Durham E-Theses

A Service Late Binding Enabled Solution for Data Integration from Autonomous and Evolving Databases

WANG, CHONG

How to cite:

WANG, CHONG (2010) *A Service Late Binding Enabled Solution for Data Integration from Autonomous and Evolving Databases*, Durham theses, Durham University. Available at Durham E-Theses Online: http://etheses.dur.ac.uk/659/

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a link is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the full Durham E-Theses policy for further details.

Academic Support Office, Durham University, University Office, Old Elvet, Durham DH1 3HP
e-mail: e-theses.admin@dur.ac.uk Tel: +44 0191 334 6107
http://etheses.dur.ac.uk

A Service Late Binding Enabled Solution for Data Integration from Autonomous and Evolving Databases

Name: Chong Wang

Supervisor: Prof. Keith H. Bennett

Ph.D. Thesis

School of Engineering
University of Durham
2010

Abstract

Integrating data from autonomous, distributed and heterogeneous data sources to provide a unified vision is a common demand for many businesses. Since the data sources may evolve frequently to satisfy their own independent business needs, solutions which use hard coded queries to integrate participating databases may cause high maintenance costs when *evolution* occurs. Thus a new solution which can handle database evolution with lower maintenance effort is required.

This thesis presents a new solution: Service Late binding Enabled Data Integration (SLEDI) which is set into a framework modeling the essential processes of the data integration activity. It integrates schematic heterogeneous relational databases with decreased maintenance costs for handling database evolution. An algorithm, named Information Provision Unit Describing (IPUD) is designed to describe each database as a set of Information Provision Units (IPUs). The IPUs are represented as Directed Acyclic Graph (DAG) structured data instead of hard coded queries, and further realized as *data services*. Hence the data integration is achieved through service invocations. Furthermore, a set of processes is defined to handle the database evolution through automatically identifying and modifying the IPUs which are affected by the evolution.

An extensive evaluation based on a *case study* is presented. The result shows that the schematic heterogeneities defined in this thesis can be solved by IPUD except the relation isomorphism discrepancy. Ten out of thirteen types of schematic database evolution can be automatically handled by the evolution handling processes as long as the evolution is represented by the designed data model. The computational costs of the automatic evolution handling show a slow linear growth with the number of participating databases. Other characteristics addressed include SLEDI's scalability, independence of application domain and databases model. The descriptive comparison with other data integration approaches shows that although the *Data as a Service* approach may result in lower performance under some circumstances, it supports better flexibility for integrating data from autonomous and evolving data sources.

Acknowledgements

I would like to express my deep sense of gratitude to everyone who has helped me with my research. In particular, I am heartily thankful to my supervisor, Professor Keith H. Bennett, whose untiring effort, encouragement, guidance and support have helped me greatly in my research. My sincerely thanks and appreciation go to Professor Roger Crouch whose advice and encouragement have been extremely helpful during the final phase of my thesis writing. My thanks also go to all the other people in the department who have discussed my work with me and provided technical support, Bin Weng, Trevor Nancarrow and Michael Wilson in particular.

I have thoroughly enjoyed my time as a Ph.D. student and much of this is due to the people I have shared an office with over the years. My thanks go to Giovanni Airoldi and Graciela Becci for many enjoyable discussions.

I would like to thank my parents, Chending Wang and Xuerong Mei, who have provided endless patience and inspiration with their unconditional love, and supported me in all the ways at all time. I am forever indebted to them.

Finally, my greatest thanks go to my wife Chanjuan Liu. Without her unfailing love, company, encouragement and confidence in me, the ups and downs of research would have been much harder to go through. She has listened to my explanations of my ideas and has always been happy to read my work. The supports she has given me are immeasurable and this thesis is dedicated to her with all my love.

Statement of Copyright

The copyright of this thesis rests with the author. No quotation from it should be published without their prior written consent and information derived from it should be acknowledged.

Declaration

The material presented in this thesis is the sole work of the author and has not been previously submitted for a degree at this or any other university.

Contents

Chapter 1 Introduction

1.1Context

Along with the wide use of database technology and the Internet, it is usual for large business consortiums to require information data from multiple data sources. These data sources may be autonomously designed, supported by heterogeneous database systems, managed by their own applications and situated in distributed locations. Through integrating the data from theses data sources to provide a unified vision, end users can interact with the unified vision directly to only focus on their information needs instead of having to access each data source separately. It can help the businesses run more efficiently hence is crucial for their success. Therefore *data integration* is a common concern that many businesses share.

As the data sources are under autonomous control and may employ heterogeneous database and network technologies to manage their data, data integration can be a complex and difficult task to accomplish hence has been continuously attracting much research attention for decades [55, 37]. Lenzerini refers to the problem of data integration as:
"Combining data residing at different sources, and providing the user with a unified view of these data" [55].

Therefore, data integration needs to solve the heterogeneities of the data stored in the databases of the autonomous and distributed data sources to establish a unified data store for end users to utilize.

1.2 Area of Interest

Despite the large amount of research which has been done, data integration still remains a difficult and costly issue for business [38, 52]. Take the data integration in the Mental Health application domain as an example; the information of the patients may be stored in many autonomous data sources such as hospitals, social services, etc. The data sources may be situated in distributed locations, using heterogeneous databases to manage the data. More importantly, the data sources may change rapidly for various reasons such as changing of the government policy and upgrading of their IT systems. Researches in the IBHIS project [106] has shown that because the participating databases of data integration may evolve constantly to fulfil their own business drive, the evolution may impact the data integration system hence require large amounts of maintenance work. The traditional solutions such as the Federated Database System [76] and Data Warehousing [20, 44] fail to meet the requirements of data integration in a constantly evolving environment. Consequently, a new solution is required for integrating data from the autonomous and evolving data sources.

Although the data sources may employ their own Database Management System (DBMS) to manage their data, all the DBMSs involved in this research are assumed to be Relational Database Management systems (RDBMS) as the RDBMS currently dominates the DBMS market [23]. The RDBMS is based on the relational data model. [22, 23] The relational model is a database model based on first-order predicate logic. It describes a database as a set of *relations* and each relation constitutes a set of *attributes*. Each attribute is described by a name and a data type. Thus each relation can be considered as a predicate variable and the data in the database can be considered as instances of relations (an instance data of a relation is also referred as a *tuple*). The content of the database at any given time can be considered as a finite set of instances of relations. Thereafter in this research, the *database schema* refers to the set of relations of a database. Each data source is assumed to manage its data through a single relational database hence the terms *data source* and *participating database* may be used interchangeably in this thesis.

Since the data sources may be controlled by different owners and designed for different purposes over a period of time, some characteristics of the data sources must be considered as they can form troublesome problems that the data integration has to tackle. Sheth and Larson proposed to define those characters of the data sources along three orthogonal dimensions: distribution, autonomy and heterogeneity [76].

1.2.1 Distribution

Data may be distributed across multiple databases in different data sources, located in different geographical locations. The data sources may be connected to each other by the supporting communication network system. The data from different data sources may relate to each other in different ways. Take the relational database as an example; a set of tuples can be distributed vertically (i.e. different attributes stored in different data sources) or horizontally (i.e. different tuples are stored in different data sources). Since a data integration system provides a unified vision, end users only need to query against the unified vision without being concerned with where the data is stored and the data integration system must be able to manage the distribution of the data. This means that under some circumstances, the data integration system may have to access multiple databases to answer a single user query.

1.2.2 Autonomy

As the data sources may be designed independently without knowing they could be participating in the data integration system, they may be under autonomous control by their own DBMS and applications, making full decisions on their own, such as what data they are willing to share and how others can access those data. They may represent the same information in different ways such as employing different database models (schemas, constrains and query languages); associate with or disassociate from the data integration system at the time they wish; dictate the communication protocols to access their data; decide whether to accept the data access request from other parts of the system and the details of how to process the request. They may also make changes to their database without acknowledging the data integration system.

1.2.3 Heterogeneity

Since the data sources may design their databases independently for different purposes, the participating databases of a data integration system may exhibit a wide range of heterogeneities [42]. As the data managed by the databases are the abstractions used to represent the information in the real world, the design of a database is the process of creating the abstractions and realizing them into computational data through using the database technology. In [07], Batini et al. model the process of database design as four stages:

1. Requirements Specification and Analysis: an analysis to identify the information of various areas in the real world. This results in a preliminary specification of the information that is required to be realized into data.
2. Conceptual Design: modelling and representation of the information from the point view of end users. This results in a conceptual schema that represents a high-level consensus semantic description of the required information.
3. Implementation Design: transforming a conceptual schema into the logical schema of a database model such as the relational model, resulting in a database schema which provides a representation of the required information at schematic level
4. Physical Schema Design and Optimization: mapping the logical database schema into an appropriate stored representation in a DBMS such as a RDBMS. The physical parameters to optimize the database performance based on end user's requirement may be considered during the mapping. This results in a physical database in a specific software and hardware environment to manage the data that represents the required information.

Although the participating data sources design and implement their DBMSs autonomously, it is assumed the information they model are in the same application domain area in the real world. Indeed, if the participating data sources concern different area of the real world and the information they model is not related at all, the data integration may become meaningless. To describe the application domain that all the participating data sources are interested in, the *ontology* is borrowed in this research. Gruber defines the concept of ontology as "An ontology is an explicit specification of a conceptualization." [36]. Thus the role the ontology plays is to provide a high-level consensus semantic description of the salient aspects of the application domain. Since the conceptual schema is also for providing an abstract and simplified view of the application domain [07], it can be considered plays the same role of the ontology from data integration point of view.

When different data sources design their databases, divergence may still arise at any stage from stage two to stage four of the database design process. In stage two, all the data sources intend to model the information of the same application domain of the real world. Different data sources may model the application domain ontology differently from their own point of views hence introducing *semantic heterogeneities* into the application domain ontology. Take the ER model [21] as an example for representing the application domain ontology. A data object may be modelled as an *entity type* in one data source and as an *attribute* of an entity type in another data source.

In stage three, different data sources may represent the same application domain ontology by adopting different logical data models such as the relational data model and the Object-Oriented data model. They may map the domain ontology into different structured database schemas even though they adopt the same logical data model. Thus they introduce *schematic heterogeneities* into the participating databases. Kashyap and Sheth categorized the schematic heterogeneities into a taxonomy [46]. Since all the data sources are assumed to employ the relational data model to realize their databases, the taxonomy is adopted with some modifications in this thesis. Let us assume that in the application domain ontology, there is an entity type (or relationship) E_0 containing a set of attributes (EA_{00}, …, EA_{0n}), with subtypes E_1 and E_2 containing attributes (EA_{10}, …, EA_{1n}) and (EA_{20}, …, EA_{2n}) respectively; two data sources map the application domain ontology independently through their relational database schemas DS_0 and DS_1 which contains a set of relations (R_{00}, …, R_{0n}) and (R_{10}, …, R_{1n}) respectively, and each relation contains a set of attributes (RA_0, …, RA_n) The schematic heterogeneities can be described as the discrepancies between DS_0 and DS_1 at different levels:

- Attribute level discrepancy:

1. Naming discrepancy: EA_{ij} is mapped to RA_i in DS_0 and to RA_j in DS_1 where RA_i and RA_j have different names (Homonyms). Or EA_{ij} is mapped to RA_i in DS_0 and EA_{kl} is mapped to RA_j in DS_1 where RA_i and RA_j have the same names (Synonyms).
2. Attribute domain discrepancy: EA_{ij} is mapped to RA_i in DS_0 and to RA_j in DS_1; DS_0 and DS_1 are said to have Attribute domain discrepancy if any of the following conditions is true:
    1) RA_i and RA_j have different data types
    2) The information instance I_i of EA_{ij} corresponds to the data instance D_i of RA_i and D_j of RA_j, $D_i \neq D_j$ (for example, the mental illness Schizophrenia is represented as a string "schizophrenia" in DS_0 where as a string "schizo" in DS_1)

3. Attribute granularity discrepancy: EA_{ij} is mapped to a single attribute RA_i in DS_0 and a set of attributes (RA_j …, RA_k) in DS_1 (for example, the patient name is represented as a single attribute *Name* in DS_0 where as the attributes *First_Name*, *Middle_Name*, and *Last_Name* in DS_1)

- Relation level discrepancy

1. Naming discrepancy: E_0 is mapped to R_{ij} in DS_0 and to R_{kl} in DS_1 where R_{ij} and R_{kl} have different names (Homonyms). Or E_0 is mapped to R_{ij} in DS_0 and E_1 is mapped to R_{kl} in DS_1 where R_{ij} and R_{kl} have the same names (Synonyms).
2. Relation granularity discrepancy: E_0 is mapped to a set of relation (R_{0i}, …, R_{0j}) with cardinality X in DS_0 and to (R_{1k}, …, R_{1l}), with cardinality Y in DS_1 where $(X > 0) \wedge (Y > 0) \wedge (X \neq Y)$. In other words, the information instances of E_0 can be derived from data instances in DS_0 as well as in DS_1 (for example, all the attributes of a patient are modeled as one relation in DS_0 but as two relations in DS_1)

- Schema level discrepancy

1. Abstraction level discrepancy: In DS_0, E_0 is mapped to R_{ij} with an extra attribute RA_i to indicate each data instance of R_{ij} represents an information instance of either E_1 or E_2. Whereas in DS_1, E_1 and E_2 are mapped to two relations R_{ij} and R_{kl} respectively. In other words, the information instances of E_1 or E_2 can be derived from data instances in DS_0 as well as in DS_1 (for example, DS_0 has a relation "*patient*" which has an attribute "*Category*" to indicate whether an instance is an adult or child patient, whereas DS_1 has two relations for adult and child patient respectively)
2. Schematic discrepancy: schematic discrepancy refers to an attribute and its information instances from the application domain ontology are mapped to relations or attributes or data instances in DS_0 whereas they are mapped differently in DS_1 (information instances can be derived from data instances in DS_0 as well as in DS_1, the examples are provided in section 8.3.1). It could be in any of the following forms:
   1) EA_{ij} is mapped to an attribute RA_i in DS_0 whereas each instance of EA_{ij} is mapped to an attribute RA_j in DS_1
   2) Each instance of EA_{ij} is mapped to an attribute RA_i in DS_0 whereas it is mapped to a relation R_{kl} in DS_1
   3) EA_{ij} is mapped to an attribute RA_i in DS_0 whereas each instance of EA_{ij} is mapped to a relation R_{kl} in DS_1
3. Missing item discrepancy: missing item discrepancy refers to the items in the application domain ontology which are mapped to one database schema but not another. It could be in any of the following forms
   1) EA_{ij} is mapped to RA_i in DS_0 whereas it is not mapped to any attribute in DS_1
   2) E_0 is mapped to R_{ij} in DS_0 whereas it is not mapped to any relation in DS_1
4. Relation isomorphism discrepancy: E_0 is mapped to a set of relations (R_{0i}, …, R_{0j}) in DS_0

where each corresponding data instance of E_0 explicitly identifies itself as a data instance of either E_1 or E_2. Whereas E_0 is mapped to a set of relations (R_{1k}, …, R_{1l}) in DS_1 where it is not specified whether a data instance of E_0 is an instance of E_1 or E_2. (for example, both DS_0 and DS_1 has the "*patient*" relation, the relation in DS_0 has an attribute to indicate whether an instance is an adult or child patient, whereas the relation in DS_1 does not have this attribute)

Due to the autonomy of the data sources, different data sources may choose different DBMSs based on their own preference to manage their databases despite the fact that they are all implementing the same database schema. Even though all the DBMSs involved are RDBMSs, different RDBMS products may require different hardware and software environments and they may support different communication protocols, concurrency control, transaction management and the syntax of query languages. Thus the *system level heterogeneities* might be introduced in the stage four of the database design process.

1.2.4 Existing Types of Approach

For the purpose of facilitating the end users to fulfill their information requirements, data integration needs to combine the data from the participating databases which exhibit the various characteristics introduced above into a unified vision. The vision may be realized through an integrated data store (IDS) and the database schema of the IDS (also referred as *global schema*) is a synthesis of the schemas of all the participating databases. Much research has been devoted to the data integration problems and generated solutions [105, 50, 03]. The current approaches of the solutions to address data integration problems can be broadly divided into two categories: materialized data integration and virtual data integration

1. Materialized Data Integration: this approach transports the actual data from the participating databases into a materialized IDS. The approach is also known as *Data Warehousing* where Widom [95] refers to it as eager approach or in-advance approach.
2. Virtual Data Integration: this approach establishes a virtual IDS store of descriptive information instead of actual data of the participating databases. The information requirements from end users are fulfilled through accessing the actual data in the participating databases based on the descriptive information in the IDS. The approach is also referred to as a 'lazy' or 'on demand' approach.

In materialized data integration, the IDS may be constructed through firstly filtering and extracting data from the participating databases, then resolving the heterogeneities through transforming the data to comply with the global schema and loading the data into the IDS. As the consequence of this, end users can directly pose queries to the IDS and obtain the required results data. Since the data in the IDS is a replication of the data in the participating databases and the IDS is realized as a single database in this approach, end users can appreciate the efficiency of the query processing and obtain the results data even if the participating databases are offline when the queries are being answered. However, this approach requires extra time and spatial cost for maintaining large volumes of redundant data for establishing the IDS. Moreover, changes occurring in the participating database may cause complex maintenance operations in the IDS.

In virtual data integration, it is the descriptive information which is stored in the IDS while the actual data remains residing in the participating databases. The solutions such as the Federated Database System follow this approach. The heterogeneities of the participating databases are solved through constructing the descriptive information to describe the mappings between the global schema and the schemas of the participating databases. As a consequence of this, end users may raise queries against the global schema and the queries are answered through firstly examining the mappings in the IDS to determine which participating databases are relevant for answering the queries, and then accessing them to obtain the required results data. The mappings may be established by following the approaches of Local-as-View (LAV) or Global-as-View (GAV) [55, 37].

Since only the mappings are stored in the IDS, the virtual data integration approach enjoys the flexibility of maintaining modest amounts of data over large number of participating databases. However, as the results data must be obtained from the participating databases, inefficiency may arise in query processing and results data may not be obtained when the participating databases are offline. Changes occurring in the participating database may also cause large amounts of maintenance effort to revise the mappings stored in the IDS when the mappings are realized through hard coded programs.

1.3 Problem Discussion

Although the solutions such as the Federated Database System and Data Warehousing are capable of integrating data from distributed, autonomous and heterogeneous participating databases, substantial amounts of maintenance work may be required when evolution arises in the participating databases; especially for the solutions following the materialized data integration approach. Since the data sources may evolve constantly to fulfill their own business requirements due to autonomy, the evolution has to be taken into account as a characteristic of the participating databases in the data integration.

1.3.1 Evolution

The evolution is a pervasive challenge that every software system has to tackle. Since predicting every possible situation and covering all of them at system design time are simply not feasible, the maintenance operations have to be conducted through the system lifetime and typically take more than 50 percent of its total lifetime cost [61]. As suggested previously, autonomous data sources may evolve constantly to reach their own requirements, thus the evolution a data integration system has to tackle may appear in various forms. For example, new participating databases may join the system and existing participating databases may drop out of the system or go offline. The participating databases may also change their database schemas and what data they are willing to share after they have joined the system. Research in the IBHIS project has identified a set of evolutions a data integration system may encounter; the evolutions are adopted in this research with some modifications. Generally, the evolutions can be categorized as the evolutions at data integration system level and the participating database level, based on which part of the data integration system the evolutions occur.

Evolution at data integration system level:

- End user requirements evolution: the information requirements from end users may change over time, thus the application which is responsible for answering user queries through accessing the IDS may have to be maintained to deal with the evolution.
- Application domain ontology evolution: changes may occur in the application domain leading to the modification of the application domain ontology modelled for the data integration system.

Evolution at data source level:

- Organizational evolution: the participating databases may be organized into a specific organizational structure based on some features of the data sources rather than loosely gathered as an unordered set. Thus the evolution of the data sources may cause alterations in the organizational structure. For example, the location of a hospital may change from Durham to Newcastle due to the county boundary reorganization.
- Schematic evolution: the participating databases may change their database schemas to fulfill the evolving requirements of the data sources after they join the data integration system. The changes should be propagated into the IDS in order to reflect the latest state of the databases. For example, databases may rename, add and/or remove the attributes and relations in their schemas.
- System level evolution: the participating databases may change the DBMSs which accommodate them, the information such as names and URLs to describe them, and the software and network environment they reside at due to the autonomy of the data sources. The changes should be managed in order to keep the correspondence between the IDS and the participating databases.

Even for the data integration solutions which follow the virtual data integration approach, as the mappings in the IDS may be realized through hard coded programs, the evolution may still result in enormous successive maintenance effort in the data integration system as any of the evolution introduced above may cause revising, recompiling and redeploying of the mappings. The maintenance work of the data integration system may become unmanageable or even unachievable once the evolution occurs frequently. Hence a new solution is required to solve the data integration problems while decreasing the maintenance effort caused by the evolution.

1.3.2 Data as a Service

The conception of Software as a Service (SaaS) [52, 13] proposed by the Pennine Research Group provides a possible solution for building software systems in a constantly changing environment. This is achieved through turning software systems into a set of self-described, standard interfaced services and publishing the services on a central registry. The functions of the software systems can be realized through discovering the relevant services and conducting invocations onto them. A truly SaaS based software system can bind to and execute the most appropriate software product service as and when needed (i.e. very late binding). "*At the extreme, the binding which takes place prior to execution, is discarded immediately after execution in order to permit 'system' to evolve for the next point of execution.*" [52]. The main benefit of the SaaS is it can be used to build a highly flexible and agile software system which is able to meet rapidly changing business needs [52, 13]. By allowing the easy substitution of the constituent parts of the system (i.e. the *service*), an 'upgraded new system' is produced to best meet the user requirements every time the evolution occurs.

The IBHIS project [106, 85] conducted by the Pennine Research Group promotes the SaaS into Data as a Service (DaaS) to address the data integration problems by proposing a Service-Oriented data integration architecture (SODIA). It argues that through publishing the participating databases as *Data Services* and employing the *very late binding* technique, the data integration may be achieved by establishing a dynamic unified vision on demand over a set of distributed, autonomous and heterogeneous participating databases which may evolve frequently. The evolution occurring in the data integration system may be handled through modifying or substituting only the data services which are affected by the evolution without involving the rest of the system, hence the maintenance effort may be largely reduced.

1.3.3 Summary

Traditional solutions are not adequate for solving the data integration problems in a constantly evolving environment. This research is targeted at constructing a new service based solution: ***Service Late Binding Enabled Data Integration (SLEDI)*** for integrating data from distributed, autonomous, heterogeneous and frequently evolving participating databases. The participating databases are assumed to be relational databases which realize the same application domain ontology, thus the heterogeneities focused in this research are the *schematic heterogeneities* as described in section 1.2.3. The evolution focused in this research is at the data source level and hence include the *organizational* evolution, *schematic* evolution and the *system level* evolution as described in section 1.3.1.

1.4 Research Outline

1.4.1 The Database Evolution Problem

To meet the needs of integrating data from autonomous and evolving data sources, this thesis addresses the problem of automatic assisted database evolution handling in the context of data integration. More specifically, the question it needs to answer is: *How structured metadata in a service-based data integration model can be used to provide assistance for automatically handling the defined evolutions that occur in the heterogeneous and distributed participating databases?* Since the SLEDI solution is constructed for data integration with automatic evolution handling, it needs:

1. Construction of a unified vision from the data supplied by the data sources, in order to fulfill the information requirements from end users.
2. Providing automatic assistance for handling the evolution occurring at the data source level, in order to decrease the maintenance costs arising from the evolution.

1.4.2 Research Issues

The SLEDI aims to integrate data from participating databases into the IDS by following the DaaS and virtual data integration approaches. Each data source carries out its data supplying through data services and the user information needs are fulfilled through invoking the services. The mappings between the application domain ontology and each data source are established through describing the data supplied by each data source following mainly the GAV approach. Then the mappings are represented as structured data instead of hard code programs and stored in the IDS. As a consequence, the evolution handling may be achieved through employing automatic assistance to conduct the appropriate modifications onto the mappings according to the predefined processes with respect to each specific type of evolution. Therefore, the three major research issues can be identified as following:

1. **Data Source Describing**: integrating the participating databases through describing each data source and the data it supplies with respect to the application domain ontology. The data source descriptions are realized through conjunctive queries and directed acyclic graphs (DAGs) for supporting information needs from end users and automatic assisted evolution handling.
2. **Query Processing**: fulfilling the information needs from end users by answering the user

queries. End users describe their information needs through composing queries against the application domain ontology. And the queries are answered through dynamically discovering the relevant participating databases and obtaining the results data from the databases based on the data source descriptions.

3. **Evolution Handling**: employing automatic assistance to automatically examine the parts of the data source descriptions which are affected by the various types of evolution and modify them accordingly to reduce the overall maintenance effort.

Data source describing involves establishing the mappings between the application domain ontology and the participating databases and constructing the data services. Query processing includes discovering the relevant data services and invoking the services to obtain results data. Evolution handling consists of analyzing the mappings and modifying them accordingly to maintain the validity of the mappings.

1.4.3 Problem Boundaries

The solution for integrating data from autonomous and evolving data sources presented in this thesis has been developed with the assumption of certain problem boundaries.

- The dominating position of the RDBMS in the DBMS market provides a strong motivation for assuming that the data sources employ RDBMS products such as Microsoft SQL Server for managing their data. Thus all the participating databases involved in this research are assumed to be relational databases. Each data source is responsible for determining which data it is willing to share, describing the mappings between its databases schema and the application domain ontology and realizing its data supplying activity as data services.
- Each participating database is assumed to actualize the same application domain ontology by its database schema where the database schema conforms to at least the Second Normal Form (2NF) [22]. Thus each relation has a primary key and each data instance of an attribute of the relation is an atomic data value.
- For the purpose of keeping the focus of this research, although the data represents same information instance in the application domain may be stored in multiple participating databases; it is assumed that no data inconsistency and data redundancy problems need to be addressed. Security is certainly a critical problem for any DBMS, but it is beyond the scope of this thesis. It is assumed that all the data in this research are operated in a secure environment with no authentication and authorization problems.
- The addressed evolutions are those only occurring at the data source level; the application domain ontology is assumed to have no changes.
- The user queries are assumed to be read only queries and are answered one at a time. The query optimization in the query processing, the database concurrency and transaction

control of the participating databases are not considered in this thesis.

