS1

```
Staff          =   {
            attribute
                id           string;
                fname        string;
                lname        string;
                address      string;
                tel_no       string;
                sex          character;
                dob          date;
                salary       float;
                type         character;
            relationship
                id          LoanRec.id;
                id          Lecture.staff_id;
            key
                id;
            }
```

From DS2

```
Staff       =   {
            subtype
                Person;
            attribute
                salary      float,
            key
                id
            }
Unit        =   {
            attribute
                id          string;
                name        string;
            relationship
                id          Lecture.staff_id;
                id          EnrolRec.student_id;
                id          Prerequisite::unit_id;
            key
                id;
            }
```

From the above MDDLs, the *FetchDef(Ð, Ö)* process also recognises that there are further data definitions (*Person*) required because of the generalisation (*Staff* is a subtype of *Person*). Then, QTA sends another request to OWrap.

```
Person      =   {
            attribute
                id          string;
                name        struct(fname string, lname string);
                address     string;
                tel_no      string;
                sex         character;
                dob         date;
            relationship
                Borrow      set(Book)           Book.LoanBy;
            operation
                age();
            }
```

**Query Decomposing Process**

Now, QTA has enough object schema definitions for decomposing the query by the *Qtransform(Ä, Ð, Ö, Ç)* process (See Chapter 6)).

All the object and attribute identifiers defined on the users' query are global identifiers which can be mapped to local identifiers with the assistance of information in OMMetaData and AMMetaData. From TSMetaData and CVMetaData, attribute values and contexts will be converted to the corresponding source values and contexts.

Currency and period properties are specified in the requested query. The conflicts are presented in Table J.4.

TABLE J.4 EXISTING CONFLICTS OF THE QUERY 2 OF TEST SAMPLE 2

| Salary | Query | Data Sources | |
|---|---|---|---|
| | | DS1 | DS2 |
| Currency | AUD | AUD | *USD* |
| Period | Yearly | *monthly* | yearly |

The Salary in DS1 is monthly so the conversion *Day_cnv(50000, yearly, monthly)* is required to transform it to *salary(50000, "AUD", "yearly")*. From CVMetaData, the factor of the conversion is *12*[1] to change it to *salary(50000/12, "AUD", "monthly")*. *Day_cnv(50000, yearly, monthly) = 50000/12 = 4166.67*. Therefore, MQL to DS1 is:

```
SELECT          Staff.id, Staff.salary(currency="AUD", period="monthly")
FROM            Staff
IN              DS1
CONDITION       Staff.salary(currency="AUD", period="monthly") < 4166.67;
```

In addition, the currency used in DS2 is in US dollars, so conversion from *50000 AUD* to *USD* is needed. *Currency_cnv(50000, AUD, US)* is called. The factor is *0.51*[1]. *Currency_cnv(50000, AUD, US) = 50000*0.51 = 25500*. Therefore, MQL to DS2 is:

```
SELECT          Staff.id, Staff.salary(currency="USD", period="yearly")
FROM            Staff
IN              DS2
CONDITION       Staff.salary(currency="USD", period="yearly") < 25500;
```

### Creating a Pre-defined Template Process

QTA also prepares a template in MDRS format.

```
(Staff.id, Staff.salary (currency="AUD", period="yearly") )
```

[1] From CVMetaData

### Query Translation Processes

Each subquery will be sent to the QTP of its associated wrapper for query translation which is performed by the SQLGen($\chi$) or OQLGen($\chi$) process (See Chapter 7).

SQL to DS1

```
SELECT      Staff.id, Staff.salary
FROM        Staff
WHERE       Staff.salary < 4166.67;
```

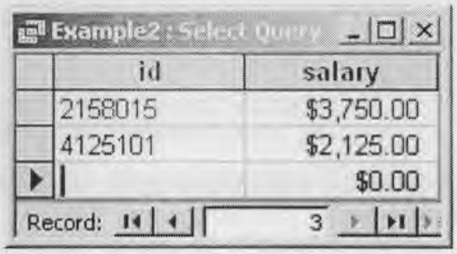
OQL to DS2

```
SELECT      Staff.id, Staff.salary
FROM        Staff
WHERE       Staff.salary < 25500;
```

### Data Translation Processes

The subqueries above are performed by query processing of local database management systems. Then, query results are returned to wrappers. The DTPs will translate any query results which are in disparate models into MDRS.