1.4.4 Summary

In summary, this thesis is targeted at constructing a solution (i.e. the SLEDI) to integrate data from participating heterogeneous relational databases which are evolving frequently by following DaaS and virtual data integration approaches. The solution solves the database heterogeneities and handles the database evolution with automatic assistance to provide a unified vision to fulfill the information needs from end users through answering user queries. The heterogeneities focused on are the schematic heterogeneities and the evolution focused on is the organizational, schematic and system level evolution.

The solution is capable of solving most of the schematic heterogeneities introduced in section 1.2.3 although the participating databases may not be able to supply results data for answering user queries under some circumstances. For example, for the missing item discrepancy, a participating database may not provide results data if its database schema does not model the entity types, relationships or attributes involved in the user queries. And for the relation isomorphism discrepancy, a participating database may fail to contribute if it cannot precisely describe the instance of which entity type it provides (for example, a database cannot contribute to the query asking for child patients information if it does not distinguish the adult and child patients in its schema). The evolution occurring in the application domain ontology cannot be automatically handled as it may cause substantial modification even a discarding of the data source descriptions. The organizational, schematic and system level evolution are covered although user queries involve multiple versions of organizational structure cannot be directly answered as the late binding can only find the relevant data sources based on their latest states. However, further research may be conducted to address the issues.

1.5 Research Aims and Criteria for Success

The aims of the research presented in this thesis, characterized by the criteria for success, cover many aspects of both the problem and solution. A framework to model the data integration is required to allow the easy comparison between the solution constructed in this research and other related works. The problem of integrating data from heterogeneous and frequently evolving participating databases presented previously must be solved. An experimental implementation of the solution is produced to illustrate the viability of the solution. A case study [104, 28] in the Mental Health application domain is then conducted to evaluate the solution. The criteria for success are formally stated below:

1. The definition of a framework for integrating data from the autonomous and evolving data

sources is constructed. The framework should capture the essential processes involved in the data integration activity.

2. The creation of the Service Late-binding Enabled Data Integration (SLEDI) solution which defines the processes and data structures involved in the solution precisely with respect to the framework discussed in criterion 1.
3. The schematic heterogeneities of the participating databases defined in section 1.2.3 can be solved by the Information Provision Unit Describing (IPUD) algorithm created for the data source describing in the solution.
4. The organizational and schematic evolutions introduced in section 1.3.1 can be handled by the evolution handling process constructed in the solution allowing the data integration system to still function properly despite the evolutions which occurred.
5. The DAG structured data source descriptions can support the evolution handling process hence reducing the maintenance effort of the data integration system caused by the evolution.
6. The DaaS approach and service late binding technique can help to mitigate the maintenance costs of the data integration system caused by the evolution introduced in section 1.3.1

The above criteria set out the goals this research is targeted at and are used in chapter 9 as the references for the discussion of the success of the research.

1.6 Evaluation Criteria

The primary objective of this work is to define a data integration solution featuring automatic assisted evolution handling to integrate heterogeneous databases compliant with a simple application domain ontology. It should be capable of being successfully applied to real world applications with the data integration requirements described previously. The experimental implementation and case study are discussed in chapter 7 and 8 which presents an extensive evaluation of the solution constructed in this thesis. In addition to the criteria of success introduced in the previous section, the following criteria are also used in the evaluation:

- Computational costs of evolution handling
- Scalability
- Expandability
- Programming language independency
- Application domain independency

The above criteria cover a wide range of aspects which can help to explore the strengths and weaknesses of the solution in order to determine where the solution can be better applied.

1.7 Contribution

The main contribution of this work is a new solution: Service Late binding Enabled Data Integration (SLEDI) for integrating data from schematic heterogeneous databases where the databases are frequently evolving. It follows the virtual data integration approach and is realized through data as a service and service late binding technique. The participating databases are integrated through describing themselves by following mainly the GAV approach regarding simple application domain ontology. The descriptions are then represented as DAG structured data and realized as metadata maintained in the data service. End users raise queries against the application domain ontology without concerning the participating databases and the queries are answered through invocating the data services. The evolution of the participating databases is handled by automatically conducting modifications to the metadata based on a set of pre-defined processes. The solution is targeted at relational databases and three main research issues within the data integration problem are addressed:

1. **Data source describing**: each participating database of the data sources is described by the IPUD algorithm which organizes the data the database is willing to share into a set of Information Provision Unit (IPU). Each IPU is described through a Global Definition (GD) and a Local Definition (LD) where GD represents what data this IPU provides with respect to the application domain ontology and LD represents how the data can be constructed through a conjunctive query over the database schema. All the IPUs of the database are then embodied into a data service for others to utilize.
2. **Query processing**: the user queries are answered by examining the GDs of the IPUs to find the relevant participating databases that provide the results data through accessing the metadata of the data services. The data services are then invoked to construct the results data only at run time thus hard wired queries can be avoided.
3. **Evolution handling**: a set of processes are defined regarding various types of evolution to identify the IPUs that are affected by the evolution and the corresponding modifications required. Thus automatic assisted evolution handling can be conducted on the data services descriptions based on the processes.

The solution is set in the context of a framework describing the data integration activity. This captures the essential data structure and processes involved in the activity for both virtual and materialized data integration approaches. A formal model of the framework expresses its data structures in set theory, and SLEDI is compared to other related work throughout the thesis. An extensive evaluation of the solution and where the solution may be applied in the real world data integration applications is presented.

1.8 Thesis Structure

The thesis constitutes nine chapters
Chapter 1 introduces the context and motivation of the research, discusses and states the problem the research is focusing on, and sets out the research aims and criteria for success.

Chapter 2 discusses the current approaches for solving the data integration problem in detail. The evolution problem of the data integration, Service-Oriented Computing and its implementation technologies are introduced.

Chapter 3 develops a framework modelling the data integration activity to capture the essential data structures and processes. An overview of the SLEDI solution is presented; the processes and components involved are introduced in general.

Chapter 4 constructs the IPUD algorithm to describe the participating databases into a set of IPUs. The global definitions and local definitions of the IPUs are represented as structured data and the IPUs are accommodated into IPSs. The IPU and IPS are formally presented using set theory and the corresponding constraints are defined.

Chapter 5 introduces the algorithm of answering user queries. The service plan for answering a user query is generated based on the available data services and the query rewriting. The results data of the user queries are constructed through executing the service plans.

Chapter 6 describes the evolution handling algorithm based on a set of processes. Various types of evolution are identified and represented, the automatic assisted evolution handling are illustrated in detail.

Chapter 7 represents an experimental implementation of SLEDI using web service technology. A case study is also presented.

Chapter 8 presents the comprehensive evaluation of the SLEDI based on the case study. The criteria outlined in section 1.5 and 1.6 are used for the evaluation.

Chapter 9 summarizes the work accomplished with a general discussion. The success of the research is considered in terms of the criteria presented in section 1.5 and ideas for further work are suggested

1.9 Summary

Chapter 1 has introduced the work presented in this thesis. The motivation and context of the research have been explained with reference to other related researches in the field. Three major research issues have been identified within the data integration problem: data source describing, query processing, and evolution handling. Evaluation criteria have been presented and the structure of the thesis has been outlined.

Chapter 2 discusses the background of the current approaches for solving the data integration problem in detail. The evolution problem of the data integration, Service-Oriented Computing and relevant implementation technologies are introduced.

Chapter 2 Background

2.1 Introduction

Chapter 1 presented the brief overview of the context and motivation of this work. The research problem was formally defined and three key research issues have been identified. Criteria for evaluating both the solution and the research were presented. The structure of the thesis was also outlined.

This chapter examines the background of the SLEDI solution presented in this thesis. The current issues of the data integration problems and the software evolution problems are further investigated. The approaches and technologies for solving the data integration problems together with the research methods and application domain are discussed in detail to provide the context of this research.

2.2 Data Integration

Cost effective operation is critical for all business. As many organizations store their information in distributed databases, it is a common requirement to synthesize the information from these different databases within the organization to provide a unified view to support the executive operations. Take the Mental Health domain as an example; information of patients is held by many different entities (primary care practices, hospitals, mental health trust and community health services). The end users (e.g. doctors) need to have a unified view of the data from all the participating entities to have the comprehensive and latest information about the patients in order to provide better services. From a software engineering point view, the need is to integrate data from distributed and autonomous data sources to provide a unified database so that the end users can raise queries against the database.

As introduced in section 1.2, the data integration problem has attracted a great deal of research and the solutions [57, 08, 31] can be broadly categorized as virtual data integration and materialized data integration. The former has been followed by the Federated Database Management System (FDBMS) and the latter leads to Data Warehousing.

2.2.1 Federated Database Management System (FDBMS)

The Federated Database Management System provides a unified access to heterogeneous data from diverse and distributed databases [76, 40]. It is an integrated collection of the component databases. These component databases share their data through adding interfaces allowing others to access them. The FDBMS serves as a middle layer stands between the end users and the component databases and provides transparent access for the end users to access and manipulate the components databases to various degrees. The user can enjoy heterogeneity transparency by translating different data formats from component databases into a common or canonical model (CDM). Other heterogeneities exist in the hardware and operation systems, the data access and manipulation methods and the programming languages of the component databases can also be shield by the FDBMS. The FDBMS can also provide the distribution transparency by masking the distributed nature of the component databases and the network communication details. The general architecture of the FDBMS is illustrated below:

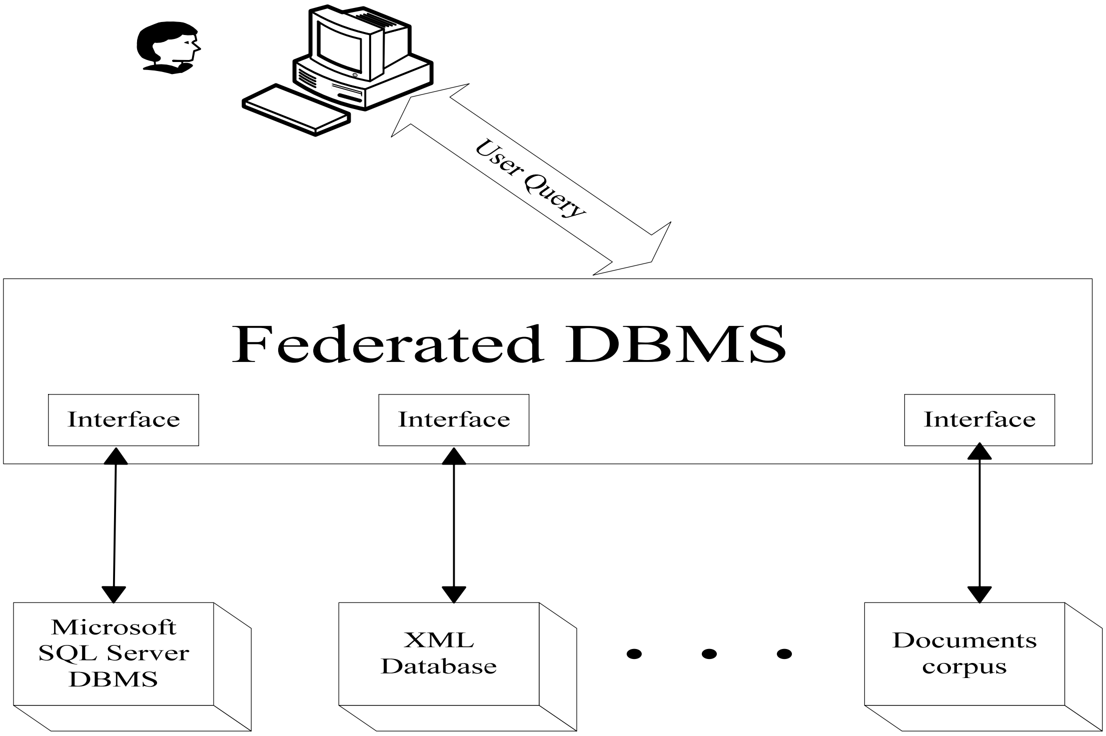


Figure 2-1 the FDBMS architecture

FDBMS can be categorized as loosely coupled and tightly coupled based on who administrates the FDBMS and how the components are integrated.

- Loosely coupled FDBMS: a FDBMS is loosely coupled if the user takes the responsibility of creating and maintaining the federation and there is no central control enforced by the federated system and its administrators. The loosely coupled FDBMS may also be referred to as multidatabase system or interoperable database system [76]

- Tightly coupled FDBMS: a FDBMS is tightly coupled if the federation and its administrators take the responsibility of creating and maintaining the federation, managing the FDBMS by enforcing actively central control on the component databases. A tightly coupled FDBMS is said to have single federation if it only creates and maintains one federation schema.

In both loosely and tightly coupled FDBMS, sharing of any part of a component database or invoking a capability (i.e., an operation) of a component DBS is controlled by the administrator of the component DBS. [40]

FDBMS integrates components and provides a single interface for end users to query. The common process of the user query process involves two steps:

1. User queries over the federation schema are decomposed into smaller fragments called subqueries based on the information of the component databases in the federation. The subqueries are then sent to corresponding component databases to obtain the results.
2. The FDBS communicates with the component databases through their interfaces. The interfaces are responsible for translating the subqueries into local queries which the component database can process directly. An interface may be a simple mechanism which only delivers the sub queries if both the user queries and local queries are based on the same common data model. Or the interface may be complex to do further filtering, transforming and aggregating operations on the data apart from query translating.

One of the key limitations the FDBMS faces is that the users have to explicitly specify the component databases in the queries. Hence if a new component database joins in or existing component database drop out, the user queries have to be explicitly changed. The approach may also suffer lower performance as the user query has to be decomposed and delivered to the components databases to be answered. Furthermore, each component database must build its own interface. As a result, the changes inside the component database such as schematic changes may lead to the interfaces have to be reengineered to keep the DBMS working properly [51].

2.2.2 Data Warehousing

Data Warehousing integrates data from component databases into a single repository called the data warehouse which is available for querying and analysis [20, 44]. Under this approach, the data integration problem is tackled by *Extracting* the data from component databases, *Transferring* the data into a common data model, and *Loading* the data into the warehouse (i.e. the ETL process). Since the model and semantic difference has already been solved during the ETL process and the user queries are processed against the single repository (i.e. data warehouse), this approach enjoys the higher availability and better query performance. The general architecture of Data Warehousing is illustrated as below:

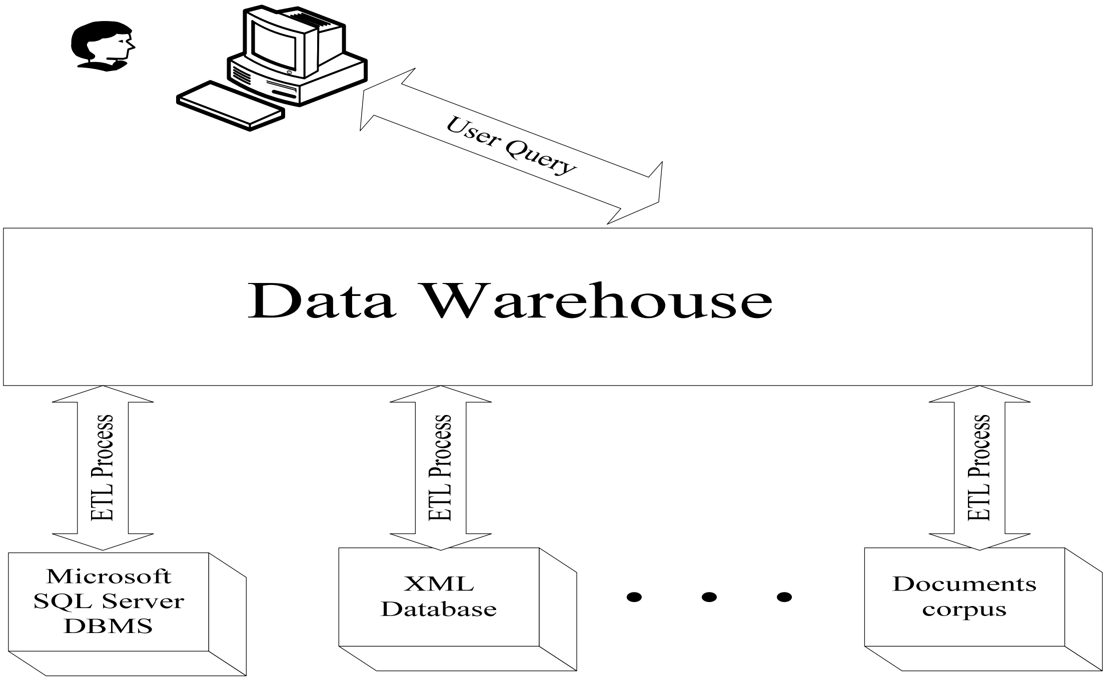


Figure 2-2 the Data Warehousing architecture

The main idea behind this approach is to integrate the data in advance of the query processing. Hence there is no need for query translation and subqueries delivery and executions in remote component databases. The ETL takes care of the processes such as data format converting, information filtering, merging and summarizing. As the component databases might change after the data warehouse was built, the ETL also needs to detect the changes and propagate the changes into the data warehouse.

In the Data Warehousing approach, the component databases always retain their autonomy. As the data in the data warehouse are the replication of the data in the component databases, they can be manipulated asynchronously without affecting each other. One of the key limitations of the Data Warehousing approach is that it has to specify in advance what data should be extracted and integrated from which component databases. After the data warehouse has been built, it is relatively complex to change the original design (e.g. change the specification about what data should be integrated and from which component databases). Furthermore, the physical replication of data introduces potential inconsistencies. The data in the data warehouse may be out of date; hence the changes in the component databases must be explicitly propagated.

2.3 Software Evolution

2.3.1 Software Evolution Process

Software maintenance and evolution activities are pervasive challenges for many software systems. The activities are characterized by their huge cost and slow speed of implementation and their inevitability [54]. Software maintenance is defined in IEEE Standard [43] as:

“The modification of a software product after delivery to correct faults, to improve performance or other attributed, or to adapt the product to a modified environment”

Lientz and Swanson surveyed and categorized maintenance activities into four classes:

- Adaptive: changes in the software environment
- Perfective: new user requirements
- Corrective: fixing errors
- Preventive: prevent problems in the future

The survey showed that around 75% of the maintenance effort was on the first two types and many subsequent studies suggest a similar magnitude of the problem [12, 61]. These studies show that the core problem is addressing the *Environment* and the *User requirement changing*. Bennett and Rajlich suggest these changes should be explicitly addressed and referred to as software evolution and proposed a staged model to better describe the software maintenance and evolution activities in the software system’s lifecycle [12, 11].

The staged model divides the lifecycle of the software system into five stages: Initial development, Evolution, Servicing, Phase-out and Close-down. The software architecture and the knowledge acquired during the development are two main outcomes of the initial development stage. The architecture of the software will persist during its lifetime and the knowledge is a crucial prerequisite for the stage of evolution. Hence the main challenge in the evolution stage is to make the software architectures permit the software system to evolve in controlled ways and raise the abstraction level in the way the evolution is expressed, reasoned about and implemented.

During the service stage, the change of the system component may cause it no longer properly interact with other components. In this case, secondary changes must be made in the neighboring components, which may trigger further changes. This process is referred to as change propagation [101]. The subsequent stages are phase-out and close-down. In the former stage, the software may be still in production, but no servicing is being undertaken. In the later stage, the software used is disconnected and the users are directed towards a replacement.

The evolution handling activities may be conducted through making changes onto the software system during different phases such compile-time and run-time. The changes made at compile-time are also referred to as *static evolution handling* which concerns the source code hence may result in system shut down, recompile and restart. The changes made at run-time are also referred to as *dynamic evolution handling* which normally does not require stopping of the system. The static evolution may cause high cost and the dynamic evolution may require an appropriate level of control over the change to prevent the system crashing or behave erroneously [17, 65].

2.3.2 Data Integration System Evolution

Data Integration System faces the evolution challenge just as other software systems do. As introduced previously, the evolution problems addressed in this research are defined through analyzing the requirements of integrating data from autonomous and evolving relational databases. And the evolution problems can be categorized into organizational, schematic and system level evolutions. The following table provides examples to demonstrate the different types of evolution problems. It is assumed the participating databases are organized into groups and each database schema contains a set of relations.

Evolution Category	Examples
Organizational Evolution	A participating database moved from a virtual group (e.g. Durham) to another virtual group (e.g. Newcastle) A new virtual group adds in or drops out
Schematic Evolution	An attribute changes its name or data type An attribute is split into two or more attributes Two or more attributes are combined into an attribute A relation changes its name A relation adds or removes its attributes A relation is split into two or more relations Two or more relations are combined into a relation A database adds or removes its relations New databases join in or existing databases drop out
System level Evolution	A database changes its name A database changes its description (e.g. URL)

Table 2-1 Database Evolution

When the data integration system is built by following the traditional approaches such as

FDBMS and Data Warehousing, the described evolution may cause static evolution handling onto the system every time the evolution occur. It can be too costly when the amount of participating databases is big and the evolution occurs frequently. Consequently, dynamic evolution handling requires support under these circumstances.

2.3.3 Evolution Support

Generally, FDBMS and Data Warehousing lack the support of dynamic evolution handling. Data Warehousing builds a static centralized repository; hence the evolution handling may be conducted through changing the source code of the ETL process and repopulating the repository. In the tightly coupled federations, since the end users have to explicitly specify which participating databases are involved in the user queries and the federation schema is mostly constructed by manual and static schema integration. The evolution handling may include rebuilding the federation schema and changing the user queries. In the loosely coupled federations, the query translation is normally realized through hard-coded programs based on query based schema matching of the federation schema and the schemas of the participating databases. Thus the evolution handling may include revising the programs which can result in high costs. As a consequence of this, all three solutions for data integration require large amounts of maintenance work for handling the databases evolution introduced above, especially the schematic evolution. Further comparison of the solutions can be found in the next chapter.

2.4 Service Based Concepts and Technologies

2.4.1 Software as a Service and Data as a Service

Although the supply-side approach of developing software works well for the systems with rigid boundaries, it cannot meet the requirement of systems in a constantly changing environment. The new demand-side approach has been proposed to fill the gap: Software as a Service (SaaS) [52, 84]. Under this approach, the system is a service that is composed of smaller ones. The smaller services are only bound at the time of execution and discarded afterwards. Hence the software is procured and paid for on demand, as and when needed.

The IBHIS project promotes the SaaS into Data as a Service (DaaS) through proposing a Service-Oriented data integration architecture (SODIA) [106, 85]. Through publishing the participating databases as *Data Services* that are dynamically determined and bound at the time of execution, a dynamically unified vision can be established on demand over a set of distributed, autonomous and heterogeneous participating databases which may frequently evolve.

2.4.2 Service-Oriented Computing and Architecture

Service-Oriented Computing addresses the relevant research issues of the software application development methods to achieve the SaaS. From the SaaS point of view, a Service is not only a mechanized process; it involves humans managing supplier-consumer relationships [52]. SOC focuses on more technical issues by treating the services as autonomous, platform-independent entities that can be described, published, and discovered; using services which support the development of rapid, low-cost, interoperable, evolvable and massively distributed applications. [79, 70]

Service-Oriented Architecture (SOA) focuses on addressing the requirement of loosely coupled, standards-based, and protocol-independent distributed computing from the software architecture point of view [26]. By using a set of statements that describes software components and assigns the functionality of the system to these components, the SOA describes the technical structure, constraints and characteristics of the components and the interfaces between them. The architecture is the blueprint for the system and therefore the implicit high-level plan for its construction [69, 49]

2.4.3 Grid computing and Web Service

The technologies such as Grid computing and Web Service can be used to develop software following the SOA. Grid computing addresses the research issues of the infrastructure that can synthesize multiple regional software and hardware facilities into a Grid to create a universal source of computing power [29, 72]. It intends to develop and promote the standards and specifications to support a wide range of applications. Some specifications and implementations which focus on the data integration problem in a distributed environment have been developed such as Open Grid Service Architecture Database Access and Integration (OGSA-DAI) and Web Service Data Access and Integration (WS-DAI) [35, 01]

Web Service is developed and promoted by The World Wide Web Consortium (W3C) to help on building SOA based software systems [94, 70]. It is currently the most promising technology that the IT industry as a whole has enthusiastically embraced [25, 90]. Web Service is based on three roles of Service Provider, Service Registry and Service consumer, and consists of a set of standard protocols such as Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description Discovery and Integration (UDDI). The typical scenario of the interactions among the three roles is illustrated in the following figure:

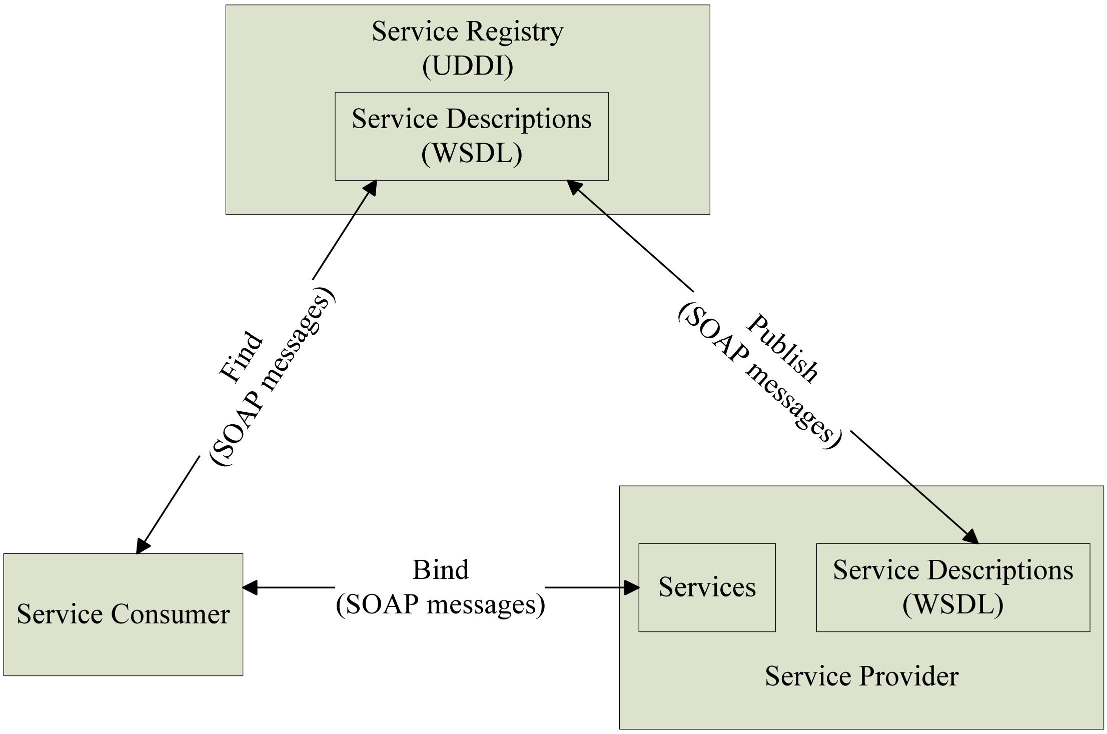


Figure 2-3 Web Service roles and interactions

A service provider hosts a web service and uses WSDL to describe the service, and then publish the service description into the service registry. The service consumer may then access the service registry to discover the service description and invoke the service based on the description. The interactions are realized through SOAP messages exchanging.

The protocols are designed to be independent of any particular programming implementation and to be built on top of the Extensible Markup Language (XML) which has already been largely employed as a basic building block across many application domains. The protocols also accommodate the Hypertext Transfer Protocol (HTTP) to use the current Internet infrastructure for supporting information exchange. These characteristics speed up the software development by reusing the current software and the infrastructure assets with little extra investment.

2.4.4 RDF and SPARQL

The Resource Description Framework (RDF) is a language promoted by W3C for representing resources, particularly the metadata about resources in the World Wide Web [73]. It provides a common framework for identifying things using Uniform Resource Identifiers (URIs) and describing resources in terms of simple properties and property values, thus simple statements about resources can be represented as a *graph* of nodes and arcs. The application designers can leverage the availability of common RDF parsers and processing tools to exchange information between different applications. Consequently, the information created by one application may be made available to other applications to achieve higher interoperability.

Since RDF is in a data form of directed labeled graph, the SPARQL query language [82] is designed and promoted by W3C for querying RDF data. The SPARQL defines syntax for specifying queries as required and optional graph patterns, along with their conjunctions and disjunctions. Hence the queries may be evaluated through traversing the RDF data to find out the sub graphs which match the patterns. Although the properties in RDF can be used to describe the relationships between resources, the RDF itself does not provide mechanism for doing so. The RDF Schema (RDFS) [74] is promoted by W3C as a semantic extension of RDF which defines a set of vocabulary to describe groups of related resources and the relationships between them. Based on the RDFS, richer vocabulary or ontology languages such as OWL [66] can contribute to capture meaningful generalization about data in the Web.

The specifications such as RDF and OWL introduced above can be used to describe web resources and their relationships. The data in other formats, apart from web resources, such as relational data may be transformed into RDF data through middleware. Hence applications can also be developed to exploit the semantics contained in the RDF data.

2.4.5 Service Based Data Integration Solution

Data Integration Solution following the DaaS may achieve higher flexibility. By publishing the participating databases as Data Services, user applications can interact with the databases through the standard service interface without having to know the details of the databases. The evolution of the databases may be handled through changing the corresponding data service without affecting other parts of the data integration system. However, some gaps still exist for the current web service technology to fully achieve dynamic discovery and binding processes in a large-scale, open and ever-changing computational environment [85]. The main challenge is using automated means to accurately discover and bind services with minimal user involvement.

For tackling this challenge in DaaS, the semantics of both the data services and user requests need to be explicitly stated. Although the WSDL explicitly specifies how a service consumer can invoke the service, it does not contain high level semantic information about the service such as what data the service provides. Thus the information needs to be added in addition to the WSDL to support the service discovery and binding at run-time. The UDDI also lacks the support of the additional information. Consequently, the SLEDI solution proposed in this research realizes the additional information as metadata complemented by WSDL to form a meaningful description of the participating databases. The details will be introduced in the following chapters.

2.5 Case Study Method

Research methodology is essential for any research. As a method of conducting academic research, among other empirical methods such as experiments and survey, case studies have been extensively applied and examined in social science research topics [102]. Among the definitions of the case study method from different researchers with different perspectives, Yin has given a technical definition of the method. He suggests that the definition is constituted as a twofold. The first part begins with the scope of a case study [104]:

"*A case study is an empirical inquiry that*

- *investigates a contemporary phenomenon within its real-life context, especially when*
- *the boundaries between phenomenon and context are not clearly evident*".

Yin has pointed out that early works treated the case study not as a formal research method and only used it as complementary to other methods. A common misconception had been established that because the case study method only examines a single example in details in the longitude dimension; it cannot provide reliable information about the broader class thus is only appropriate for the exploratory phase of a research. In fact, the case study method can be used for the purpose of descriptive and explanatory of the research as well. [103]

Flyvbjerg also specified the five common misunderstanding from the early works about the case study method in his research [28]. Including

1. theoretical (context-independent) knowledge is more important than practical (context-dependent) knowledge
2. the results from case studies cannot be generalized to apply to a class of phenomenon as they are based on a single case
3. the case study is appropriate only for the hypotheses generation. Other methods are more suitable for hypotheses testing
4. there is a tendency to confirm preconceived notions of the researchers as the case study contains a bias toward verification
5. it is often difficult to summarize and develop general propositions based on specific case studies

These five propositions were discussed and examined in the research. The conclusion was drawn that all of them are not necessarily true and the disadvantages of the case study method that claimed in the propositions can be avoided.

The case study method can provide a systematic approach for the empirical research by carefully selecting the case, examining the phenomenon, collecting the data, analyzing the information and reporting the results. As a result of applying the case study method, the researchers may gain a sharpened understanding of why the instance of the phenomenon happened in the way as it did, and what might become salient and need to be examined extensively in future research. Consequently, the case study method is suitable to both hypotheses generating and testing [28].

As a research method, the case study can be adopted by the software engineering field from social science. Some pioneering researches for this adoption have been conducted to examine how to apply the case study appropriately in software engineering [48, 47]. Kitchenham, Pfleeger and Pickard [48] suggest that the case study method can help industry evaluate the benefits of methods and tools and provide a cost-effective way of ensuring that employing the methods and tools provides the desired results. Although the case study method may not achieve the scientific rigor of the formal experiments, it can provide sufficient information for judging if specific technologies will benefit the certain organization or project.

The choice of the method depends on the size and nature of the organization or project the study will be conducted on, and whether the technology or tools being studied are in advance or after the fact [48]. The experiments method may be used for comparing several competing technologies or tools in order to conclude which one to choose. The case study method may be employed for assessing a new technology or tools by applying it to a pilot project in order to conclude the effects of it. The survey method may be favoured for examining a technology or tool that has already been implemented across a large number of projects in order to conclude the benefits of using it.

The experiments method requires full control hence is difficult to conduct when the degree of control is limited. Because the experiments method is normally used for small scale research in a laboratory, it can be a problem when trying to increase the scale from the laboratory to a real project. On the other hand, the case study method can avoid the scale up problem. This characteristic makes it particularly useful for industrial evaluation of software engineering methods and tools. Consequently, the case study method is preferable for helping the researchers to understand the improvement that a new technology or tool can bring to a class of objects by assessing the technology or tool through applying them to a pilot project.

2.6 Mental Health Application Domain

The Mental Health application domain provides a perfect example of the research of integrating data from autonomous and evolving data sources as it demonstrates the characteristics of the research by its natural requirements [106, 33]. In the mental health domain, the patient information may be stored in many autonomously managed data sources such as local surgeries, hospitals, social services, etc. Each data source may realize the patient information in different database schemas and employ different DBMS to manage the databases in their own IT systems as shown in the following figure:

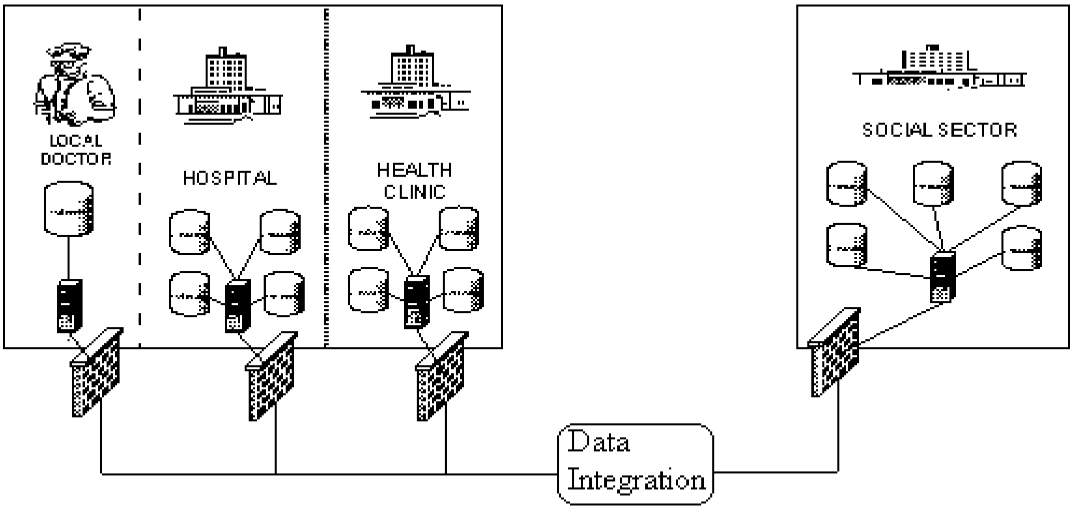


Figure 2-4 Data integration in the mental health application domain

Integrating the data from the data sources is a common requirement in the domain. For example, patient records may be held by different data sources when the patient has been living in different locations and treated by different hospitals and social services. Thus it is required to integrate the data from the data sources to construct a comprehensive view of the patient information in order to provide a better service for the patient. The mental health researchers may also require data integration based on their own criteria to do their research such as the epidemiology, the effect of a specific medicine or treatment, etc.

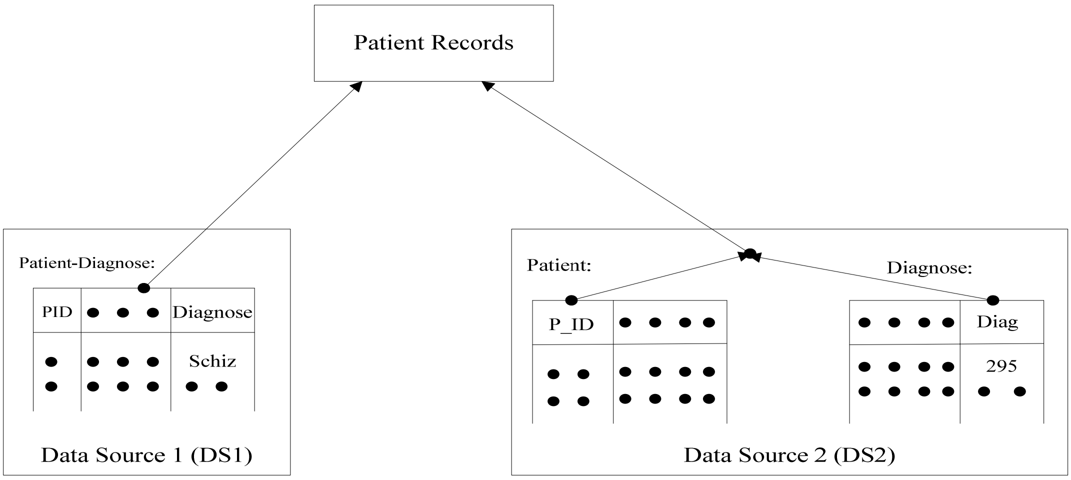


Figure 2-5 Data integration example

The databases in the data sources may present various heterogeneities as they are autonomously managed. As the example illustrated in figure 2-5, the patient record is realized as one relation in DS1 and as two relations in DS2. Diagnosis of patient in DS1 is represented as simple text and as integer code in DS2. Furthermore, the autonomous nature of the data sources may result in continuing evolution of the participating databases and data sources may join in and drop out at their convenience. Thus the database schemas may evolve continuingly in various ways as introduced in section 2.3.2. The DBMS, software and hardware environment of the databases may also evolve due to upgrading of the IT system of the data sources. Consequently, the data integration solution should be able to solve the heterogeneity and evolution problems to provide a unified view hence end users can focus only on their own information needs.

2.7 Summary

This chapter has introduced the current approaches for solving the data integration problems. The issues of the software evolution and data integration system evolution are then presented. The service based concepts and technologies, the case study research method and the mental health application domain are also introduced. This background provides the clear view of the setting of the research presented in this thesis.

Chapter 3 presents a descriptive framework to model the activity and representation of the process of integrating data from autonomous and evolving data sources. The framework provides the context for the novel solution (i.e. SLEDI) created in this research which will be described through the next chapters. The services and processes of the SLEDI are also briefly discussed.

Chapter 3 Data Integration Framework

3.1 Introduction

Chapter 2 introduced the background of current solutions for solving data integration problems. The evolution problems the data integration may encounter were presented. The case study method of research and the mental health application domain were briefly discussed.

This chapter introduces a descriptive framework to characterize the data integration activity from the abstract level, hence the virtual and materialized approaches can both fit into it. The new solution for integrating data from autonomous and evolving data sources, the Service-Late Binding enabled Data Integration (SLEDI), is presented. The constituted services and the processes of the SLEDI are described and the characteristics of the solution are briefly discussed with respect to the framework.

3.2 Data Integration Activity Framework

As discussed in section 1.3, in order to combine data from multiple data sources to provide an information provision service to end users, the activity of data integration is required. This section presents a descriptive framework for the data integration activity. It specializes in integrating data from distributed, autonomous and evolving data sources. The framework unifies the virtual and materialized approaches and is used later in this thesis for a discussion of the cost saving which might be achieved when using automatic assistance for handling the evolution occurred in data sources.

Initially, it is assumed there are a set of distributed data sources. The data sources manage their data autonomously and are constant evolving. The data integration activity intends to combine the data from the data sources into a single centralized database (i.e. the integrated data store IDS) and provides an information supplying service to end users. This is shown in figure 3-1

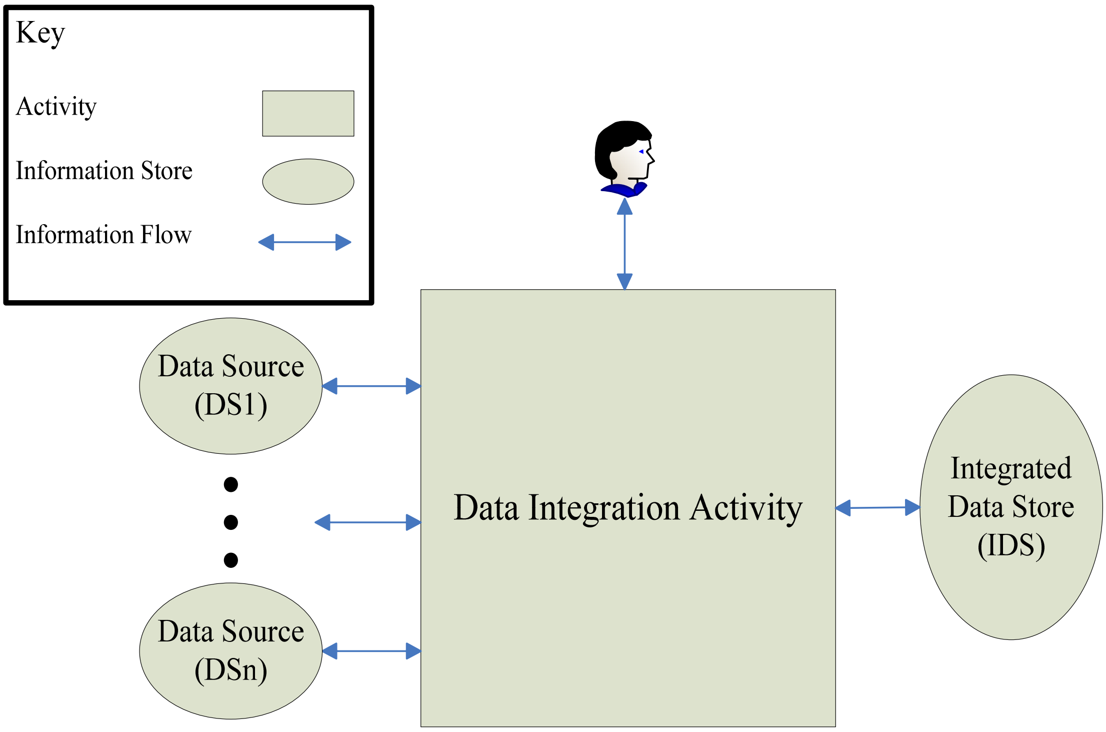


Figure 3-1 Basic Framework describing the data integration activity

The data integration activity can be characterized as the creation, access, and maintenance of the IDS, which is achieved by the processes: *data integrating, information supplying*, and *evolution handling* respectively. The three processes works together to fulfill the requirement of the data integration. This is shown in figure 3-2

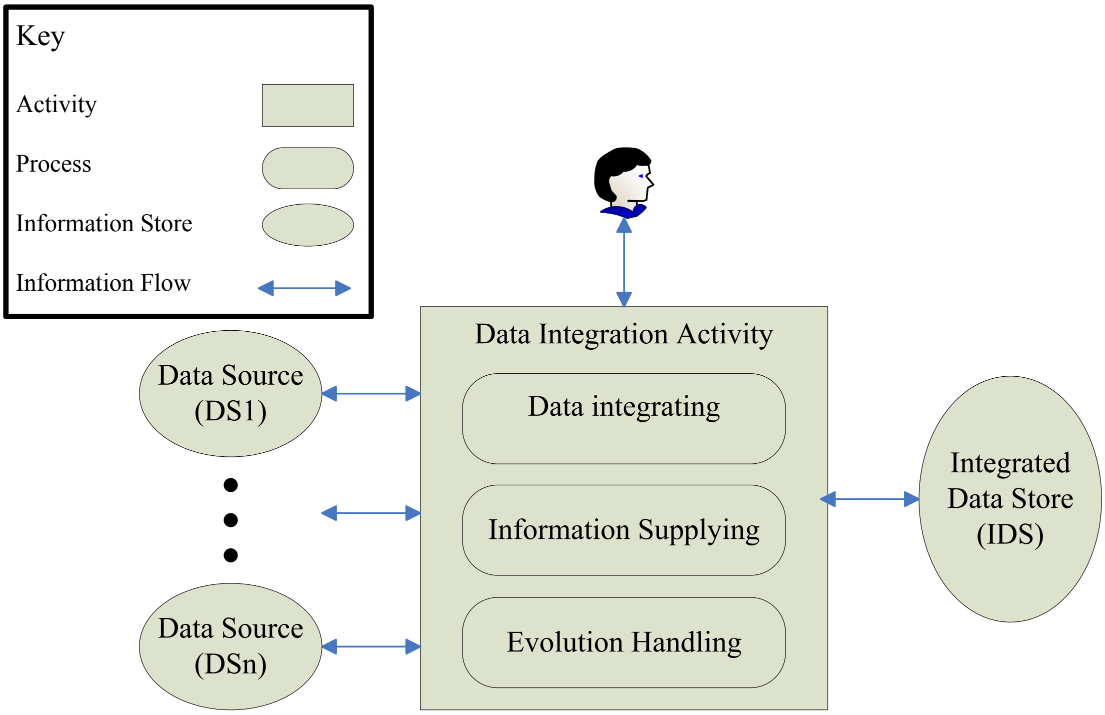


Figure 3-2 The Framework of data integration activity

1. Data integrating: the role of the data integrating process is to select the data from the data sources, harmonize the heterogeneities among those data and combine the data together to create the unified data store (i.e. IDS). It is the preliminary step in the data integration activity and builds up the IDS hence other processes can be conducted based on the IDS
2. Information supplying: the role of this process is to communicate to the end users to fulfill their information needs. It takes the description of the information needs from end users (e.g. queries raised by end users); processes it and conducts appropriate operations against the IDS; it then constructs the results and sends the results back to the end users (e.g. results for answering the queries).
3. Evolution handling: this process is triggered when evolution in data sources occur. It investigates the evolution occurring and take actions against the IDS accordingly. Hence maintaining the coherence between the IDS and data sources. As a consequence, the end users' information needs can be processed based on the latest state of the data sources.

The framework in Figure 3-2 captures the general requirements of a data integration activity. Since the framework models the requirements from a highly abstract level, both the materialized and virtual approaches for conducting the data integration activity can be fitted into the framework. Hence the comparison can be made between different approaches and the strength and weakness of the approaches and the cost related issues can be discussed within the framework

In the materialized approach, the IDS is a materialized data store. The data integrating process is carried out by first selecting the data from data sources and harmonizing the heterogeneities. Then the integrated data is transported into the IDS. Although some addition and alteration may be made to the data during this process, the data in the IDS are mainly replications of the data in the data sources. The information supplying process examines the descriptions of the information required from end users and meets the requirements by directly manipulating the actual data in the IDS. When evolution occurs in the data sources, the evolution handling process analyses the evolution and takes appropriate actions against the actual data in the IDS in order to propagate the evolution from data sources into the IDS.

In the virtual approach, the IDS is a virtual data store, which means instead of the actual data, it is the descriptive information about the data which is contained in the IDS. This information represents how the data in the IDS can be formed from the data sources and normally is realized by formally specifying the mappings between the actual data in the data sources and the IDS (e.g. queries over the data sources). The data integrating process is performed by investigating the data sources, and then creating the formal declarations to describe the mappings. The information supplying process first evaluates the information requirements and then interrogates the descriptive information in the IDS to generate the operation plan. Finally it executes the plan against the data sources to accomplish the requirements. The evolution handling process analyses the evolution occurring; and then reviews the descriptive information in the IDS and makes appropriate modification to it in order to maintain the coherence between the IDS and the data sources.

3.3 The Service Late-binding Enabled Data Integration Solution

As discussed in chapter 1.3, the focus of the work in this thesis is integrating data from autonomous and evolving data sources. All the data sources involved employ relational DBMS to manage their data and the evolution addressed are the schematic, organizational and system level evolution occurring in the data sources. This thesis presents a new solution: the Service Late-Binding Enabled Data Integration (SLEDI) which takes the virtual approach and employs the concept of Software as a Service [52] and Web Service technology [94], using Late Binding techniques [13, 106] to achieve the required data integration.

To meet the design purpose, the SLEDI must not only have the ability to integrate the data from the data sources to provide the information supplying service to end users, but also have the potential to decrease the maintenance costs while the data sources are evolving. Conceptually, the SLDEI constitutes four types of services: Information Provision Service (IPS), Broker Service (BS), Evolution Handling Service (EHS) and Registry Service (RS). In practice, each type of service might be implemented by multiple instances. Based on the services, the SLEDI accomplishes the data integration activity through three processes: Data Source Describing, Query Processing and Evolution Handling. Each process may involve one or more type of services to achieve its purpose. The general architecture of the Services is shown in Figure 3-3

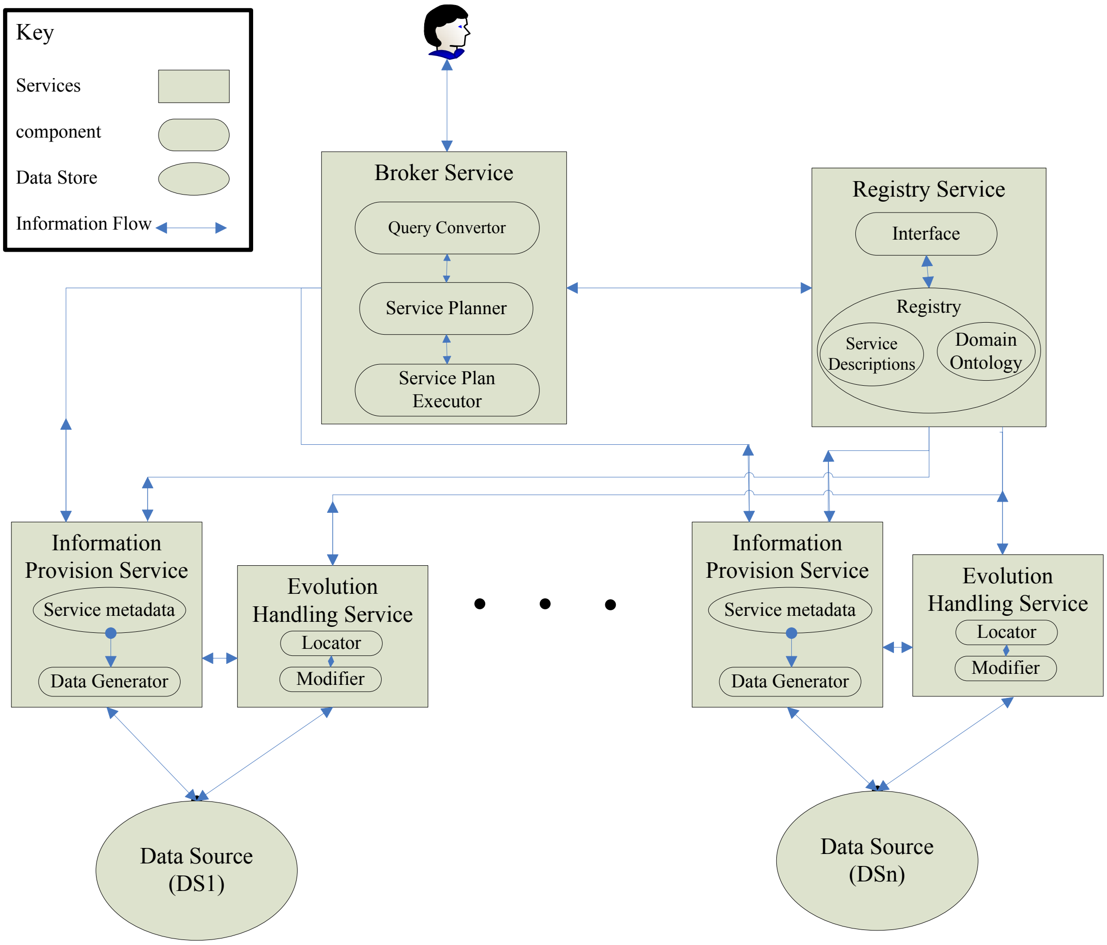


Figure 3-3 the General architecture of the SLEDI

3.3.1 Overview of the Services

1. Information Provision Service: Each data source shares its data through the Information Provision Unit (IPU) mechanism. An IPS aggregates all the IPUs of a data source and reveals its service description metadata which includes the *Global Definition* of the IPUs to describe what information the IPS provides. It accepts invocation calls from the Broker Service. Finally, it generates results by consulting the *Local Definition* of the IPUs and accessing the actual data to respond the invocation calls.
2. Broker Service: this Service accepts the information requirement descriptions (e.g. queries) from end users. It analyses the descriptions and interacts with the Registry Service to discover the appropriate IPSs which provide the information required. Then, it filters out these IPSs and produces a service execution plan based on them. It then executes the plan by invoking the IPSs and collects and combines the results of the service invocations into a final result. Finally, it sends the results back to the end users.
3. Evolution Handling Service: this Service accepts the descriptions of the evolution occurring

in the data sources. It analyses the descriptions to locate the IPSs which are affected by the evolution. It then modifies the IPSs accordingly to reflect the evolution, hence maintaining the virtual IDS to be consistent with the latest state of the data sources.