DS1



| id | salary |
|---|---|
| 2158015 | $3,750.00 |
| 4125101 | $2,125.00 |
|  | $0.00 |

Which has this format

```
(Staff.id, Staff.salary (currency="AUD", period="monthly))

{("2158015", 3750.00(currency="AUD", period="monthly)),
("4125101",2125.00(currency="AUD", period="monthly))}
```

### DS2

```
(Staff.id, Staff.salary (currency="USD", period="yearly))

{("1542545", 15200.00(currency="USD", period="yearly)),
("1478523",25000.00(currency="USD", period="yearly))}
```

However, the results still cannot be integrated because they are still in a different currency and period basis.

### Applying MDRS Results to the Pre-defined Template Process

The results from DS1 and DS2 still need the conflict resolving process *ApplTemp(ρ, τ, θ)* performed by CRA to apply it to the predefined template.

**(Staff.id, Staff.salary (currency="AUD", period="yearly") )**

There is no translation required for *Staff.id* in either source. However, *Staff.salary* both in *DS1* and *DS2* needs to be translated into "AUD" currency on a "yearly" basis according to the pre-defined template.

### DS1

```
{("2158015", 3750.00(currency="AUD", period="monthly)),
("4125101",2125.00(currency="AUD", period="monthly))}
```

converted by *Day_cnv(salary, "monthly", "yearly")* which is:

```
{("2158015", 45000.00(currency="AUD", period="yearly")),
"4125101",25500.00(currency="AUD", period="yearly"))}
```

### DS2

```
{("1542545", 15200.00(currency="USD", period="yearly)),
("1478523",25000.00(currency="USD", period="yearly))}
```

converted by *Currency_cnv(salary, "USD", "AUD")*. Reverse direction conversions are calculated, because there is only an AUD to US conversion factor available in Semantic MetaData.

```
{("1542545", 29803.92(currency="AUD", period="yearly")),
("1478523",49019.61(currency="AUD", period="yearly"))}
```

### Integrating the Mediated Data Representation Structure Process

Now all query results can be integrated by CP using the union operator.

```
{("2158015", 45000.00(currency="AUD", period="yearly")),
("4125101",25500.00(currency="AUD", period="yearly")),
("1542545", 29803.92(currency="AUD", period="yearly")),
("1478523",49019.61(currency="AUD", period="yearly"))}
```

### Generating the Integrated Result Process

RA finally can present the integrated query result to the user as shown in Table J.4.

TABLE J.5 INTEGRATED RESULT OF THE QUERY 2 OF TEST SAMPLE 2

| Staff.id | Staff.salary(currency="AUD", period="yearly") |
|---|---|
| 2158015 | 45000.00 |
| 4125101 | 25500.00 |
| 1542545 | 29803.92 |
| 1478523 | 49019.61 |

## Appendix K – Published Papers

Mukviboonchai, S., & Chirathamjaree, C. (2001, 1-5 July). *XMInt: a mediated integration model.* Paper presented at the Eleventh Annual International Symposium of the International Council On Systems Engineering (INCOSE 2001), Melbourne, Australia.

Mukviboonchai, S., & Chirathamjaree, C. (2001, 22-25 July). *An XML based approach for the integration of database and legacy systems.* Paper presented at the 5th World Multi-Conference on Systemics, Cybernetics, and Informatics (SCI2001), Orlando, FL.

Chirathamjaree, C., & Mukviboonchai, S. (2002, 8-9 August). *A Mediated Data Model for heterogeneous data integration.* Paper presented at the 2nd Annual International Conference on Computer and Information Science (ICIS '02), Seoul, Korea.

Chirathamjaree, C., & Mukviboonchai, S. (2002, 28-31 October 2002). *The Mediated Integration architecture for heterogeneous data integration.* Paper presented at the 17th IEEE Region 10 International Conference on Computers, Communications, Control and Power Engineering (IEEE TENCON'02), Beijing, CHINA.