4. Registry Service: this Service provides a central repository as a registry that all the IPSs, EHS and BS can be published onto. The service publication is realized by storing the description of the services into the repository and the Registry Service provides an interface allowing the services to publish, search, retrieve and update the services descriptions in the registry. Hence the services can be discovered by accessing the repository and invoked by parsing the service descriptions.

As the research focus is integrating data from autonomous and evolving data sources, the SLEDI is designed to provide a solution for building data integration systems with flexibility. The data sources can connect into and disconnect from the data integration system by simply adding or removing the IPSs. The descriptions of IPSs provide an interface for data sources to specify what data they provide and how the data can be formed. As the description is realized in structured metadata instead of a hard wired programme, the evolution handling can be achieved through employing automatic assistances to modify the metadata without human interventions thus decreasing the maintenance costs of the system. The interoperability the Web Service provides further enhances the flexibility of the SLEDI. Moreover, the EHS and BS may have multiple instances in the implementation of the SLEDI to achieve a better performance.

3.3.2 Overview of the Processes

As a solution for data integration, the SLEDI can also be fitted into the framework which has been introduced in section 3.2. The processes of data integrating, information supplying and evolution handling of the framework are carried out in the SLEDI by the processes of *data source describing*, *query processing*, and *evolution handling* respectively.

1. Data source describing: Each data source first determines the data in its databases it is willing to share. Then the data is organized into a set of Information Provision Units (IPU). An algorithm named Information Provision Unit Describing (IPUD) is created for producing the IPUs. Each IPU is described by a *Global Definition* and a *Local Definition*. The global definition is specified by an expression over the application domain ontology and the local definition is represented by an expression over local database schema. The global definition provides a description which can be understood across the application domain hence insulating the schematic heterogeneities among the data sources. All the IPUs provided by a data source are then aggregated into an IPS and the definitions of the IPUs are realized as service description metadata of the IPS. Then data sources publish the IPSs through the Registry Service for other services to discover and invoke.
2. Query processing: end users describe their information requirements through raising queries

over the application domain ontology. The queries are then processed by the Broker Service and each query is transformed into a service plan; the Broker Service then executes the plan by sending invocation calls to the appropriate IPSs. Each IPS responds to the invocation by generating results from local database and sending them back. Finally, the Broker Service composes the results into the final answer to fulfil the requirements.

3. Evolution handling: every evolution occurring in the data sources is described and sent to the Evolution Handling Service. This analyses the evolution descriptions, interacts with the Registry Service and the IPSs to identify the IPSs which are affected by the evolution. It then examines and automatically modifies the affected IPSs accordingly. Under some circumstance, the modification may be achieved with human interventions.

The three processes work together to establish the data integration system which is realized by the four types of services. Hence the required data integration activity can be accomplished to fulfill the information needs from end users.

3.4 The processes of the SLEDI

This section illustrates the constituted processes of the SLEDI in more detail. Each process is delineated by describing the requirements it intends to fulfill, the detailed processes of which it consists, the algorithms and operations involved and its realization over the services.

3.4.1 Data Source Describing

In this research, all the databases involved in the data sources are assumed to be relational databases. As introduced in chapter 1, all the data sources in this research are in the same application domain. A canonical data model (also referred to as a domain ontology model) is employed to characterize the salient parts of the application domain. The model is used as the virtual schema of the IDS hence end users can describe their information requirements by raising queries against it. The data source describing process intends to build the virtual IDS by establishing the descriptions of the data sources with respect to the ontology. It is composed of the following steps

1. Constructing the domain ontology: a Description Logic based data model is created to model the application domain ontology. It provides an abstract description of the application domain by modelling it through concepts, roles and rules. As the data in the data sources are also representing the information in the application domain, they can be described by aligning to the application domain ontology. This step is a preliminary step which is supposed to be accomplished before all other steps and the domain ontology is stored into the registry for other components to access.
2. Establishing the IPUs: Due to the autonomy, the data sources may only agree to share some

parts of their data. An IPU is an information unit which can be described unambiguously by an expression over the domain ontology. By employing the IPU, each data source can share its data through a set of IPUs.

3. Building the definitions of IPUs: the definitions of each IPU consists two parts: the *Global Definition* and the *Local definition*. The global definition is in the form of an expression over the application domain ontology which semantically describes the information the IPU provides, hence can be understood by other components across the application domain. The local definition is in the form of a conjunctive query over the local database schema which specifies how the data of the IPU is formed from the data sources.
4. Constructing the IPSs: All the IPUs a data source provides are aggregated into an IPS. The global and local definitions of the IPU are converted into structured metadata. A service metadata model is designed to accommodate the metadata together with other descriptive information to describe the IPS.
5. Publishing the IPSs: As described in chapter 1, there is an organizational structure among the data sources; a set of DAGs are employed to represent the organizational structure. Then all the IPSs of the participating data sources are published into the registry together with the DAGs. Hence other components can discover and access the IPSs.

The data models, algorithms and the detailed processes of the data source describing are further described in the next chapter.

3.4.2 Query Processing

After the processes of data source describing for all the participating data sources are accomplished, the IDS is successfully built. Then the description of information requirements from end users (e.g. queries) can be processed. In this research, each user query consists of two parts: the *targeting data source set* and the *query content*. The targeting data source set specifies the set of data sources the query intends to query against. And the query content describes the information requirements by an expression over the domain ontology. In this research, the expression is in the form of a *union of conjunctive queries* (the definition of the logical form *conjunctive query* can be found in [88]). Each conjunctive query constitutes the concepts, roles and rules from the application domain ontology. The process of query processing for a user query consists of the following steps:

1. Targeting data sources filtering: the targeting data source set specified in the user query is processed to identify the targeting data sources. Then the IPSs of the targeting data sources are filtered out into a list.
2. Query rewriting: the query content of the user query is rewritten into a set of subqueries. Each subquery is in the form of conjunctive query so that each of its atomic conjunct refers to the global definition of an IPU from the IPSs. The *query rewriting* technique [09, 34] is employed to generate the maximum contained rewritings to answer the user query.

3. Service plan generating: examines each of the subqueries, identifies the IPUs whose global definition is in the subquery and then generates the service invocation calls to the corresponding IPSs from the IPUs. Then constructs the service plan of the subquery based on the service invocation calls.
4. Service plan executing: executes the service plan by invoking the IPSs. Each IPS responds to the invocation call by firstly examining the local definition of the IPU specified in the invocation call, then constructing the local queries against the local database schema. Finally it executes the local queries to construct the results data for the invocation call from the data source.
5. Result constructing: collects the results data of the service invocation calls of every subquery, combining the results data to form the answer to the subquery. As a consequence, the answer for the user query can be finally constructed by integrating the results data from each of the subqueries.

The above steps are accomplished by cooperation among the Broker Service, IPSs and the Registry Service. The details of query processing are further discussed in chapter 5

3.4.3 Evolution Handling

As one of the central issues in this research, evolution handling is designed to deal with the evolution occurring in the data sources to decrease the maintenance costs. By describing the data sources through the IPU mechanism, the impact the evolution have on the IDS can be reflected through the IPUs and, more specifically, on the definitions of the IPUs. Because these definitions are realized through the structured metadata, automatic assistance can be employed to traverse and modify the metadata accordingly to handle the evolution. The evolution handling process is composed of following steps:

1. Identifying the evolution: The evolution covered in this research are the schematic, organizational and system level evolution as discussed in chapter one. Evolution data models are created to accommodate the detailed data for describing each type of the evolution. Thus the evolution can be identified and further actions can be determined based on the detailed data.
2. Constructing the modification processes: establishing the specification to describe the processes of modifying the local definitions of the IPUs with respect to each distinct type of evolution.
3. Identifying the affected IPUs: investigating the descriptions of the evolution occurring. Identifying the IPUs which have been affected by the evolution based on the evolution description and the local definition of the IPUs. Then the affected IPUs are filtered out into a list.
4. Modifying the affected IPUs: analysing the evolution descriptions and the definitions of the

IPUs in the list. Modifying the IPUs based on the processes defined in step 2.

The above steps are accomplished by cooperation among the Evolution Handling Service, IPSs and the Registry Service and the details of the evolution handling are further discussed in chapter 6

3.5 The services of the SLEDI

As introduced in section 3.3, the SLEDI is based on the Web Service technology and late binding techniques and is realized by four types of Service. This section explains the mechanism and the working process of each type of service.

3.5.1 Information Provision Service

The IPS in the SLEDI has got two roles to play. The first role of the IPS is to expose the global definition of its IPUs to declare what information the IPS supplies. Thus other services can discover the IPSs which fulfill their needs. The second role of the IPS is to generate the data which represent the information it supplies from the data source it belongs to. It accepts the invocation calls from and sends the results data back to the Broker Service. In this research, each data source is assumed using a single relational database which is represented by a relational schema (i.e. local database schema) and provides exactly one IPS. The IPS contains all the IPUs the data source provides; it accepts invocation calls from the Broker Service and the Evolution Handling Service and does not invoke other services. The IPS consists of the following components:

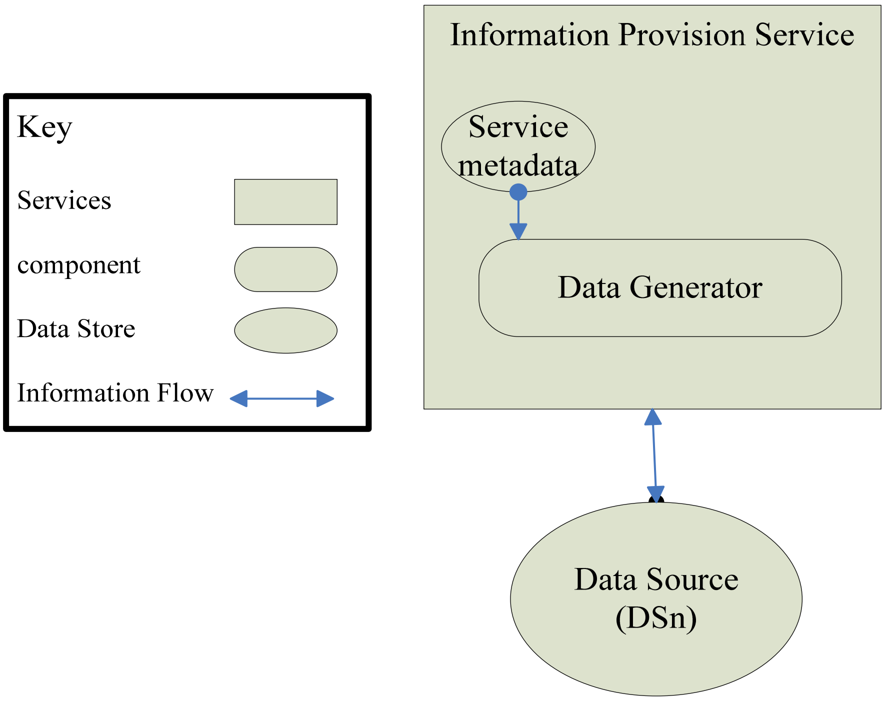


Figure 3-4 Information Provision Service

1. Service metadata: the service metadata accommodates the description of IPS which includes its name, address, exposed methods with parameters, global definition and local definition of the IPUs. The metadata apart from the local definition of the IPUs are exposed through publishing onto the Registry Service for other services to access.
2. Data generator: at each time the IPS is invoked by the Broker Service, the data generator takes the parameters from the Broker Service as the input. It then extracts the IPU specified in the parameters and retrieves the local definition of the IPU from the service metadata. Next, it composes the local queries and executes the queries against the data source to generate the results data. Finally the IPS sends the results data back to the Broker Service.

Each data source accommodates the IPS it provides at its own site and publishes the descriptions of the IPS into the registry.

3.5.2 Broker Service

The role the Broker Service plays is to communicate with end users, accepts the information requirements descriptions from end users and delivers the results back. It is composed of the following components.

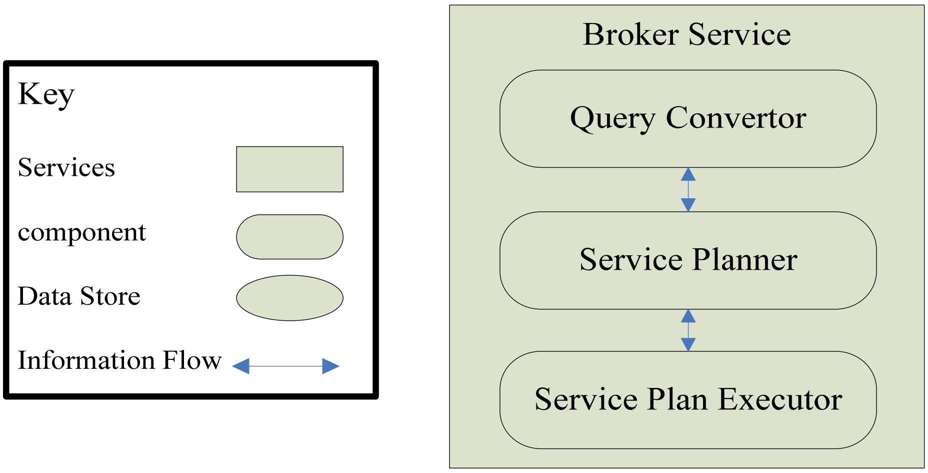


Figure 3-5 Broker Service

1. Query convertor: the query convertor consists of a series of operations which take one query at a time from end users. The query is in the form of an extended conjunctive query over the application domain ontology. The query convertor firstly analyses the query and acquires the organizational structure information through accessing the Registry Service to construct the targeting IPS set. Then it rewrites the query into a set of subqueries based on the global definition of the IPUs contained in the IPS set through employing the query rewriting technique. Subsequently, it outputs the set of subqueries for further processing.
2. Service planner: the service planner takes the set of subqueries produced by the query convertor as the input and processes one subquery at a time. Since each atomic conjunct of a

subquery refers to the global definition of an IPU. Each subquery is then segmented and converted into a list of service invocation calls. Once all the subqueries are processed, the service planner collects all the lists of service invocation calls to construct a service plan.

3. Service plan executor: the service plan executor takes the service plan as the input and processes one list at a time. For each list, it invokes each of the service invocation calls in the list sequentially by sending the call to the corresponding IPS and acquiring the results data. Once all the service invocation calls in a list are invoked, the results data of the list can be composed. After all the lists are processed, the final results data can be produced and sent back to the end user to answer the user query.

The Broker Service is the only type of service in the SLEDI which communicates with the end users. It accomplishes the task of fulfilling the user information requirements through interacting with the IPSs and the Registry Service. In practice, another layer might be appended on top of the Broker Service in order to employ applications to provide a more user friendly interface for the end user to interact with. The Broker Service might also be realized by multiple instances to distribute the workloads of information supplied in order to achieve a better performance. Although the Broker Service is only invoked by end users in SLEDI, all its instances are required to register through the Registry Service. Hence other applications or systems may discover and interact with it to achieve higher flexibility.

3.5.3 Evolution handling service

The role the evolution handling service plays is to tackle the evolution occurring in the data sources to mitigate the maintenance costs of the system caused by the evolution. Initially, it is assumed at every time the evolution occurring, the data source takes the responsibility to identify the evolution and invoke the EHS to propagate the evolution into the IDS. The evolution description is passed to the EHS through the invocation call. The EHS responds to the invocation call by processing the evolution descriptions; it then locates the IPS whose IPUs are affected by the evolution. Then it modifies the service metadata of the IPS. The modification process might involve invocations to the Registry Service to be accomplished. The EHS consist of the following components:

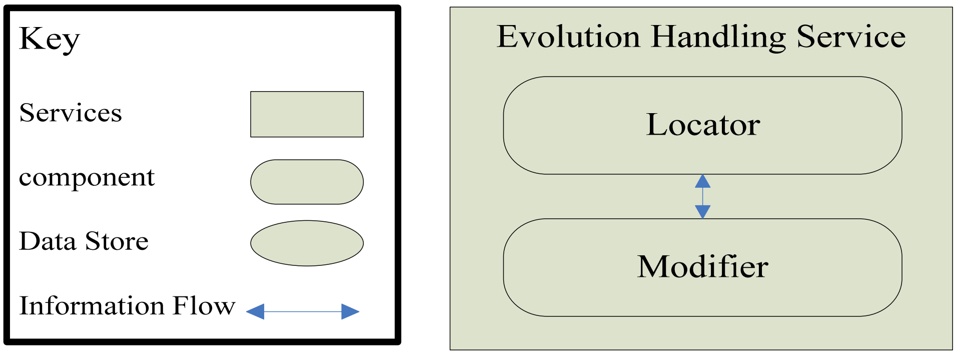


Figure 3-6 Evolution Handling Service

1. Locator: the locator accepts one evolution description at a time as the input. For each evolution description, the locator detects the evolved local database specified in the evolution description to find the IPS the local database (i.e. data source) provides. Then for each IPU contained in the service metadata of the IPS, the locator analyses its local definition against the evolution description to determine whether the IPU is affected by the evolution. Subsequently, for each evolution description, the locator generates a list containing all the affected IPUs as the output.
2. Modifier: the modifier takes the output of the locator as its input. For each IPU in the list, by parsing the evolution description and consulting the predefined modification processes, the service metadata of the IPS corresponding to the IPUs are modified accordingly. The modifier then determines whether further modification onto the registry is required. If this is the case, the modifier invokes the Registry Service to conduct the modification.

Conceptually, only one EHS is required in the IDS, it accepts the evolution description from the participating data sources and applies the modifications onto the service metadata of corresponding IPSs. In practice, the evolved local data source which invokes the EHS also accommodates the IPS that may require modification. Thus each data source may implement an instance of EHS to enhance the efficiency of the evolution handling.

3.5.4 Registry Service

The Registry Service contains a data repository (i.e. registry) which consists of services descriptions and the domain ontology. An interface is exposed to interact with other services. It accepts invocation calls, accesses the registry and sends the results back to other services. Through the interface, all the services can publish and maintain their descriptions into the registry. Based on the Web Service technology, the descriptions are realized in a standard and neutral way to achieve better interoperability across heterogeneous environments. The Registry Service consists of the following components:

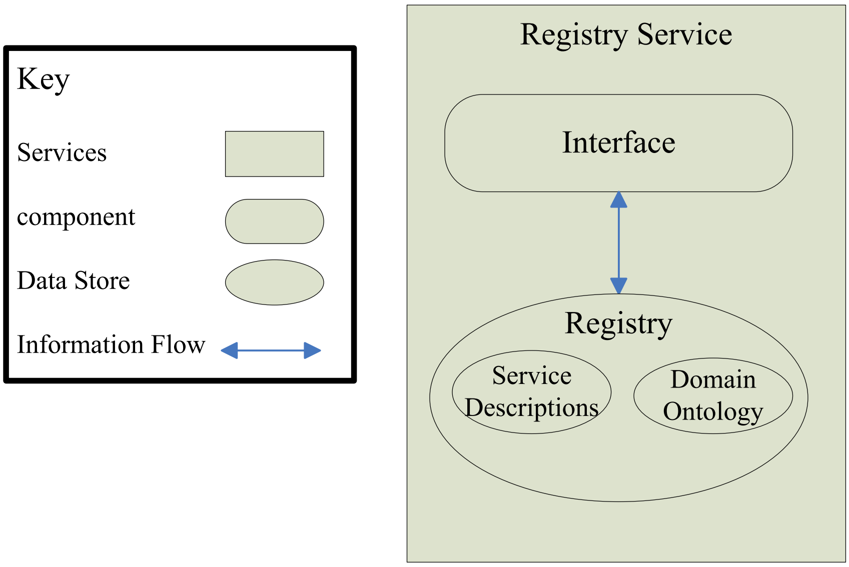


Figure 3-7 Registry Service

1. Registry: the registry is a conceptual data store and might be realized by different data management technologies. In this research, the registry is implemented as XML data.
2. Interface: the interface interacts with other services by exposing a set of methods for other services to invoke. It accepts one invocation call at a time as the input; it then analyses the information received from the invocation call such as the method specified and the parameters passed over and finally it conducts appropriate operations against the registry and sends the results back in response to the invocation call.

The Registry Service only accepts invocation calls and does not call other services. It might be realized in multiple instances in practice to distribute the workload to achieve a better performance.

3.6 Characteristics of the SLEDI

After the creation of the data integration activity framework, various characteristics of the SLDEI can be discussed by comparing it with other solutions of the data integration with respect to the framework. The solutions include the federated database system and the data warehousing which have been introduced in previous chapters and the comparisons are made from different aspects including the complexity, flexibility, scalability and the performance. The summary of the comparisons is shown in the Table 3-1 followed by a brief discussion. Technical details of the three main processes are provided in chapters 4, 5 and 6 respectively, whilst details of the comparison are provided in chapter 8.

	Federated Database System	Data warehousing	SLEDI

		Tightly coupled	Loosely coupled		
Integrated data store		Virtual	Virtual	Materialized	Virtual
Data Integrating Process	Complexity	Medium	High	Very High	Low
	Flexibility	Low	Medium	High	Very High
	Scalability	Medium	High	Low	Very High
Information Supplying Process	Complexity	Medium	High	Low	Very High
	Flexibility	Medium	High	Low	Very High
	Performance	High	Medium	Very High	Low
Evolution Handling Process	Complexity	High	Medium	Very High	Low
	Flexibility	Medium	High	Low	Very High
	Scalability	Medium	High	Low	Very High

Table 3-1 Characteristics of the SLEDI

In the Federated database system, the data integrating process is achieved by creating federate schemas to specify the relationships between the virtual IDS and the data sources. The tightly coupled federated database normally only contains one federated schema and the loosely coupled one may contain multiple federated schemas. The information supplying process is realized by processing the user queries over the federate schemes into the queries referring to the data sources. And the evolution handling process is accomplished by maintaining the federate schemas. Compared to other solutions, building the federated schemas is a medium complex process and may not have good scalability since the set of data sources is getting larger, the federate schemas are getting more complex. The query answering is based on the processing of the federated schemes hence might delivers medium performance and may not be very flexible. The maintenance of the federated schemas may become complex when the scale and frequency of the evolution occurring in the data sources are getting higher, and can be infeasible in the worst cases.

In Data Warehousing, the data integrating process is achieved by extracting the data from data sources and transferring them into the materialized IDS. The information supplying process evaluates the user queries directly against the IDS. And the evolution handling process is achieved by maintaining the IDS. Compared to other solutions, building the IDS can be very complex and hard to scale, as data sources may require complex processes to extract their data and more data sources mean more processes. However, Data Warehousing may provide the best performance for the supply of information as user queries are processed directly against the materialized IDS. The evolution handling in Data Warehousing has got the highest complexity, as evolution may result in building new complex processes to refresh the data in the IDS and can quickly become infeasible when data sources frequently evolve.

In SLEDI, the data integrating process is achieved by building application domain ontology at an abstract level, and then each data source specifies the information it provides with respect to the application domain ontology through the IPU mechanism. The information supply is realized by processing the user queries and producing a service execution plan to generate the results. The evolution handling is achieved by maintaining the definition of the IPUs which are accommodated as service metadata of the IPSs. Compared to other solutions, creating the IPSs provides lower complexity and is relatively easy to scale, as more data sources simply means more IPSs to publish. The query processing is relatively complex as it first needs to determine the IPSs who provide the answers, and then extract the data from data sources to generate the final results data hence may result in lower performance. Nevertheless, the evolution handling is relatively easy to achieve, as evolution may only affect the descriptive information in the service metadata of the IPSs and automatic assistance might be employed, hence the maintenance costs can be decreased.

3.7 Summary

This chapter presents a descriptive framework for data integration activity. The new solution: Service Late binding Enabled Data Integration (SLEDI) is introduced. The constituent components of the SLEDI are presented to set up the context for the next few chapters. A brief discussion of the characters of the SLEDI is presented.

Chapter 4 introduces the process of data sources describing. The process is the first stage of the SLEDI. It virtually integrates data from autonomous data sources into the IDS. The IPUD algorithm is presented which describes data sources through the IPU mechanism. The IDS is then built through assembling the IPUs into IPSs and organizing the IPSs according to the organizational structure.

Chapter 4 Data Sources Describing

4.1 Introduction

Chapter 3 provided a descriptive framework for the data integration activity and an overview of the SLEDI method. The services and processes of the SLEDI were introduced and the characters of the SLEDI were briefly discussed with respect to the framework.

This chapter introduces the process of Data Sources Describing with its three basic steps: Application domain ontology construction, IPU creation and IPS assembly. The IPUD algorithm is presented for describing data sources through the IPU mechanism. The details of using global definition and local definition of IPU to integrate the data from autonomous data sources are explained. This is followed by the description of assembling the IPUs into IPSs with respect to the organizational structure, hence the IDS is formally built up. The formal model of the IPU definitions, IPS and the IDS are presented by using set theory.

4.2 Overview of the Data Sources Describing

As introduced in Chapter 3, one of the design purposes of the SLEDI is to integrate data from the autonomous data sources into an Integrated Data Store (IDS). As a result, end users can interact with the IDS to meet their information requirements. In the SLEDI method, the data integration is achieved by building the IDS through the process of Data Sources Describing. The process is the first stage in establishing the data integration system hence other processes can be conducted against the IDS. From the point view of the end users, there is only one single integrated data source: the IDS. Because the SLEDI adopts the virtual approach of data integration, the IDS is a virtual database where all its data actually reside in the data sources. Hence the IDS has two objectives:

1. It provides a unified vision to represent the integrated data thus end users can describe their information requirements through raising queries against the vision
2. It provides the mappings to specify the relationship between the unified vision and the data sources. Consequently, the user queries can be answered by extracting the data from the data sources based on the mappings.

In order to achieve the above objectives, the unified vision of the (virtual) IDS is required to not only have the ability to represent the application domain but also to provide a platform for end users to raise queries. The mappings between the unified vision and the data sources are required to be specified in a well defined way thus users can get the answers to their queries by employing automatic assistance to find the relevant data sources and obtain the results data from them. More importantly, the mappings provide a solution for solving the various heterogeneity problems existing in the participating data sources as defined in section 1.2.3.

In this research, a domain ontology model is employed to represent the unified vision and the mappings are realized through the mechanism of the Information Provision Unit (IPU). Through formulating the Global Definition and the Local Definition of each IPU, the mappings are formally specified to establish the correlation between the application domain ontology (i.e. the unified vision) and the database schemas of the data sources. An algorithm termed Information Provision Unit Describing (IPUD) is designed which adopts the approaches of the Global-as-View (GAV) and parts of the Local-as-View (LAV) to create the definitions of the IPU. Because the SLEDI follows the approach of Software as a Service, the IPUs are finally realized through the Information Provision Service (IPS). After all the data sources are processed, the IPSs produced are grouped together with respect to the organizational structure of the data sources and published into the registry for other parts of the system to utilize.

The process of Data Source Describing can be divided into three elementary steps: *Application Domain Ontology Construction*, *IPU creation* and *IPS assembly*. After the application domain ontology is constructed, the IPU creation can be achieved by using the IPUD algorithm to create IPUs from the data sources. Consequently, the IPSs can be established from the IPUs and assembled according to the organizational structure of the data sources. The application domain ontology is denoted as **Onto**, a data source is denoted as **DS**, the local schema of a DS is denoted as $\mathbf{DS_{LS}}$, and the organizational structure of the data sources is denoted as **Org**. The IPU and IPS produced from a data source is denoted as $\mathbf{IPU_{(DS)}}$ and $\mathbf{IPS_{(DS)}}$ respectively. The IPUD algorithm hence can be formally defined as a function:

$\mathbf{IPUD : (DS_{LS}, Onto) \rightarrow \{\mathit{x} : IPU^{i}_{(DS)}\ (1 \leq i \leq n)\}}$

And the process of Data Source Describing for a set of data sources $\{DS_1, \dots, DS_n\}$ can be formally defined as a function:

$\mathbf{Data\ Source\ Describing : (\{\mathit{x} : DS_j\ (1 \leq j \leq n)\}, Onto) \rightarrow (Org, \{\mathit{y} : IPS_{(DS^{j})}\ (1 \leq j \leq n)\})}$

The elementary steps, the IPUD algorithm, the data models and the processes involved in the Data Source Describing are delineated in details in the following sections.

4.3 Application Domain Ontology

As discussed in section 1.3.3, although different data sources might store their data in schematically heterogeneous relational schemas, the real world objects described by these data are in the same application domain. Indeed, if the objects represented by different data sources do not overlap each other, the data integration will become too difficult. Obviously, a canonical model capturing the salient parts of the application domain can provide a standard representation of the objects, hence each data source is able to describe the data they provide with respect to the canonical model without having to concern other data sources. In this research, a domain ontology model is created to represent the application domain ontology thus playing the role of the canonical model.

4.3.1 Knowledge Representation

4.3.1.1 Representation Formalisms

To fulfill the requirement of explicitly and precisely representing the application domain, the technique of *Knowledge Representation* can be employed. Knowledge Representation is a research field that mainly concerns the formalism for describing a domain of discourse [15]. By defining a symbol system and assigning semantics to it, the formalism provides a high level description of the domain. Hence the explicit knowledge the symbols represents can be searched and the implicit knowledge can be further inferred based on the reasoning methods the formalism supports. The formalisms may be roughly divided into two categories: logical-based and non-logical-based [04]. The logical-based formalism is developed based on the intuition that the facts of the domain can be represented by predicate calculus hence can be unambiguously interpreted. On the other hand, the non-logical based formalism is thrived from more cognitive notions which simulate the experience of memory recall and task execution of humans.

The Semantic Networks [53] and Frames [63] are two examples of the non-logical-based formalisms which follow the network-shaped cognitive structures. Although they can represent the sets of individual objects and relationships among them, they suffer from lacking precise semantic characterization. This can result in systems with identical components and relationship names behaving totally differently to each other [04]. For tackling this problem, the network structure can be assigned semantics by relying on First-order logic; hence the research in the area of Description Logic began by following this approach.

4.3.1.2 Description Logic

Description Logic (DL) is a family of knowledge representation formalisms [04, 15] which can be used to define formal languages for representing knowledge and reasoning about it. The network structure based knowledge can be expressed in DL such as assigning a group of individuals into a class and categorizing the classes into taxonomy. A complex class can be formed from atomic classes by declaring its definition formulae in first-order logic. Equipped with model-theoretic semantics, logical reasoning may be achieved by assigning the proper inference procedures.

DL languages represent the knowledge of an application domain by first defining the relevant concepts of the domain (i.e. its terminology), and then using these concepts to specify properties of objects occurring in the domain. The elementary descriptions in DL languages are atomic concepts and atomic roles. Complex concepts can be further constructed from them by using concept constructors inductively. The main characteristic that distinguishes different DL languages is the concept constructors the languages supports. The notations for denoting the atomic concepts, atomic roles and the concept constructors used here follow the conventional syntax and rules [05]. The letters A and B are used for denoting atomic concepts, the letter R for atomic roles, and the letters C and D for concept descriptions:

C, D $\rightarrow$

A | (atomic concept)

$\top$ | (universal concept)

$\bot$ | (bottom concept)

$\neg A$ | (atomic negation)

$C \sqcap D$ | (intersection)

$\forall R.C$ | (value restriction)

$\exists R.\top$ | (limited existential quantification)

The above syntax forms the basic description language AL (attributive language). Other languages with more expressive power can be added by extending the constructors the language supports. For example, in the ALN DL language, the cardinality restriction is added to describe the roles:

$\geqslant n\ R$ (at least cardinality restriction)

$\leqslant n\ R$ (at most cardinality restriction)

The semantics of DL languages can be formally assigned by introducing the interpretations *I* that consist of a non-empty set Δ^I (the application domain) and an interpretation function. Every atomic concept A is assigned a set $A^I \subseteq \Delta^I$, and every atomic role R a binary relation $R^I \subseteq \Delta^I \times \Delta^I$. Intuitively, the Δ^I are all the individuals in the application domain. A^I is the set of all individuals which are the instances of the concepts A, and R^I is the set of all the pairs of individuals with the atomic role relation holding in each pair. The interpretation function can be extended to concept descriptions by the following inductive definitions [05]:

$\top^I = \Delta^I$

$\bot^I = \emptyset$

$(\neg A)^I = \Delta^I \setminus A^I$

$(C \sqcap D)^I = C^I \sqcap D^I$

$(\forall R.C)^I = \{ a \in \Delta^I \mid \forall b.(a,b) \in R^I \rightarrow b \in C^I \}$

$(\exists R.\top)^I = \{ a \in \Delta^I \mid \exists b.(a,b) \in R^I \}$

$$(\geqslant \text{ n R})^{I} = \{ a \in \Delta^{I} \mid |\{b.(a,b) \in R^{I}\}| \geqslant n \}$$

$$(\leqslant \text{ n R})^{I} = \{ a \in \Delta^{I} \mid |\{b.(a,b) \in R^{I}\}| \leqslant n \}$$

Under the definition above, the concept C is said to be subsumed by the concept D if $C^I \subseteq D^I$ and C is said to be equivalent to D when $C^I = D^I$. Since the semantics of concepts identifies DL languages as fragments of the first-order logic, the subsumption relationships between concepts can be reasoned out even if they are not explicitly specified.

4.3.2 Domain Ontology Representation

The ALN DL language, though simple, provides a richer modeling framework compared to the pure relational approaches. In addition, the ALN DL is equipped with reasoning services for checking concept and rule subsumption, hence has been chosen in the PICSEL project for representing the domain ontology [34]. In order to keep the focus of this research on solving the evolution problems, the domain ontology representation in this research is also based on ALN DL and extended with extra characters. Therefore the query rewriting algorithm of the PICSEL can be employed in this research, which will be discussed in details in the next chapter. The domain ontology model consists of two components: the *Terminology* and the *Rule*.

4.3.2.1 Concept Description

As discussed above, the DL lends itself well to naturally conceptually modelling the application domain by grouping the individual objects and categorizing their relationships. In this research, because of the autonomous nature of the data sources, the same objects might be realized in different forms of data (e.g. different data types). As one of the objectives of the domain ontology is to provide an interface with which the end user can interact, it is required that the domain ontology also presents a canonical specification about how the objects are realized in data. In other words, the domain ontology is not only required to provide an agreement across the application domain at a semantic level but also at syntactic level.

By extending the interpretation *I* described in section 4.3.1.2, we refer to the non-empty set Δ^I as the *semantic spaces* and further introduce another non-empty set Δ^D as the *syntactic spaces* and a set of *predicates* P^D over Δ^D. The elements of Δ^D are purely data symbols and the elements of P^D are ordinary predicates with arity n. Another function f^I is introduced to assign each element in Δ^I with an element of the Δ^D, $f^I : \Delta^I \times \Delta^D$. Thus each individual object in the application domain is denoted by a specific data symbol from the Δ^D. The predicates in P^D discussed in this research are ordered relations (e.g. ≤, <) which are used to describe the possible partial order associated with a subset of Δ^D. For each concept A, syntactic space $A^D \subseteq \Delta^D$ indicates the data symbols used to denote the instances of A in the application domain. The subset A^D might be described by a data type and associated with ordered relations. Consequently, each concept is equipped with a semantic space to describe its semantic meaning by a possible complex ALN DL statement and a syntactic space to provide a canonical format of the data to denote its instance objects.

4.3.2.2 The Terminology

Similarly with the PICSEL project, the Terminology component of the domain ontology model consists of a set of statements which can be categorized into two types: the *concept definition* and the *concept inclusion*. The concept definition is in the form of **Concept := Expression** and the concept inclusion is in the form of **Concept** $\sqsubseteq$ **Expression** or $\mathbf{Concept_1} \cap \mathbf{Concept_2} \sqsubseteq \bot$[34]. The Expression is an ALN DL concept description hence the constructors allowed are intersection, value restriction, cardinality restriction and atomic negation as described in section 4.3.1.2. In this research, for the concept definition statements, any concept is allowed to appear at the left hand side of the statements at most once, and the concepts do not appear in the left hand side for any of the statements termed as *Atomic Concepts*. A concept C is considered depending on the concept C' if C' appears in its definition expression. A set of the statements is said to be acyclic if there is no cycle in the concept dependency. Only atomic concepts are allowed in the concept inclusion statements. As only acyclic concept definition is considered in this research, the set of concept definition can be unfolded by iteratively substituting every concept with its definition. Thus every concept definition can be unfolded and put in a normal form of a conjunction of *basic forms*: A (atomic concept), $\neg$A, ($\geqslant$ n R), ($\leqslant$ n R), or the complex form $\forall R_1 \forall R_2 \cdots \forall R_K.D$, where D is a basic form.

4.3.2.3 The Rule

Since the terminology statements are DL based, only concept (unary relation) and role (binary relation) can be expressed. The domain ontology model is extended with the *Rule* component to express the ordinary relation (n-ary relation). The Rule is also constituted by a set of statements. Each statement is in the form of Horn Rules: $\mathbf{R(\bar{y})\ :\!-\ \forall \bar{x}\ \ P_1(\bar{x}_1) \wedge \cdots \wedge P_n(\bar{x}_n)}$ where the left hand side is the *consequence* and the right hand side is the *antecedent*. The $\bar{x}_1$, …, $\bar{x}_n$, $\bar{y}$ are tuples of variables or constants which are all included in $\bar{x}$. The relations P_1, …, P_n may be concepts, roles and ordinary relations. We require that any variable appears in $\bar{y}$ must also appear in $\bar{x}_1 \cup \ldots \cup \bar{x}_n$. The relations which do not appear in any consequence of the Rule statements are called *base relations*. The base relation P($\bar{x}$) is a *concept-base* if P is a concept from the terminology and P(x,y) is a *role-base* if P is a role. A relation R is said to depend on a relation P if P appears in the antecedent of a statement where R is the consequence. A set of rule statements is recursive if there is a cycle in the dependency relationship among the relations. In this research, we only consider non-recursive rule statements; every base relation must be either a concept-base or a role-base and the consequence of a rule must be an ordinary relation. Hence every statement can be rewritten into a normal form so that the antecedent only includes concept-bases and role-bases. We also employ the unique name assumption that for every pair of distinct variables appearing in a rule statement, there is inequality that $x \neq y$.

After augmenting the domain ontology model with the Rule component, the interpretation *I* is also extended. It assigns every constant *a* with an object $\alpha^I(a) \in \Delta^I$, and every relation of arity n with a relation of arity n over the Δ^I. Thus an interpretation *I* is a model of a rule statement $R(\overline{y})$:- $P_1(\overline{x}_1) \wedge \ldots \wedge P_n(\overline{x}_n)$ if whenever α is a mapping from the variables to the Δ^I such that $\alpha(\overline{x}_i) \in P_i^I$ for every base relation of the antecedent, then $\alpha(\overline{y}) \in R^I$. Consequently, an interpretation *I* is a model of a domain ontology if it is a model of each of its Terminology and Rule components.

4.3.3 Domain Ontology Model

After the introduction of the domain ontology which has been given in the previous section, the formal model of the domain ontology can be defined.

4.3.3.1 The Terminology

Concepts and roles are the central elements in the domain ontology. Role has three properties. The first property *Name* is to identify it in the domain ontology. The second and third properties *con1* and *con2* refer to the subject concept and the filler concept associated by the role respectively. Concept is described by four properties: *Name*, *Level*, *Semantic space*, and *Syntactic space*. The *Name* is a string which can be used to identify the concept. The *Level* is either atomic or composite. Any concept which is not an atomic concept is labelled as composite in its level property. The *Semantic space* is a concept definition or concept inclusions to describe the semantic meaning of the concept. As discussed in 4.3.2.2, concept definition can only appear in the semantic space of composite concepts and concept inclusions can only appear in the semantic space of atomic concepts. The concept definition is represented by a statement in the normal form of an ALN DL expression, and the normal form is a conjunction of the various basic forms. The concept inclusion can be represented by either an ALN DL expression or by a contradiction of two atomic concepts. The *Syntactic space* is a string to denote the data type of the data for representing the instance objects of the concept. The model of the terminology is described below:

Role : (*nrole* : **Name**, *con1* : **Concept**, *con2* : **Concept**)
Name : String

Concept : (*nconcept* : **Name**, *l* : **Level**, *sem* : **Semantic Space**, *syn* : **Syntactic Space**)

Level : String

∀X : **Level** X ∈ {*“atomic”, “composite”*}

Semantic Space : Concept Definition | Concept Inclusion

Concept Definition : ALN-Expression

ALN-Expression : {*bf* **: Basic Form}**

Basic Form (*ft* **: Form Type,** *f* **: Form**)

Form Type : String

∀Y : **Form Type** Y ∈ {*“atomic”*, *“atomic negation”*, *“cardinality restriction”*, *“value restriction”*}

Form (*otr* : **Operator,** *ord* : **Operand**)

Operator : String

∀Z : **Operator** Z ∈ {“null”, “¬”, “>=” , “<=”, “=”, “∀ ” , “Ø”}

Operand : { *ord*: **Positive Integer | Concept | Role | Form}**

There are four types of basic form. Each type employs a different set of operators and operands to construct its specific representation hence each is defined individually.

Form (*otr* = “null”**,** *ord* : **Concept**) (when *ft* = *“atomic”*)

Form (*otr* = *“*¬*”*, *ord* : **Concept**) (when *ft* = *“atomic negation”*)

Form (*otr* = *“>=”* | *“<=”* | *“=”, ord* : (*n* : **Positive Integer**, *r* : **Role**)) (when *ft* = *“cardinality restriction”*)

Form (*otr* = *“*∀ *”*, *ord* : (*r* : **Role,** *f* **: Form**) (when *ft* = *“value restriction”*)

Concept Inclusion : ALN-Expression | Contradiction

Only when the subject concept is an atomic concept, the concept inclusion might be assigned into its semantic space. As the contradiction considered here is only required to indicate the object atomic concept which has no intersection with the subject atomic concept, the contradiction is defined below.

Contradiction : (*otr* : **Operator**, *ord* : **Concept**) (*otr* = *“*Ø*”*)

Syntactic Space : (*dt* : **Data Type**)

Data Type : String

∀X : **Data Type**, X ∈ {*“string”, “decimal”, “integer”, “date”, “Boolean”*}

The *syntactic space* only has one property *data type*, the data type is used to specify the type of data associated with the concept for denoting the instance of the concept in the application domain. The data types defined above can be extended in practice.

After the concepts and roles have been formally defined, the Terminology of the domain ontology can be defined.

Terminology : ({*con* **: Concept**}, {*rol* **: Role** })

4.3.3.2 The Rule

As introduced in 4.3.2.2, the Rule contains a set of statements with each statement in the form of Horn Rules. Each statement consists of two parts: the *antecedent* and the *consequence*. The consequence is an ordinary relation with a variable list and the antecedent is the conjunction of a set of base relations. The Rule is formally defined below:

Rule : {*sta* **: Statement}**

Statement : (*conse* : **Consequence,** *ante* **: Antecedent**)

Consequence : (*nconse* **: Name, {***vconse* **: Variable}**)

Variable : (*vname* : **Name,** *dt* **: Data Type**)

Antecedent : (*nante* **: Name, {***br* **: Base Relation}**)

Base Relation : (rtp **: Relation Type,** (*con* **: Concpet |** *role* **: Role |** *conse* : **Consequence),** {(*vbr* **: Variable**) **|** (*const* **: Constant**)})

Constant : (*dt* : **Data Type,** value : **String**)

Relation Type : String

∀X : **Relation Type**, X ∈ {*"concept" , "role" , "ordinary"* }

For each of the distinct relation types, the base relation is represented differently hence is defined separately below

Base Relation : (rtp=*"concept"*, *con* **: Concpet,** *vbr* **: Variable |** *const* **: Constant**)

Base Relation : (rtp=*"role"*, *role* **: Role, {***vbr* **: Variable} |** {*const* **: Constant}**)

Base Relation : (rtp=*"ordinary"*, *ordi* **: Consequence {***vbr* **: Variable} |** {*const* **: Constant}**)

The *constants* have two properties; the first property indicates its data type and the second property represents its value in the form of string. As discussed above, some constraints have to be followed when constructing the Rules. If the relation type is *ordinary*, the consequence of other rule statements can be used as the value of the *ordi* and no recursion is allowed. The variables appearing in the consequence of any statement must also appear in the base relations of its antecedent.

4.3.3.3 Domain Ontology Construction

After formally defined the model of the Terminology and Rule components of the domain ontology, the model of domain ontology can be formally denoted as

Domain Ontology : (*ter* **: Terminology,** *rule* **: Rule)**

Thus the domain ontology of the application domain can be constructed based on the model. In practice, the application domain ontology might already exist with realization in other conceptual data models such as the Entity-Relationship (ER) model. In this case, a transformation is required in order to realize the application domain ontology into the model described above. Take the ER model as an example, a property of an Entity type might be defined as an atomic concept *Cp* and the Entity type might then be defined as a composite concept *Ce*. A role might be defined to take *Ce* as its subject and *Cp* as its filler. In similar fashion, a relationship type might be defined as a concept and connected by roles to the concepts which represent its participating Entity types. How to transfer the domain ontology represented by one formalism (e.g. ER model) into another formalism (e.g. the domain ontology model above) is out of the scope of this research hence will not be discussed further. However, a very simple example from the mental health application domain is used here to illustrate the ontology construction:

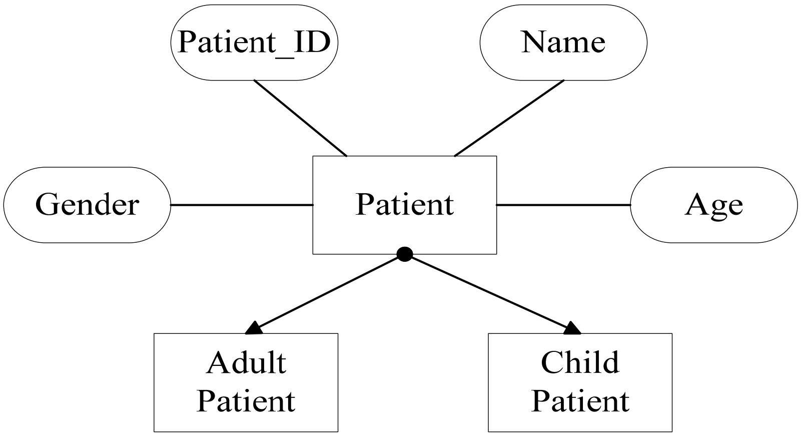


Figure 4-1 simple example of Patient information

In the above ER model, the patient entity type has four properties: Gender, Patient_ID, Name and Age; and two subclass entity types: Adult and Child patient. The application domain ontology for modelling the patient information can be constructed in the following way.

Atomic Concept:

{ PatientID, Name, Gender, Age, Classification }

Role:

{HasPatientID, HasName, HasGender, HasAge, HasClassification}

Composite Concept: {

Adult $\sqsubseteq$ Classification; Child $\sqsubseteq$ Classification; Adult $\cap$ Child $\sqsubseteq$ $\perp$;

Patient := (=1HasPatientID.PatientID) $\sqcap$(=1HasName.Name) $\sqcap$ (=1HasGender.Gender) $\sqcap$ (=1 HasAge.Age) $\sqcap$ (=1HasClassification.Classification);

AdultPatient **:=** Patient $\sqcap$ $\forall$HasClassification.*Adult*;

ChildPatient **:=** Patient $\sqcap$ $\forall$HasClassification.*Child*; }

Rule: {… ;

Patient-Info(X_PatientID, X_Name, X_Gender, X_Age) **:-** Patient(X) $\wedge$ HasPatientID(X,

X_PatientID) ∧ HasName(X, X_Name) ∧ HasGender(X, X_Gender) ∧ HasAge(X, X_Age);
…}

Table 4-1 application domain ontology example

The domain ontology model provides a mechanism for representing the application domain, but the design of the content of the domain ontology still relies on the expertise on the application domain and the data sources. Through exploiting the expressive power of the domain ontology model, a high quality design of an application domain ontology should be able to express the application domain precisely to fulfill the users' information requirements; as well as allowing the data sources to easily correlate their data with the application domain ontology.

4.4 The Information Provision Unit Describing Algorithm

After the domain ontology is constructed, the first objective of the IDS as discussed in section 4.2 is achieved. The second objective can be achieved through specifying the mappings between each data source and the domain ontology. The IPUD algorithm for formulating the mappings is described in detail below.

4.4.1 Overview of the IPUD

In this research, as introduced previously, despite the fact that the participating data sources might manage their data through employing a distributed database system or even an integrated database system, all the data are realized in the relational data model. Hence each data source is represented by a single relational schema referred to as *local schema*. Because the data sources are autonomously controlled, they may only expose part of their data to be accessed. The *correlations* between the exposed data and the domain ontology are specified formally through the IPU mechanism. The IPUD algorithm is designed to process one data source at a time which takes the local schema and the domain ontology as input, and generating the IPUs as the output (i.e. the mapping between the data source and the domain ontology). Through the IPUD, each data source first organizes its exposed data into a set of IPUs and then specifies the global definition and local definition for each IPU to constitute the mapping. The process of using IPUD to describe a data source is depicted in Figure 4-2

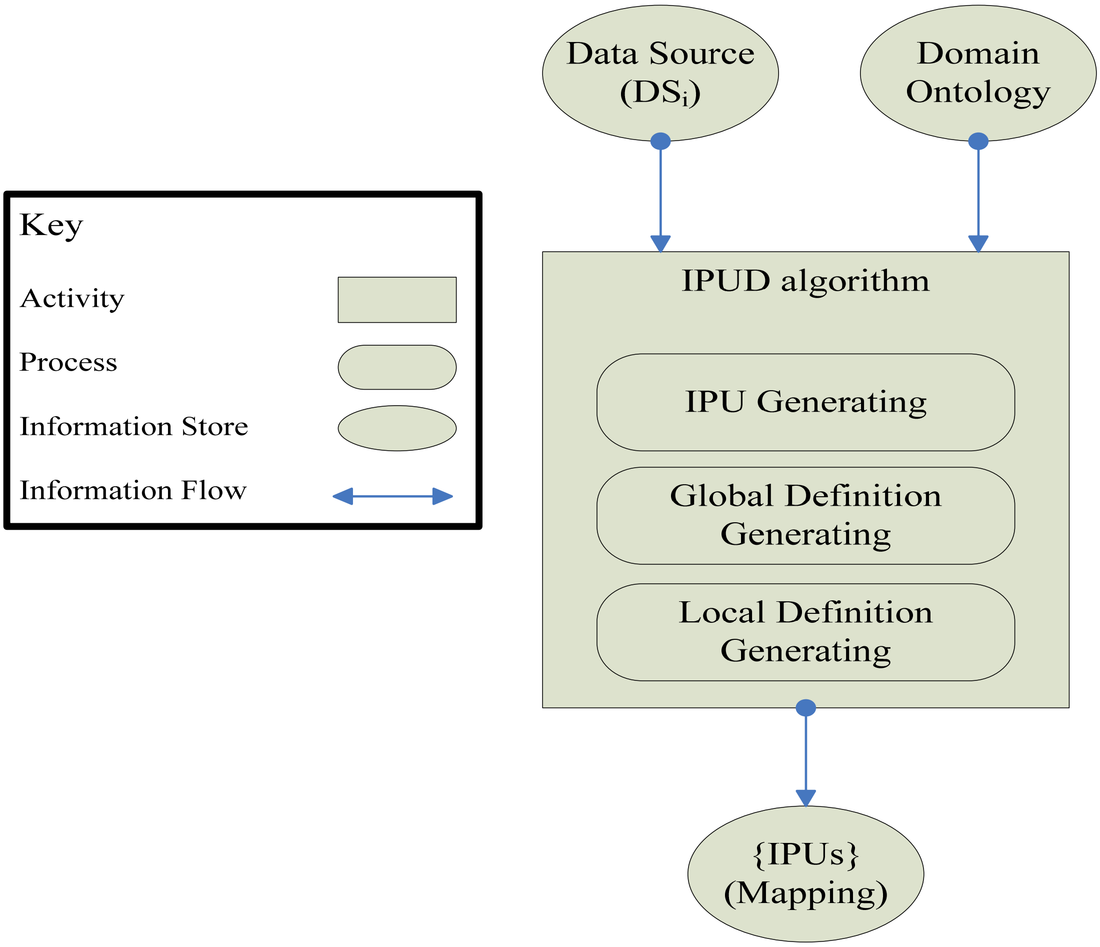


Figure 4-2 Data Source Describing by IPUD

4.4.2 The Information Provision Unit

For the purpose of correlating each data source with the domain ontology, the mapping is required to describe formally the relationship between the elements in the local schema and the elements in the domain ontology. The ways of specifying the mapping can be broadly categorized into two approaches: Local-as-View (LAV) and Global-as-View (GAV) [55, 37]. In the LAV approach, each element of the local schema is associated with an expression over the domain ontology, the concept the element represents corresponds to the concept the expression represents. On the contrary, in the GAV approach, each element of the domain ontology is associated with an expression over the local schema. Each approach has its advantages and drawbacks. While the LAV approach appreciates its high expressivity and flexibility of representing the content of the data source relatively to the domain ontology, the user query processing may encounter high complexity. Compared to the LAV approach, on the other hand, the GAV approach suffers limited flexibility for representing the data source but enjoys the simplicity for the user query processing

[60]. For example, when the user query is based on function-free Horn rules such as Datalog, the query processing for GAV can be done by simply rules unfolding. In this research, the mapping is built by following the GAV approach and augmented with some features of the LAV approach in order to exploit the simplicity of user query processing while appreciating the flexibility for describing the data source.

As the data sources are autonomously managed, each data source is required to firstly determine the data they are willing to share. Then describe the data through the *Information Provision Unit* (IPU). Each IPU is described through a *global definition* and a *local definition*. The global definition is formulated by the elements from the application domain ontology to describe what data this IPU provides at the application domain level; and the local definition describes how the data can be formed from the data source by using the elements in the local schema. A data source can generate as many IPUs as needed to describe its exposed data. Since the IPU acts as a middle layer between the application domain ontology and heterogeneous participating data sources, the various heterogeneity problems defined in section 1.2.3 may be solved through the IPU mechanism. The details of the global definition and local definition are explained in the following sections.

4.4.3 Global Definition

The global definition of an IPU constitutes a compulsory *content* part and an optional *constraint* part. The content is expressed by a rule which associates a *source relation* with a *base relation* from the domain ontology in the form of $\mathbf{V(\overline{x})\ {:}{-}\ P(\overline{x})}$. The source relation $\mathbf{V(\overline{x})}$ (also referred to as the *head* of the content) represents the data the IPU provides in the form of a relation. The base relation $\mathbf{P(\overline{x})}$ (also referred to as the *body* of the content) indicates the element from the domain ontology whose instance can be found in the source relation. The base relation is either a concept or a role in the domain ontology which itself may be associated with an (possibly complex) ALN DL expression. We require that the variables in the head and the body of the content are identical hence the source relation $\mathbf{V(\overline{x})}$ can only be either a unary or a binary relation.

The content specifies the mapping between the data the IPU provides and the application domain ontology by following the GAV approach. For the purpose of providing higher flexibility in describing the data source, the global definition is augmented with the constraint for fine-tuning of the characterization of the data the IPU provides. Two types of constraints are employed in this research: the *terminological constraints* and the *integrity constraints*. The terminological constraints are expressed in the form of $\mathbf{V(X) \sqsubseteq C}$ where C is a concept expression over the domain ontology. The $\mathbf{V(X)}$ and $\mathbf{C}$ are referred to as *head* and *body* of the terminological constraints respectively. The head is required to be identical to the source relation of the content part. And the integrity constraints are expressed in the form of $\mathbf{V_1(\bar{x}_1) \wedge \ldots \wedge V_n(\bar{x}_n)} \sqsubseteq \perp$where the $\mathbf{V_n(\bar{x}_n)}$ $\mathbf{(1 \leq i \leq n)}$ is the source relation or negation of the source relation of the content of the IPUs the same data source provides. The $\mathbf{V_1(\bar{x}_1)}$ is also required to be identical with the source relation of the content part.

After the introduction, the formal model of the global definition of the IPU can be defined:
Global Definition : (*nante* **: Content,** [*br* **: Constraint**])

Content : (_conthe_ **: Head,** ***contbo*** **: ContentBody)**
Head : (_hdn_ **: Name, {**_vhd_ **: Variable})**
ContentBody : (_cbbr_ **: Base Relation)** (where *rtp* ∈{*"concept"*, *"role"*} ∧ *vbr* = *vhd*)

Constraint : (_cstra_ **: Terminological Constraints | Integrity Constraints)**
Terminological Constraints : (_tche_ **: Head,** *tcbd* **: ALN-Expression)** (where *tche* = *conthe*)
Integrity Constraints : (_ich_ **: Head, {** *icbd* **: Head })** (where *ich* = *conthe* ∧ *icbd* ∈{*conthe*})

Through the global definition model, the data an IPU provides is specified with respect to the domain ontology hence the data can be understood unambiguously across the application domain without concerning how the data are actually realized in different data sources.

4.4.4 Local Definition

After the global definition is specified, the data an IPU provides is precisely described at the application domain level. Because of the actual data is stored in the data source, the way in which the IPU data is generated must be formally specified. Thus each IPU is equipped with a local definition for achieving this purpose. As the data is realized through the relational data model, the local definition is formulated as a virtual *view* of the local schema of the data source. A virtual view is essentially a named query over the local schema and the data of the view is constructed through executing the query against the local schema. The virtual view provides the mechanism for adding a layer between generating the data the IPU provides and the actual realization of the data in the local schema. Hence each data source can enjoy the flexibility of implementing their own local schema according to their needs and the data the IPU provides can be derived from the actual relations of the local schema.

4.4.4.1 Query Languages

For expressing the virtual views, some query languages are commonly used in literatures such as *relational algebra*, *relational calculus* and *conjunctive query* [24, 87]. The relational algebra and relational calculus have identical expressive power while the views expressed by the relational algebra are more prescriptive oriented and the views in the form of relational calculus are more descriptive oriented. The conjunctive query is a restricted form of first-order logic query which provides a logic based approach that can be used to specify the queries over the relational schema. The expressive power of the conjunctive query is equivalent to the Select-Project-Join queries in relational algebra [87]. As this research is only considering the read only queries from the end users, the conjunctive query is chosen and extended with extra features for expressing the views to fulfill the research needs. Furthermore, because the logic based conjunctive query formulates the views in a well formed and abstract way, not only can it be translated into different forms of local queries with respect to the various DBMS systems the data sources might employ (which will be discussed in details in chapter 5), but it also provides the ground for supporting the automatic assisted evolution handling (which will be discussed in details in chapter 6).

4.4.4.2 Conjunctive Query

The notations used for describing the relational schemas and conjunctive query in this thesis follow the syntax in [87]. Let a local schema **LS** be constituted by a set of relations $\mathbf{R_1,\ldots R_n}$. each relation $\mathbf{R_i}$ ($\mathbf{1 \le i \le n}$) is constituted by a set of attributes $\mathbf{A_{i1},\ldots A_{im}}$. And each attribute $\mathbf{A_{ij}}$ ($\mathbf{1 \le i \le n,\ 1 \le j \le m}$) is described by a name and a data type which indicates in which types of data the instances of this attribute are realized. A logical rule is expressed by assigning a *body* to a *head* in the form of $\mathbf{H(\overline{y})\ :-\ \forall \overline{x}\ B_1(\overline{x}_1) \wedge \cdots \wedge B_n(\overline{x}_n)}$ where $\overline{x}_{1,}$ …, $\overline{x}_{n,}$ $\overline{y}$ are tuples of variables or constants. The variable quantification is omitted thereafter for a shortcut and the rule is regarded as *safe* if $\mathbf{\overline{y} \subseteq \overline{x}_1 \cup \ldots \cup \overline{x}_n}$. This safety requirement is to assure that there are no undefined variables in the head. Each atom of the body $\mathbf{B_i}$ ($\mathbf{1 \le i \le n}$) is called a subgoal. A *conjunctive query* (CQ) is a safe rule which is evaluated by applying all possible substitutions of values for the variables in the body. If a substitution makes all the subgoals true, it forms a valid instance data for the head relation. Hence a CQ can be treated as a virtual relation which is derived from its subgoal relations.

To meet the research needs, the conjunctive query is extended by applying certain constraints. The subgoals of a CQ are categorized as three different types: *ordered relation subgoal*, *converting relation subgoal* and *regular relation subgoal* (also referred to as ORS, CRS and RRS respectively for short). A subgoal $\mathbf{B_i(\overline{x}_i)}$ is an ORS when it satisfies the following constraints:

1. $\mathbf{B_i(\overline{x}_i)}$ is a binary relation ($\mathbf{B_i(X_i,Y_i)}$)
2. $\mathbf{X_i}$ and $\mathbf{Y_i}$ are either two variables or one variable and one constant.
3. the binary relation expresses the relationship between $\mathbf{X_i}$ and $\mathbf{Y_i}$ by the means of orders (i.e. >, ≥, <, ≤, =)

A CRS is a unary relation $\mathbf{B_i(X_i)}$ where $\mathbf{X_i}$ is a variable and assigned with an expression. The expression is constituted by *operands* and *operators* where the operands are either constants or variables from other subgoals. The operators specify how to construct the instance data of the $\mathbf{X_i}$ from the instance data of the operands such as arithmetic calculation and string operations. A RRS $\mathbf{B_i(\overline{x}_i)}$ is neither an ordered relation nor a converting relation, and $\mathbf{B_i(\overline{x}_i)}$ can be in any arity and $\mathbf{\overline{x}_i}$ are all variables.

4.4.4.3 View Definition

A virtual view can be defined as a union of CQs where all the CQs have identical heads. A CQ can assign each of its RRS with an actual relation from the local schema to derive the instance data for its head and all the instance data of the view can be derived by the union of the heads of the CQs. Because the view itself can be treated as a virtual relation which is represented by its head, it can be assigned as a RRS for other CQs. By allowing the assignment of RRSs to views as well as actual relations, the arbitrary complex view can be defined through the CQs and their assignments.

A *normal form* of CQ assignment is defined in this thesis to facilitate building the view expression. A CQ assignment is in the normal form if *all* of its RRS are assigned to the actual relations in the local schema. And a CQ assignment is in the *negation normal form* if none of its RRS is assigned to an actual relation, in other words, all of its RRS are assigned to virtual views. Because any actual relation can be assigned to a virtual view, any CQ can be rewritten to get the (negation) normal form assignment by creating extra virtual views. The (negation) normal form is also applied to the view definition in similar fashion that a view is considered to be in the normal form if all of the CQs in its definition are in the normal form and it is in the negation normal form if all of the CQs are in negation normal form. It is required that all the CQ assignments and views must be in either the normal form or negation normal form. This constraint does not affect the expressive power of the views and provides the leverage for the user query processing and the automatic assisted evolution handling which will be discussed in the next chapters.

A view $\mathbf{V}$ is said to directly depend on a view $\mathbf{V_1}$ if $\mathbf{V_1}$ is assigned to a RRS in the definition of $\mathbf{V}$. It is required that there is no circle in the dependency relation among the views. Thus a view definition can be conceptually illustrated as a directed acyclic graph (DAG) where each *vertex* represents an actual relation or a virtual view and each *arc* represents the direct dependency from a view to a view or to an actual relation. The view $\mathbf{V}$ is referred to as the *root* which has no incoming arc and the *leaves* which have no outgoing arc which are assigned to actual relations in the local schema:

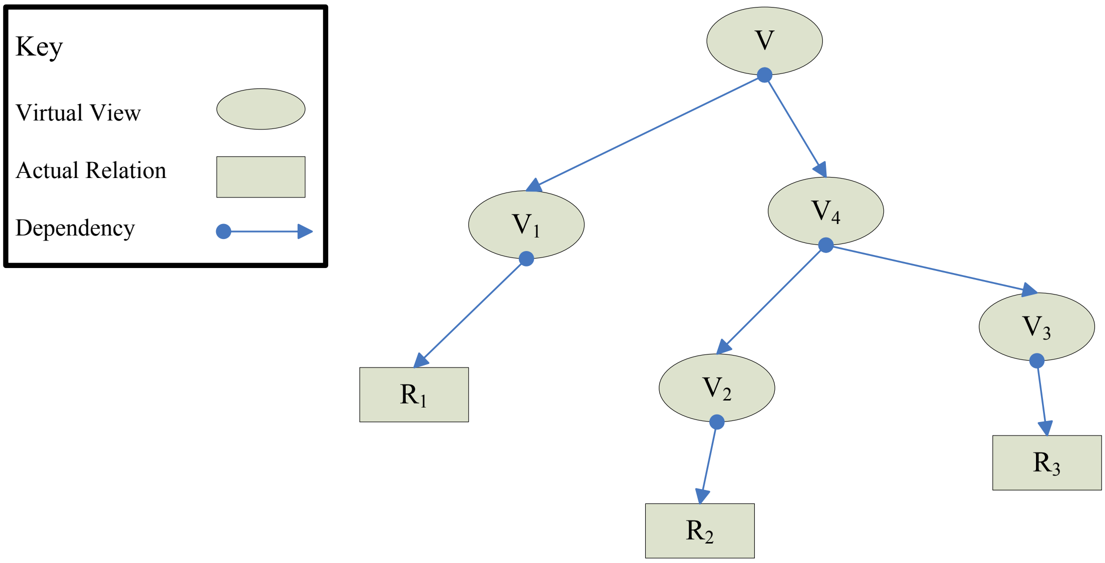


Figure 4-3 DAG structure of view expression

All the virtual views involved in a view definition **V** are categorized as two types: *basic* and *intermediate*. The views in the normal form are considered as the basic views (e.g. V_1, V_2, V_3, in Figure 4-3), and the views in the negation normal form are intermediate views (e.g. V, V_4, in Figure 4-3). The **V** is also referred to as the *root view* which must be an intermediate view. It is obvious that a view can be evaluated only on the premise that all the views it depends on have already been evaluated. From the DAG structure point of view, an actual relation is a leaf vertex and a view is a non-leaf vertex and any vertex must be evaluated before its direct predecessor.

4.4.4.4 The Model

After the introduction of the mechanism of the local definition, the formal model can be defined.

Local Schema : (lsname : **Name**, {rela : **Relation**})

Relation : (*relname* : **Name**, {*attr* : **Atrributes**})

Attributes : (*attname* : **Name**, *atttype* : **Data Type**)

A view is described by its name, head, type, all the CQs in its definition and a dependency list. The dependency list contains all the views (or actual relations) which are the direct successors of the view.

View : (*viewname* : **Name**, *vhead* : **Head**, *vtype* : **View Type**, *defi*: {*cq* : **Conjunctive Query**}, *dependency* : {***dpc*: View | Relation**})

View type : String

$\forall$X : **View type**, X $\in$ {*"basic"* , *"intermediate"* }

Conjunctive Query : (*cqname* : **Name**, *cqhead* : **Head**, *body* : {*sbg* : **Subgoal**}) (where *cqhead* = *vhead*)

Subgoal : (*sbgname* : **Name**, *sbghe* : **Head**, *sbgtp* : **Subgoal Type**, *sbgas* : **Assignment**)

Subgoal Type : String

∀Y : **Subgoal Type**, Y ∈ {*"regular", "ordered", "converting"*}

Different types of subgoals and views result in different assignments hence are modelled separately below:

An RRS of a subgoal of a CQ in a basic view is assigned to an actual relation of the local schema and each variable of the RSS head is assigned to an attribute.

Assignment : (*rela* : **Relation**, **{**(*asvar* : **Variable**, *asatt* : **Attribute**)**}**) (where *asvar* = *sbghe .vhd* ∧ *asatt* = *rela. attr* ∧ *sbgtp* = *"regular"* ∧ *vtype* = *"basic"*)

It is worth noticing in this type of assignment, an actual relation does not have to assign all of its attributes to the variables.

An RRS of a subgoal of a CQ in an intermediate view is assigned to a view and the head of the RRS is required to be identical with the head of the view.

Assignment : (*vie* : **View**) (where *sbghe* = vhead ∧ *sbgtp* = *"regular"* ∧*vtype* = *"intermediate"*)

The assignment of an ORS requires that the head of the subgoal contains exactly one variable and the assignment describes the correlation of order between the head variable and a variable or a constant. The variables and constant are required to have identical data type.

Assignment : (*ordoperator* : **Ordered Relation**, *odopd* : **Variable | Constant**) (where *sbghe.vhd.dt* = *odopd2.dt*)

Ordered Relation : String

∀Y : **Ordered Relation**, Y ∈ {">" , "<" , ">=" , "<=", "="}

The assignment of a CRS also requires the head of the subgoal contains exactly one variable. The assignment expresses the operations for constructing the instance of the head variable over a set of constants or variables. The operators described in this model are binary operators include arithmetic and string operators; and a unary operator "&" which transforms the instance of one variable into the instance of another variable. The operators can certainly be extended in practice.

Converting Operator : String

∀Z : **Converting Operator**, Z ∈ {"+", "-", "*", "/", "||" "&"}

Assignment : (**{** *opn* : **Operation }**)

Operation : (*convopt* : **Converting Operator**, *convopd1* : **Variable** | **Constant**, *convopd2* : **Variable** | **Constant**) (where *sbghe.vhd.dt* = *convopd1.dt* ∧ *convopd1.dt* = *convopd2.dt* ∧ *converopa* ≠ *"&"*)

Operation : (*convopt* : **Converting Operator**, *convopd* : **Variable** | **Constant**) (where *converopa* = *"&"*)

Local Definition : (*root* : **View**, *inter* : **{***intv* : **View},** *basic*: **{***basv* : **View}**) (where *basv.vtype* = *"basic"* $\wedge$ *inter.vtype* = *"intermediate"*)

4.4.4.5 Validation Rules

In order to formulate meaningful local definitions to construct the data of the IPU correctly, certain constraints have to be applied. The constraints are referred to as *validation rules* which are discussed with respect to the model.

1. For the assignment of ORS and CRS when its converting operator is not "&", all the variables and constants are required to have identical data type.
2. For the assignment of a RRS to an actual relation, the corresponding CQ of the RRS must belong to a basic view definition and the actual relation must belong to the local schema. Each variable of the RRS head must be assigned to an attribute of the relation.
3. For the assignment of a RRS to a view, the corresponding CQ of the RRS must belong to an intermediate view definition and the RRS head must be identical with the view head it is assigned with.
4. For a CQ, the variables in the head of the CQ, the RRS, the ORS and the CRS are denoted as *cqh*, *rrsh*, *orsh* and *crsh* respectively. The variables in the assignment of the ORS and CRS are denoted as *orsa* and *crsa*. Thus ($orsa \in rrsh$) and ($crsa \in rrsh$) and ($cqh \subseteq rrsh \cup orsh \cup crsh$).
5. For a view, all the CQs in its definition must have identical head and the heads must be also identical with the head of the view.
6. For the *dependency* list of a *basic* view, the list must contain exactly one actual relation from the local schema. For the *dependency* list of an *intermediate* view, all the elements in the list must be views.
7. For a local definition, its root view, intermediate views and basic views are denoted as *root*, *inter*, and *basic* respectively. Thus ($root \notin (inter \cup basic)$) and ($root.dependency \subseteq inter \cup basic$) and ($basic \neq \emptyset$)
8. For a local definition, the head of its root view must be identical with the head of the content of the corresponding global definition.

The validation rules have to be complied with during the formulation of the local definitions. This not only helps to build the local definition which constructing data properly for the IPUs where the user query processing can rely on, but also provides the base for checking the validity of the local definitions after the automatic assisted evolution handling has conducted modifications onto the local definitions. It is also required that the actual relations involved in the local definition and the basic views of the local definition must have a bijection mapping relationship. The exact one to one mapping between the involved actual relations and the basic views guarantees the views are in the (negation) normal form thus facilitating the automatic evolution handling. The details will be discussed in the next chapters.

4.4.5 Summary of the IPUD

Once the global definition and the local definition of an IPU are formulated, the IPU is formally constructed. An IPU is modelled as:

IPU : (*iname* : **String**, *gd* : **Global Definition**, *ld* : **Local Definition**)

After a data source is processed by the IPUD algorithm, all the IPUs with respect to the data it exposes are produced. Hence the data source is described through the IPUs. Conceptually, as the IPU specifies the correlation between the data exposed by data sources and the domain ontology, the data sources are virtually integrated into the IDS. After all the participating data sources are processed by the IPUD, The IDS is established. As a consequence, the end users can raise queries based on the domain ontology and the data integration system can answers the queries through accessing the IPUs in the IDS.

4.5 The Information Provision Service Assembly

4.5.1 Information Provision Service

As the SLEDI is following the SaaS approach, each data source is required to share their data through a service which is termed as the Information Provision Service (IPS). Conceptually, an IPS is a data intensive service which follows the general concept of *service* in SaaS. It uses metadata to describe itself thus other components can understand what data the IPS provides and how to interact with the IPS by processing the description. Because the global definition of all the IPUs of a data source unambiguously specifies what data the data source provides, the IPS can accommodate the global definition of the IPUs into its service description metadata. Although in practice a data source may construct more than one IPS and distributes all of its IPUs among them for efficiency, conceptually, a data source provides one IPS which includes the global definition of all its IPUs in its service description metadata. In other words, each participating data source is represented by *exactly one* IPS.

By following the concept of service, the IPSs are implemented at the data source sites and publish their descriptions into the registry for other components to access. When an IPS receives invocation calls, it accesses the local data source to generate the results data based on the local definition of the IPUs and sends the results data back to respond to the invocation calls. From the IPUs' perspective, their global definition is realized through the metadata in the IPS description and their local definition is maintained at the data source site for IPS to utilize. Thus an IPS can be described by the following model:

IPS (*ipsdescription*:(*ipsname* : **String**, {(*ipuname* : **IPU.iname,** *gd* : **IPU.gd)** }), *ls*: $\mathbf{DS_{LS}}$, {(*ipuname* : **IPU.iname,** *ld*: **IPU.ld)**})

4.5.2 Organizational Structure

For the participating data sources in a data integration system, apart from what data they provide, they may also be described by other aspects such as where the data sources are geographically located or which group the data sources belong to. These aspects help to organize all the participating data sources into an *organizational structure*. The structure provides leverage for fine-grained control for the end users hence the end users can specify precisely which of the data sources they are willing to query against. Consequently, instead of searching all the participating data sources for answering every single user query, only the specified data sources are considered which narrows down the search space for query processing.

Which aspects are used for describing the data sources are much depends on the practical situation of the data integration system which may be different from case to case. Take an example from the Mental Health application domain; a clinic provides *primary care service* and locates at *Durham* which further belongs to County Durham. In this example, there are two different classifications involved. One describes the data source of the clinic from the aspect of what type of health service it provides (i.e. Primary care service) and another from where it is located geographical (i.e. Durham). In this research, a generalized data structure is designed for representing the organizational structure which fits into different practical situations. Each data source is represented by its corresponding IPS and each aspect used to describe the data sources is represented by a *classification* which is conceptually modelled as a DAG as illustrated below:

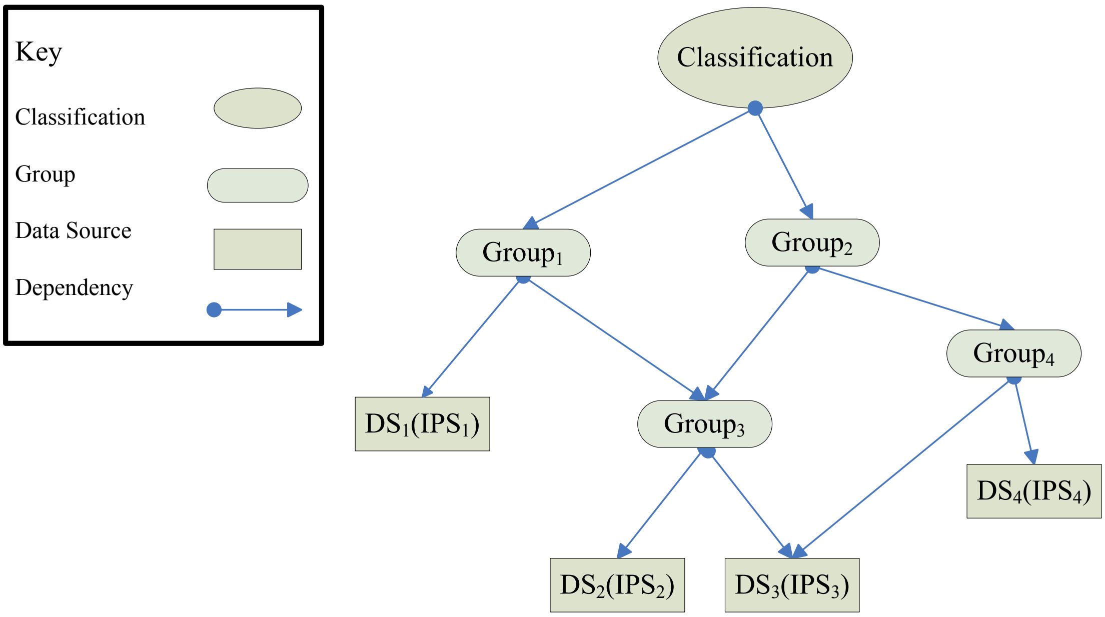


Figure 4-4 Organizational Structure expression

The root represents the classification; other non-leaf nodes represent the virtual groups to describe the different classes specified in the classification and the leaf nodes represent the data sources (IPSs). Each group is described by a name and a list contains all of its direct successors, each of its direct successors may represent a sub-group or a data source belonging to the group. Hence the organizational structure can be represented by the following model:

Organizational Structure : {*clas* : **Classification** }

Classification : (*rootname* : **String**, {*group* : **Group**})

Group : (gname : **String**, {gcontent : **Group** | **IPS**})

Applying the model to the example above, the two aspects correspond to two classifications respectively. From the geographical aspect, the clinic is represented by an IPS which belongs to the *group* “Durham” which further belongs to another *group* “County Durham”, and the *root* of the classification represents it as “geographical classification”; From the mental health service type aspect, the IPS belongs to the group “Primary Care Service”, and the *root* of the classification is described as “mental health service type classification”. An organizational structure of a data integration system may have as many classifications as needed.

4.5.3 Registry

As introduced in section 2.5, the *Registry* is a central component of the software systems which are implemented by employing the SOA. The role the registry plays is to provide a repository to accommodate the description of the services hence facilitating the communications between the services in the system. The *Registry* in the SLEDI follows the general concept of the registry in the service architecture and is extended with extra features to fulfill the research needs. First of all, all the IPSs provided by the participating data sources of the data integration system are required to publish their descriptions in the registry. And then, the application domain ontology and the organizational structure of the data sources are also accommodated in the registry. As a consequence of this, when the queries from end users are processed, by simply accessing the registry, the relevant IPSs can be filtered out by searching the domain ontology and the organizational structure. Then through traversing the description of the IPSs, a service plan can be generated to produce the results data for the queries. The details of query processing in SLEDI will be discussed in the next chapter. The registry is modelled as a triple:

Registry : (*onto* : **Domain Ontology**, *org* : **Organizational Structure**, {*ipsdescription* : **IPS.ipsdescription**})

Through describing the data sources by IPS and aggregating the description of the IPSs, the domain ontology and the organizational structure into the registry, the IDS is formally built and the data from the autonomous data sources are virtually integrated. The IDS is modelled as

IDS : (*reg* : **Registry**, {*ips* : **IPS**})

4.6 Summary

This chapter has presented the process of data sources describing which virtually integrates data from autonomous data sources into the IDS. The IPU mechanism which constitutes its global definition and local definition is introduced in detail. The IPUD algorithm is presented for describing the data sources through the IPU mechanism. The process of IPS assembly is then explained and the formal model of the IPU, IPS and IDS is described.

Chapter 5 introduces the information supplying process of the SLEDI: query processing. Query processing is responsible for answering user queries with results data. The detailed process of rewriting the user queries into resulting queries with respect to the IPUs and constructing the results data through generating and executing service plans are described.

Chapter 5 Query Processing

5.1 Introduction

Chapter 4 presented the first stage of the SLEDI: establishing the IDS through data source describing. The IPUD algorithm was introduced which virtually integrates data from the data sources by describing the data sources through IPU mechanism. The IPUs are then assembled into IPSs which are provided by the data sources at their own sites.

This chapter describes the process of query processing. The process addresses the information needs from end users through answering user queries. Each user query is formulated based on the application domain ontology and the organizational structures of the participating data sources, and is answered through firstly rewriting the query into resulting queries according to the global definition of IPUs, and then the resulting queries are answered through generating and executing service plans with respect to the IPSs. Finally, the results of the service plan executions are combined into the final results data and sent back as the answer of the query to the end users. Although query processing is not the focus of this research, it needs to be described to form the entire structure of the SLEDI.

5.2 Overview of Query Processing

As introduced previously, the data integration problem addressed in this thesis combines data from autonomous and evolving data sources in order to provide the information supplying service to end users. The service is delivered by the process of *query processing* which is an important constituent part of the SLEDI method. Through query processing, end user describe their information needs as queries, then query processing takes one user query at a time as input and produces a result for the query as output to fulfill the information needs. Hence query processing can be formally described as:

Query Processing : User Query → Result

For the purpose of describing user information needs in a data integration system, the user queries are required to be expressed in a formal way. Generally speaking, the formalism employed for representing the user queries much depends on the formalism used for representing the data sources the end user intends to query against. In this thesis, as described in the previous chapter, the IDS have already been successfully built thus the *application domain ontology* and the *organizational structure* provide the base for end users to express their queries. From the perspective of end users, they pose queries directly against the IDS and the query processing of the SLEDI is responsible for processing the queries, accessing the actual data in the data sources and producing the results data to answer the queries.

The detailed process of the query processing in SLEDI can be characterized through three consecutive sub-processes: *Data Source Filtering*, *Query Rewriting* and *Result Generating*, with each sub-process involving one or more elementary steps. The sub-processes and the steps are illustrated and briefly explained below:

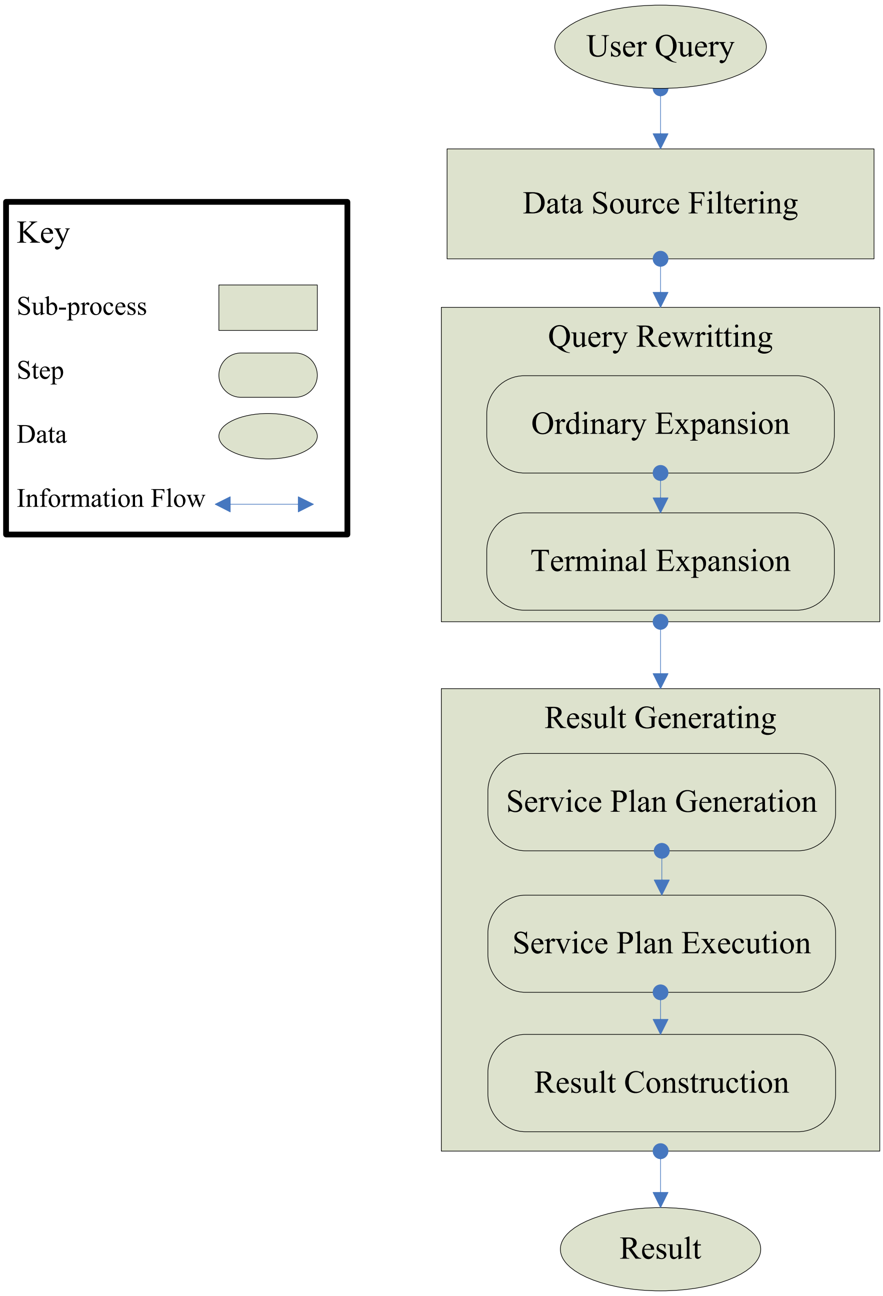


Figure 5-1 Query Processing Mechanism

1. Data Source Filtering: this sub-process accepts the user query and parses the query preliminarily. As there may be many data sources involved in the data integration system

and a user query may only involve querying against a subset of all the data sources, this sub-process is responsible for filtering out the targeting data sources the user query is interested in by analyzing the information of the *targeting data source set* specified in the user query. As the consequence, the output of this sub-process is a *set* containing all the targeting data sources and the reformed query in which the elements of the query only refer to the application domain ontology.

2. Query Rewriting: this sub-process takes the targeting data source set and the reformed query from the previous sub-process as its input. It rewrites the query into resulting queries as its output. As introduced in the previous chapter, each data source exposes its data through IPUs and each IPU has a global definition. The global definition can be considered as a *virtual view* defined over the application domain ontology. This sub-process is responsible for rewriting the query expressed with respect to the application domain ontology into the resulting queries with respect to the global definition of the IPUs. The technique of *answering query using views* is employed to find the rewriting (i.e. the resulting queries) of the input query. In other words, the input query can be answered by using the answers of the resulting queries. Because the global definition is formed by mainly following the GAV approach, this sub-process is achieved by two consecutive elementary steps:
   - Ordinary Expansion: this step expands the input query by accessing the *Rule* component of the application domain ontology. If an element of the query is the *head* of a *rule*, it is substituted by the body of the rule. The substitution is carried out recursively as a standard backward chaining unfolding until all the elements of the query are traversed. As the result, the output of this step is the ordinarily expanded query where all the elements are either a *concept* or a *role* defined in the *Terminology* of the application domain ontology.
   - Terminal Expansion: this step takes the ordinarily expanded query from the previous step as its input for continuing expanding. All the elements of the query are traversed and expanded by considering the Terminology of the application domain ontology and the global definition of the IPUs provided by the data sources in the targeting data source set. As a result, the terminally expanded query is obtained in which every element of the query is the global definition of an IPU. The terminal expansion represents all the ways of deriving the answers of the original user query from the *views*. Thus the user query is rewritten into a set of resulting queries with respect to the IPUs.
3. Result Generating: this sub-process takes the terminal expansion of the user query from the previous sub-process as its input and generates the final result for the query as its output. After the query rewriting is accomplished, finding the result for the original user query is transformed into finding the result for the resulting queries in the terminal expansion. And the result of each resulting query can be obtained from the IPUs in the query. As introduced

in the previous chapter, the IPUs are accommodated through the IPS mechanism. The acquisition of the results data of the IPUs can be realized through the service invocation calls against the corresponding IPSs. As the consequence, the final result can be constructed by composing the results data sent back from all the service invocation calls. This sub-process can be characterized by three consecutive elementary steps: *Service Plan Generation*, *Service Plan Execution* and *Result Construction*.

- Service Plan Generation: this step takes the terminal expansion as input. For each IPU in a resulting query in the terminal expansion, a service invocation call is generated regarding to the IPS which accommodates this IPU. After all the IPUs in a resulting query are traversed, the generated invocation calls can be used to find the results data for the resulting query hence produce a *partial service plan* for the original user query. This procedure is carried out iteratively until all the resulting queries in the terminal expansion are processed. As the result, all the partial service plans generated constitute the *Service Plan* for answering the original user query.
- Service Plan Execution: this step takes the Service Plan produced from the previous step for execution. For each service invocation call in the plan, the invocation is dispatched to the relevant data source site which hosts the IPS the invocation refers to. Then the IPS is responsible for producing the *results data* to respond the service invocation call. As introduced in the previous chapter, the results data can be produced based on the local definition of the IPUs of the IPS. Because a local definition is represented as a *view* over the local schema of the data source, the *local query* can be formulated to produce the results data. As all the data sources involved in this research are assumed to employ the relational data model, the results data is in fact in the form of a relation. Consequently, this step executes the Service Plan and generates the results data for every invocation call in the plan.
- Result Construction: this step gathers the results data from all the invocation calls of the plan and composes them to produce the *Final Result* to answer the original user query. Because the results data of an invocation call is a relation, the results data of different invocation calls can be combined by applying relational operations. After all the results data of the invocation calls of a partial service plan are combined together, the results data of the partial service plan is acquired. This procedure is then performed iteratively until the results data of all the partial service plans are attained. Consequently, the Final Result of the Service Plan can be constructed in a similar fashion. Finally, the Final Result is sent back to the end user as the answer of the original user query to meet the information needs.

The elementary steps, sub-process, the formalisms and operations involved in query processing are elucidated in detail in the following sections.

5.3 Data Source Filtering

As introduced in the previous section, a user query may specify a certain set of data sources the query is targeted at. The data source filtering sub-process intends to exclude the data sources which are certainly not interested in the query. Thus these data sources will not be considered in the subsequent sub-processes. As a result, only a subset of all the data sources is selected and brought to the following sub-processes for further processing. Hence the search space is narrowed down and the efficiency of the query processing may be enhanced.

5.3.1 User Query

In this research, since the application domain ontology and the organizational structure provide the base for end users to express their information needs, the formalism for expressing the user query constitutes two parts: The first part provides the interface so that the user can specify the data sources the query is interested in by exploiting the organizational structure. And the second part is for the user to describe the information they require by utilizing the application domain ontology. Through formally denoting the first part as **Q(org)** and the second part as **Q(onto)**, the user query can be formally denoted as:
User Query : (Q(org), Q(onto))

As introduced in section 4.5.2, the participating data sources of the data integration system are classified by the organizational structure which is realized through a set of *classifications*. Each classification describes the data sources from an aspect. A classification is modelled as a DAG with the data sources being denoted as leaf nodes and the virtual *groups* being denoted as non-leaf nodes. Hence through stipulating the name of classification and groups, end users can specify the data sources they are willing to query against. In this thesis, the *targeting data sources set* specified by the **Q(org)** is expressed through a set of *atomic targets*. An *atomic target* specifies the data sources through a pair of names where the first name indicates the classification and the second name indicates the virtual group. And the *targeting data sources set* is formed through the union of the atomic targets. As a result, the **Q(org)** is formally denoted as:
Q(org) : { *atar***: Atomic Target }**
Atomic Target : (*Classification root name***: String,** *Group name***: String)**
This model provides the flexibility which allows the end users to apply fine-grained control for specifying the targeting data sources for each user query.

As introduced in section 4.3.2, the elements which constitute the application domain ontology are *concepts*, *roles* and *rules* which are represented as unary, binary and n-ary relations respectively (all the relations are also referred to as *domain relations*). These *relations* provide the basic materials that end users can manipulate to describe their information needs through the *query content* (i.e. **Q(onto)**). Thus the *conjunctive query* is chosen as the formalism for expressing the **Q(onto)**. Because the global definition of an IPU can be considered as a *view* over the same relations, the technique of *answering queries using views* can be applied to find the answer of **Q(onto)** which will be discussed in detail in the next section.

In this thesis, **Q(onto)** is formed as a union of conjunctive queries over the application domain ontology which is in the form of: $\mathbf{Q(\overline{x}) :- \cup_{i\in[1..k]} P_1^{i}(\overline{x}_1,\ \overline{y}_1)\wedge\ldots\wedge P_m^{i}(\overline{x}_m,\ \overline{y}_m)}$ where the P_j^i's are domain relations. The variables of $\overline{x} = \overline{x}_1 \cup \ldots \cup \overline{x}_m$ are *distinguished variables*; the variables of $\overline{y} \subseteq \overline{y}_1 \cup \ldots \cup \overline{y}_m$ are *existential variables*. The distinguished variables represent the instance data that users are interested in knowing and the existential variables are used to constrain the distinguished variables. It is assumed without loss of generality that inequalities $x \neq y$ are implicit for every pair of distinct variables that appear in the query.

Classically, a query is interpreted relatively to a database *db* containing a finite set of stored data. The data represents information instances of the application domain ontology *onto* which consists of the terminology *T* and rules *R*. Given a model *I* of *db* and *onto*, a conjunctive query $\mathbf{Q_i(\overline{x}) :- P_1^{i}(\overline{x}_1,\ \overline{y}_1)\wedge\ldots\wedge P_m^{i}(\overline{x}_m,\ \overline{y}_m)}$ over *db* and *onto* is interpreted as the set Q_i^I of tuples $\overline{o}$ made of elements of the interpretation domain Δ^I such that when substituting $\overline{o}$ for $\overline{x}$, the formula $\mathbf{\exists\overline{y}\ P_1^{i}(\overline{o}_1,\ \overline{y}_1)\wedge\ldots\wedge P_m^{i}(\overline{o}_m,\ \overline{y}_m)}$ evaluate to be true in *I*. (distinct variables are mapped to distinct elements). A union of conjunctive queries $\mathbf{Q(\overline{x})}$ is interpreted as the set Q^I of tuples $\overline{o}$ made of elements of Δ^I such that the formula $\mathbf{Q(\overline{o})}$ evaluates to be true in *I*. In other words, the answer of a query $\mathbf{Q(\overline{x})}$ over a database *db* and *onto* is the set of tuples $\overline{a}$ such that:
db*, *onto $\mathbf{\models \cup_{i\in[1..k]} \exists\overline{y}P_1^{i}(\overline{a}_1,\ \overline{y}_1)\wedge\ldots\wedge P_m^{i}(\overline{a}_m,\ \overline{y}_m)}$

The **Q(onto)** is formally denoted as:
Q(onto) : {*q***: Query**}
Query : (*qhead***: Head**, {*cj***: Head** }) (where *cj.hdn* ⊆ (*onto.ter.con.nconcept* ∪ *onto.ter.rol.nrole* ∪ *onto.rule.sta.cones.nconse*))

5.3.2 Filtering

Through the **Q(org)**, the targeting data sources the user query aims to interrogate are explicitly specified. Hence these data sources can be determined by parsing the **Q(org)** and examining it against the organizational structure. The sub-process of the data source filtering takes the responsibility of identifying the data sources and organizing them as a *set* (i.e. targeting data source set). Because each participating data source is represented by an **IPS** (see section 4.5.1), the elements of the set are IPSs. After the sub-process is accomplished, the **Q(org)** is no longer needed in the following sub-processes of the query processing thus can be eliminated from the user query. As a result, the sub-process of data source filtering can be formally denoted as:

Data Source Filtering : User Query → (Targeting Data Source Set, Q(onto))

Targeting Data Source Set : { *ips* **: IPS }**

The targeting data source set can be constructed through determining the indicated data sources of the atomic targets. An atomic target can be calculated by identifying all the data sources (directly and indirectly) belonging to the group specified by it. After all the atomic targets are calculated, the targeting data source set can be generated based on the results. It is worth mentioning that user query may leave its **Q(org)** part empty to indicate that all the participating data sources are required for the query. Under this circumstance, all the data sources will be selected. The following steps are taken for constructing the targeting data source set:

1. Let ***O*** be an empty set representing the targeting data source set
2. Select all the *atomic target*s of the **Q(org)** and place them in a set ***T***
3. Select all the *classifications* of the *organizational structure* and place them in a set ***Org***
4. If ***T*** = Ø, for every element of ***Org***, traverse all the *group*s of the element recursively to find all the IPSs. Store the IPSs into ***O*** and remove duplicates.
5. If ***T*** ≠ Ø, for every element of ***T***
    - i. Let *atr* be the current element of ***T***
    - ii. Attempt to find the element *cla* in ***Org*** whose *root name* is identical with the *Classification root name* of *atr*.
    - iii. If *cla* is found, attempt to find the group *gr* whose *gname* is identical with the *Group name* of *atr*.
    - iv. If *gr* is found, traverse all the *gcontent* of *gr* recursively to find all the IPSs. Store the IPSs into ***O*** and remove duplicates.
    - v. Repeat from i for the next element.

An example used here is to illustrate how the sub-process of data source filtering works. Let us assume there are four participating data sources organized by their geographical locations, the organizational structure of which is depicted in the following figure:

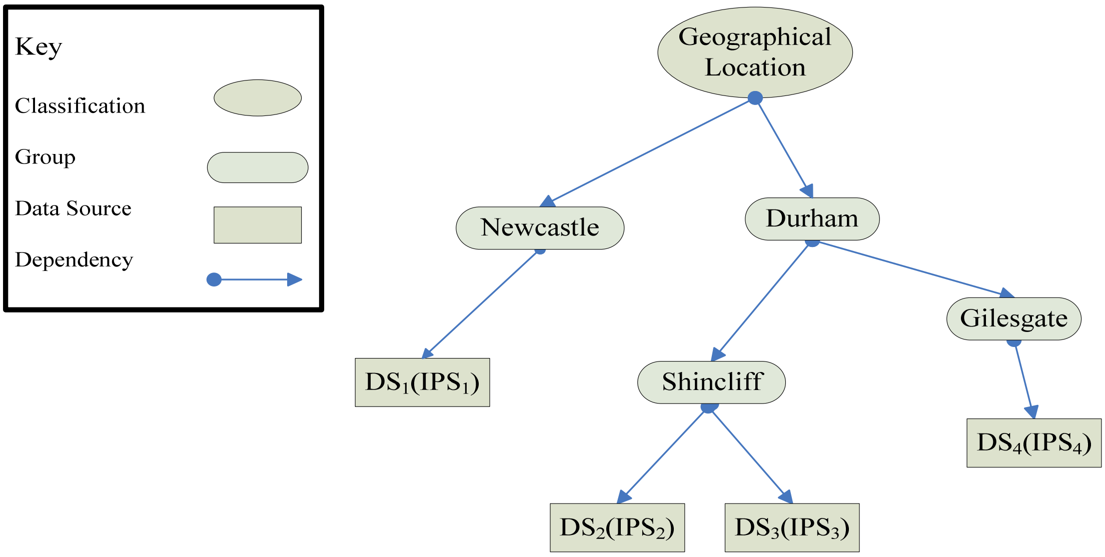


Figure 5-2 Sample Organizational Structure

A user query intends to query all the data sources located in Durham area specifies its **Q(org)** as (*Geographical Location*, *Durham*), the *targeting data sources set* can then be constructed through firstly determining all the groups belong to *Durham* (i.e. *Shincliff* and *Gilesgate*) and then finding out all the data sources belong to groups *Shincliff* (i.e. DS_2 and DS_3) and *Gilesgate* (i.e. DS_4). Although the example is simple (only have one atomic target in **Q(org)** and one classification in the organizational structure), it demonstrated the rational of how data source filtering sub-process works. In practice, user query may specify any number of atomic targets in its **Q(org)** and the organizational structure may include more classifications.

5.4 Query Rewriting

Essentially, the IDS is a database which stores a finite set of information instances of the application domain ontology. Finding the answer to a user query is in fact finding all the information instances stored in the IDS so that when substituting the distinguished variables of the user query with the information instances, the user query evaluates to be true. Since the IDS is virtual, the information instances stored in the IDS are not directly available, in fact they are stored in the data sources and abstractly represented by the IPUs. Hence the answer can be produced by using the information instances the IPUs provide. Because the global definitions of the IPUs are views over application domain ontology, the answer to the user query can be obtained by rewriting the user query into resulting queries that only refer to the IPUs. The technique of answering query using views is employed for the query rewriting.

5.4.1 Answering queries using views

The problem of answering queries using views can be informally described as given a query *Q* over a database schema and a set of views $V_1,\ldots V_n$ over the same schema, then find the answer of *Q* by only using the views. The problem can be tackled through finding the query *Q'* so that *Q'* only refers to the views and the answer of *Q'* can be used to answer *Q*. The *Q'* is termed as *rewriting* of *Q*. For the purpose of providing a semantic basis to enable the comparison between queries and their rewriting, the concepts of *query containment* and *query equivalence* are introduced: [37, 60]

- Query containment: A query *Q'* is considered to be *contained* in a query *Q* if for all database instances *D*, the computed results data for *Q'*, denoted as *Q'(D)* is a subset of the computed results data for *Q*, denoted as *Q(D)*. i.e. $Q'(D) \subseteq Q(D)$
- Query equivalence: A query *Q'* is considered to be equivalent to a query *Q* if they are mutually contained in each other. i.e. $Q'(D) \subseteq Q(D) \wedge Q'(D) \subseteq Q(D)$

Based on the above definition, the *equivalent rewriting* and *Maximally-contained rewriting* can be distinguished. Let *Q* be a query and $V=\{V_1,\ldots V_n\}$ be a set of view definitions. The *Q'* is considered as an equivalent rewriting of the query *Q* using V if

- *Q'* refers only to the views in *V* and
- *Q'* is equivalent to *Q*

In the context of data integration, the Maximally-contained rewriting is also considered. Unlike the equivalent rewriting, the Maximally-contained rewriting may differ with respect to different query languages employed in the context. Let *Q* be a query, $V=\{V_1,\ldots V_n\}$ be a set of view definitions and *L* be a query language. The *Q'* is considered as a Maximally-contained rewriting of the query *Q* using *V* with respect to *L* if

- *Q'* is a query in *L* that refers only to the views in *V* and
- *Q'* is contained in *Q* and
- There is no rewriting $Q'' \in L$, such that $Q' \subseteq Q'' \subseteq Q$ and *Q''* is not equivalent to *Q'*

5.4.2 Completeness and Complexity of finding query rewritings

Among many algorithms which have been developed for finding query rewriting using views with respect to different query languages, some problems cut across all of the algorithms from a more theoretical perspective such as *completeness* and *complexity* [37]. The completeness that a query rewriting algorithm concerns is, given a set of views *V* and a query *Q*, will the algorithm always find a rewriting *Q'* using *V* if one exists?

The completeness is characterized with respect to the specific query language in which the rewritings are expressed. In some cases, the limitation on the expressiveness of the query language (e.g. no union is allowed in the rewritings) may result in no equivalent rewriting being able to be found. For the purpose of extracting all the certain answers for the query from the views, the Maximally-contained rewriting comes up as the alternative. In fact, finding the Maximally-contained rewriting also depends on the specific query language employed as the maximal containment is defined with respect to the query language. For example, in some cases, Maximally-contained rewriting can only be found if the recursive datalog rewriting is considered [37, 34].

Another related issue is characterizing the complexity of the query rewriting algorithms. The complexity can be discussed under the specific setting of the query languages the algorithms apply. Although the complexity may be vary with respect to different query languages, in general, it is in NP as it is sufficient to guess a rewriting Q' and check its correctness (i.e. whether Q' is contained by Q) [37]

5.4.3 The query rewriting algorithm

As introduced previously, the application domain ontology in SLEDI is realized through the ALN description logic and non recursive function-free horn rules. The IPU is mapped to the application domain ontology by mainly following the GAV approach and its global definition is formulated as either a concept or a role from the application domain ontology. The user query is in the form of unions of conjunctive query over domain relations. Hence the query rewriting algorithm used in the PICSEL system [34] is employed in SLEDI. The algorithm is briefly introduced below:

Through expanding the user query in terms of the global definition of the IPUs, the algorithm computes a representative set of all the possible rewritings of the original user query with respect to the available views. Apparently, when the user query is a single concept and the global definitions are all mapped with concepts, the query rewriting can be reduced to subsumption checking. The algorithm constitutes the steps of *ordinary expanding*, *terminal expanding* and *rewritings verifying*.

5.4.3.1 Ordinary Expanding

Ordinary expanding takes the targeting data source set and **Q(onto)** as input and expands **Q(onto)** preliminarily. For the convenience of describing the algorithm, for a subquery $\mathbf{Q_i(\overline{x}) :- P_1^i(\overline{x}_1, \overline{y}_1) \wedge \ldots \wedge P_m^i(\overline{x}_m, \overline{y}_m)}$, the atomic conjunct $\mathbf{P_i^i(\overline{x}_1, \overline{y}_1)}$ is referred to as *concept-relation*, *role-relation* and *rule-relation* if it is a concept, role and rule from the application domain ontology respectively. The ordinary expanding of a query unfolds all the rule-relations in the query according to the Rule component in the application domain ontology.

Let $\mathbf{P(\overline{x})}$ be a rule-relation; it is expanded by iteratively unfolding the rules whose consequent is in the form $\mathbf{P(\overline{x}') \text{ :- } P_1 (\overline{x}_1', \overline{y}_1') \wedge \ldots \wedge P_k(\overline{x}_k', \overline{y}_k')}$. Let α be the most general unifier of $\mathbf{P(\overline{x})}$ and $\mathbf{P(\overline{x}')}$, extend such that every variable $\mathbf{Y_i}$' is assigned to a fresh variable that appears nowhere else. Then the $\mathbf{P(\overline{x})}$ is replaced by $\mathbf{P_1(} \alpha \mathbf{(\overline{x}_1'),} \alpha \mathbf{(\overline{y}_1') \wedge \ldots \wedge P_k(} \alpha \mathbf{(\overline{x}_k'),} \alpha \mathbf{(\overline{y}_k')}$. A step of expansion of the $\mathbf{P(\overline{x})}$ results in the set of rewritings obtained by unfolding all the rules whose consequent is unifiable with $\mathbf{P(\overline{x})}$.

The expansion is applied iteratively, since the rules are non recursive. After a finite number of steps, the *ordinary expansion* is obtained which is a set of conjunctions with every atomic conjunct either being a concept-relation or a role-relation (see section 4.3.2.2). This results from the soundness and completeness of the backward chaining algorithm for non recursive function-free horn rules that the ordinary expansion characterizes all the ways of deriving the original query.

5.4.3.2 Terminal Expanding

Terminal expanding takes the targeting data source set and ordinary expansion as its input and further expands the rewritings with respect to the global definition of the IPUs in the targeting data source set. For each rewriting $\mathbf{P_1^i(\overline{x}_1, \overline{y}_1) \wedge \ldots \wedge P_m^i(\overline{x}_m, \overline{y}_m)}$ in the ordinary expansion, if there exists $\mathbf{P_k^i(z, z')}$, **z'** is said to be a *direct successor* of **z** and the transitive closure of the direct successor is defined as *successor*. If using *nodes* represents variables and an *arc* links a variable to any of its direct successors, the distinguished variable X is said to be the root of a tree structure if each node in the tree has no distinguished successor. Then the X can be expanded by its *descriptive support* hence producing the *compact expansion*. For example, $\mathbf{x_1}$ has a tree structure in the rewriting $\mathbf{P(x_1) \wedge P_1(x_1,y_1) \wedge P_2(y_1,y_2) \wedge P_3(y_2)}$, the conjunction can be replaced by its descriptive support $\mathbf{[P \sqcap (\geq 1 P_1) \sqcap \forall P_1.(\geq 1 P_2) \sqcap \forall P_1. \forall P_1.P_3)] \; (x_1)}$.

After all the rewritings in the ordinary expansion are expanded inductively, the set of conjunctions can be brought for final expanding also referred to as *Grounding*. The atomic conjuncts in the rewritings are grounded to the concept-relations and the role-relations that can be obtained from the global definitions of the IPUs in the targeting data source set. For example, let $\mathbf{V(\overline{x})}$ be the content of the global definition of an IPU (see section 4.3.3), a concept-relation $\mathbf{P(z)}$ can be grounded if

- Either exists $\mathbf{V(x) \text{ :- } P_1(x)}$ or $\mathbf{V \sqsubseteq P_1}$ such that **P** subsumes $\mathbf{P_1}$, then $\mathbf{P(z)}$ is extended to $\mathbf{V(z)}$
- There exists $\mathbf{V_1(x) \text{ :- } [\forall P_1.P](u)}$ and $\mathbf{V_2(x,y) \text{ :- } P_1(u,z)}$, then $\mathbf{P(z)}$ is extended to $\mathbf{V_1(x) \wedge V_2(x,z)}$

A role-relation $\mathbf{P(z_1,z_2)}$ can be expanded if

There exists **$V(x,y)$:- $P(x,y)$**, then **$P(z_1,z_2)$** is extended to **$V(z_1,z_2)$**

Finally, the *terminal expansion* is obtained which consists of a set of rewritings. Each rewriting is in the form of conjunctive query where the head is identical with the original user query, and each atomic conjunct in a rewriting is either a global definition of an IPU or a *domain relation* which cannot be logically derived from the IPUs. Those domain relations are denoted as **Remainder($\bar{x}$, $\bar{y}$)**.

5.4.3.3 Rewritings Verifying

The final stage of the algorithm is Rewritings Verifying. It verifies each rewriting in the terminal expansion through two phases. Since each global definition has an optional constraint part (see section 4.4.3), each rewriting is checked against the constraint part to examine whether it is compatible with the constraints. As a result, the incompatible rewritings are eliminated from the terminal expansion in the first phase. It is easy to see that rewritings consisting of only global definitions provide the answers for the original user query. However, the rewritings consisting of remainder part may still entail the original query. Thus the second phase checks each of the rewritings to analyse whether they actually entail the original user query. If they do, the remainder part is deleted and the rest part of the conjunction is kept as a rewriting in the terminal expansion. Each rewriting is also referred to as a *resulting query*.

The algorithm is *complete* in the sense that the resulting queries completely characterize the rewritings of the original query. The complexity, in the worst case, is exponential in terms of maximum unfolding depth of the rule-relations and the maximum size of concept expressions in the query or application domain ontology. Although the query rewriting algorithm is an important constituent part of the SLEDI system, it is not the focus of this research. Due to the limited space of this thesis, only a brief introduction of the algorithm is provided as above, the details of the formal definition and explanation can be found in [34, 59].

5.4.3.4 A Simple Example

A simple example from the mental health application domain is used to illustrate the PICSEL query rewriting algorithm. The focus of this research is on solving evolution problems in the data integration, and the query rewriting algorithm is borrowed just for completing the data integration system. The example does not intend to show all the reasoning details of the algorithm. Though simple, the example demonstrates some subtle points of how the algorithm is applied. Assume the application domain ontology contains the following part:

Terminology: {…

Adult $\sqsubseteq$ Classification;   Child $\sqsubseteq$ Classification;   Adult $\sqcap$ Child $\sqsubseteq$ $\perp$

AdultPatient **:=** Patient ⊓ ∀HasClassification.*Adult*; ChildPatient **:=** Patient ⊓ ∀HasClassification.*Child* Doctor ⊑ Category; SocialWorker ⊑ Category ; Doctor ∩ SocialWorker ⊑ ⊥ DoctorStaff **:=** Staff ⊓ ∀HasCategory.*Doctor*; SocialWorkerStaff **:=** Staff ⊓ ∀HasCategory.*SocialWorker* …}

Rule: {… ;
Patient-Info(X_PatientID, X_Name, X_Gender, X_Age, X_ Classification) **:-** Patient(X)∧HasPatientID(X,X_PatientID)∧HasName(X, X_Name)∧HasGender(X, X_Gender)∧HasAge(X, X_Age)∧HasClassification(X, X_Classification)
…}

Table 5-1 application domain ontology

The *Terminology* declares that a *Patient* can be either an Adult Patient or a Child Patient. *Staff* can be either a Doctor or a Social Worker. The *Rule* defines an ordinary relation including the information of patient ID, name, gender, age and classification (i.e. child or adult) of a patient.

A user query is raised in the following way to ask for the name, gender and age of the patients who have been assigned to a social worker:

Q(X_Name, X_Gender, X_Age) **:-**

Patient-Info(X_PatientID, X_Name, X_Gender, X_Age, X_ Classification) ∧ Staff(Y) ∧ HasAssignedPatientID (Y, X_PatientID) ∧ HasCategory(Y, Y_Category) ∧SocialWorker(Y_Category)

Firstly in the query rewriting, the ordinary expansion can be obtained through substituting the *Rule* for its *definition* in the application domain ontology:

Patient(X)∧HasPatientID(X,X_PatientID)∧HasName(X,X_Name)∧HasGender(X,X_Gender)∧HasAge(X,X_Age)∧HasClassification(X,X_Classification)∧*Staff(Y) ∧HasAssignedPatientID(Y,X_PatientID)∧HasCategory(Y,Y_Category)∧SocialWorker(Y_Category)*

It is worth mentioning here that the existential variables such as *X* are added into the body of the query during the expansion. In fact, any fresh variable can be used here.

Since the *Y_Category* is an existential variable, the *Staff(Y)∧ HasCategory(Y, Y_Category)∧ SocialWorker(Y_Category)* can be substituted for its descriptive support: *SocialWorkerStaff(Y)*. Then the user query is further expanded as:

Patient(X)∧HasPatientID(X, X_PatientID)∧HasName(X, X_Name)∧HasGender(X, *X_Gender)∧HasAge(X,X_Age)∧HasClassification(X,X_Classification)∧**SocialWorkerStaff(Y)** ∧HasAssignedPatientID(Y,X_PatientID)*

Assume there are three data sources **DS$_1$**, **DS$_2$** and **DS$_3$** and that each data source provides a set of IPUs. The global definitions of the IPUs are in the following table:

DS$_1$: **Content**:{ V_{11}(X):- ChildPatient(X), V_{12}(X,Y):- HasPatientID(X,Y), V_{13}(X,Y):-HasName(X,Y), V_{14} (X,Y):- HasGender(X,Y), V_{15} (X,Y):- HasAge(X,Y), V_{16} (X,Y):- HasClassification(X,Y)}
Constraint:{ $V_{12}(X,Y)\wedge\neg V_{11}(X) \sqsubseteq \bot$, $V_{13}(X,Y)\wedge\neg V_{11}(X) \sqsubseteq \bot$, $V_{14}(X,Y)\wedge\neg V_{11}(X) \sqsubseteq \bot$, $V_{15}(X,Y)\wedge\neg V_{11}(X) \sqsubseteq \bot$, $V_{16}(X,Y)\wedge\neg V_{11}(X) \sqsubseteq \bot$ }

DS$_2$:**Content**:{V_{21}(X):-SociaworkerStaff(X),V_{22}(X,Y):-HasAssignedPatientID(X,Y)}
Constraint:{ $V_{22}(X,Y)\wedge\neg V_{21}(X) \sqsubseteq \bot$}

DS$_3$: **Content**:{ V_{31}(X):- Patient(X), V_{32}(X,Y):- HasPatientID(X,Y), V_{33}(X,Y):-HasName(X,Y), V_{34} (X,Y):- HasGender(X,Y), V_{35} (X,Y):- HasAge(X,Y), V_{36} (X,Y):- HasClassification(X,Y)}
Constraint:{ $V_{31}(X) \sqsubseteq (\forall$ HasClassification.*Adult*), $V_{32}(X,Y)\wedge\neg V_{31}(X) \sqsubseteq \bot$, $V_{33}(X,Y)\wedge\neg V_{31}(X) \sqsubseteq \bot$, $V_{34}(X,Y)\wedge\neg V_{31}(X) \sqsubseteq \bot$, $V_{35}(X,Y)\wedge\neg V_{31}(X) \sqsubseteq \bot$, $V_{36}(X,Y)\wedge\neg V_{31}(X) \sqsubseteq \bot$ }

Table 5-2 the global definitions of the IPUs

The final expansion can then be obtained by *grounding* the atomic conjuncts in the query based on the global definitions. Some grounding is obvious. For example, the atomic conjunct *Patient(X)* can be grounded to V_{31}(X) as the global definition shows that V_{31}(X):- Patient(X); the atomic conjunct *HasName(X, X_Name)* can be ground to V_{13}(X,Y) or V_{33}(X,Y) and the variable Y can be substituted for *X_Name*. However, some grounding may not be straightforward. For example, the atomic conjunct *Patient(X)* can also be grounded to V_{11}(X) as it can be deduced from the global definition and the application domain ontology that since V_{11}(X):- ChildPatient(X) and ChildPatient $\sqsubseteq$ Patient, then $V_{11}(X) \sqsubseteq$ Patient(X). After the grounding, the verification step then takes place based on the constraint of the global definitions hence the final expansion of the user query is obtained as following:

1) V_{31}(X) ∧ V_{32}(X, *X_PatientID*) ∧ V_{33}(X, *X_Name*) ∧ V_{34}(X, *X_Gender*) ∧ V_{35}(X, *X_Age*) ∧ V_{36}(X, *X_Classification*) ∧ V_{21}(Y) ∧ V_{22}(Y, *X_PatientID*)

2) V_{11}(X) ∧ V_{12}(X, *X_PatientID*) ∧ V_{13}(X, *X_Name*) ∧ V_{14}(X, *X_Gender*) ∧ V_{15}(X, *X_Age*) ∧ V_{16}(X, *X_Classification*) ∧ V_{21}(Y) ∧ V_{22}(Y, *X_PatientID*)

Finally, the user query is rewritten into the union of the resulting queries **1)** and **2)**. The resulting query **1)** combines the IPUs from **DS$_1$** and **DS$_2$** while the resulting query **2)** combines the IPUs from **DS$_2$** and **DS$_3$**.

5.5 Result Generating

After the query rewriting sub-process is accomplished, the original user query is reformulated into a set of resulting queries. The union of the answers of the resulting queries provides the set of answers which can be obtained from the available data sources for answering the original user query. Since the resulting queries are in the form of conjunctions where each atomic conjunct is the global definition of an IPU, the answer of a resulting query can be acquired through combining the results data extracted from the IPUs in the query. As mentioned previously, each IPU is accommodated by an IPS. Extracting results data from an IPU can be achieved through sending a service invocation call to the corresponding IPS. In SLEDI, the result generating sub-process takes the responsibility of producing the results data. It takes the terminal expansion from the query writing as input and processes each resulting query sequentially. For each resulting query, it sets up a partial service plan through generating service invocation calls from the atomic conjuncts. Then all the partial service plans constitute the service plan. Through executing the service plan, the results data can be finally composed to answer the original user query. Hence the sub-process can be formally denoted as:

Result Generating : Terminal Expansion → Result

There are three elementary steps involved: *Service Plan Generating*, *Service Plan Executing* and *Result Constructing*.

5.5.1 Service Plan Generating

This step takes the terminal expansion from the query rewriting as its input. As described previously, each resulting query in the terminal expansion is a conjunction of the global definition of IPUs. Abstractly, the resulting query is in the form of $\mathbf{Q_i(\overline{x}){:}{-}V_1^{\,i}(\overline{x}_1) \wedge \ldots \wedge V_m^{\,i}(\overline{x}_m)}$ where the $\mathbf{V_i^{\,i}(\overline{x}_i)}$ represents the global definition of an IPU. In SLEDI, the global definition of an IPU is realized through the model of **IPU.gd** and is accommodated by an IPS through pairing with the name of the IPU (**IPU.iname**) as a constituent component of the IPS (see section 4.5). Thus an atomic conjunct in the resulting query is realized as a *triple* which consists of the IPS name, IPU name and a variable list describing $\mathbf{\overline{x}_i}$. The terminal expansion and resulting query are actualized through the following model:

Terminal Expansion : (*qhead* **: Head, {***rq* **: Resulting Query})**

Resulting Query : (*rqhead* **: Head**, {*ipsname* : **IPS.ipsname**, *ipuname* **: IPU.iname, {***var* **: Variable}})**

For each atomic conjunct in a resulting query, since the triple explicitly indicates which IPU provides the required results data and which IPS accommodates the IPU, a service invocation call can be generated in a straightforward manner to obtain the results data for the atomic conjunct.

Thus the model of the service invocation call is:

Service Invocation Call : (*ipsname* : **IPS.ipsname**, *ipuname* **: IPU.iname, {***var* **: Variable})**

For each resulting query, the action of service invocation call generating is applied iteratively until all the atomic conjuncts in the resulting query are processed. As a result, a set of invocation calls are produced. These service invocation calls comprise the partial service plan which can be used to acquire the results data for the resulting query. Thus the model of a partial service plan is:

Partial Service Plan : **(***rqhead* **: Head**, {*sic*: **Service Invocation Call**}**)**

After all the resulting queries of the terminal expansion are processed, the set of partial service plans produced constitutes the service plan. Then the service plan is outputted by the step of service plan generating which can be brought to the following steps for further processing. The model of a service plan is:

Service Plan : (*qhead* **: Head**, {*psp* **: Partial Service Plan**}**)**

5.5.2 Service Plan Executing

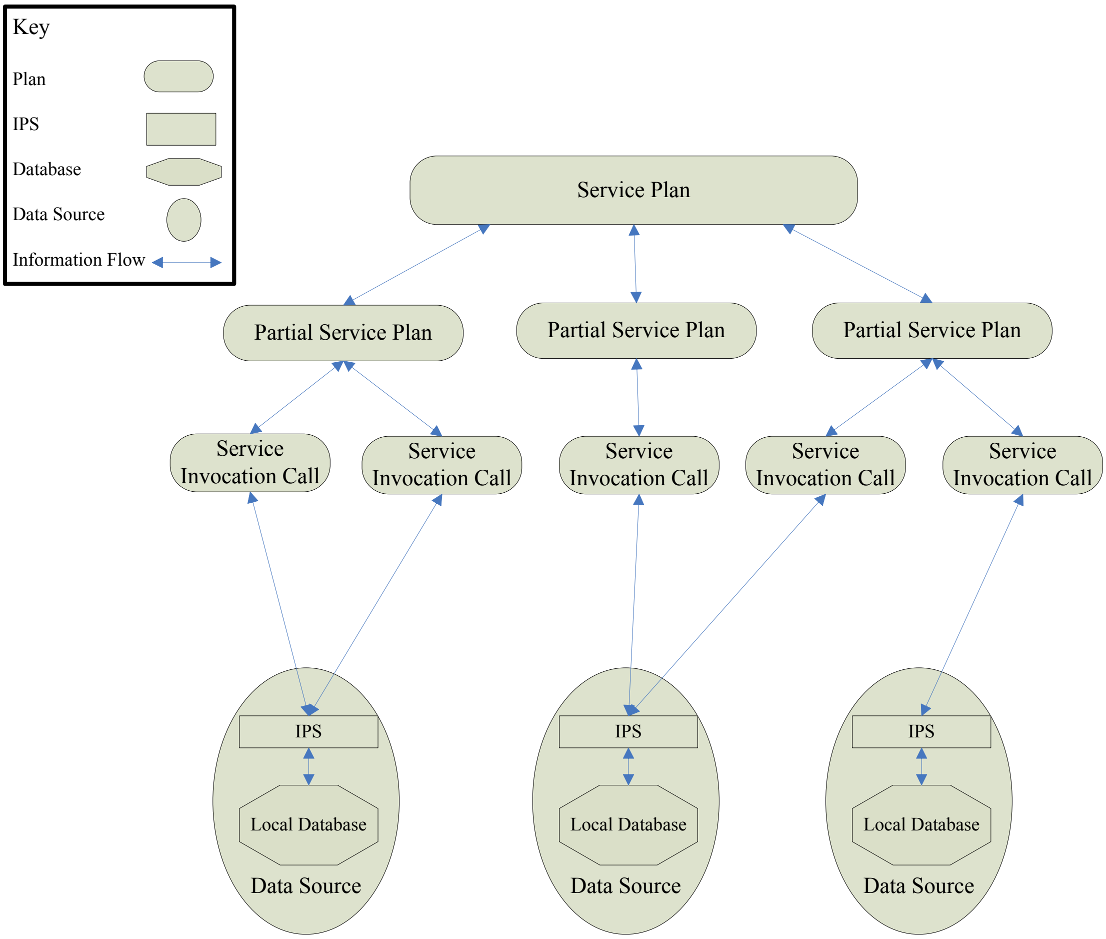


Figure 5-3 Service Plan Executing Process

The process of the service plan executing step is illustrated in figure 5-3. This step takes the service plan as input and carries out the plan through executing each of the partial service plans in it. A partial service plan is executed by sending each of its invocation calls to the corresponding IPSs to acquire the results data. For a service invocation call, the appropriate targeting IPS can be located through parsing the **IPS.ipsname**. Then the service invocation call is sent to the IPS and the **IPU.iname** is passed over as the parameter. Since each IPS is provided by a data source at its own site (see section 4.5.1), the IPS constructs the required results data through accessing the local database of the data source and sends it back to respond to the service invocation call.

Once an IPS receives a service invocation call, since the parameter **IPU.iname** clearly specifies the name of the IPU, the local definition **IPU.gd** of the IPU can be located by this name. And then, the local schema $\mathbf{DS_{LS}}$ of the data source can be obtained as it is a constituent component of the IPS (see section 4.5.1). Through interrogating the local definition and the local schema, local queries can be composed for constructing the required results data. Since the local queries refer directly to the local schema, they can be executed directly against the local database. Hence the required results data can be constructed.

As introduced in section 4.4.4.4, the local definition of an IPU defines how the data the IPU provides are formed from the local database. The local definition is in the structure of a DAG where the leaf vertices represent actual relations from the local schema and the non-leaf vertices represent virtual views. The virtual views include root view, intermediate views and basic views where the root view represents the required results data the IPU intends to provide and all the direct successors of a basic view are actual relations. As a view can be evaluated on the premise that all its direct successors are evaluated, the results data of the IPU can be obtained through evaluating the vertices of the DAG in bottom-up fashion.

The example in section 4.4.4.3 is extended here for illustrating how the results data of an IPU are produced. Suppose the local definition of an IPU includes the root view **V**, the intermediate view $\mathbf{V_4}$, the basic views $\mathbf{V_1, V_2, V_3}$ and the actual relations $\mathbf{R_1(A_{11}, A_{12}, A_{13})}$, $\mathbf{R_2(A_{21}, A_{22}, A_{23})}$, $\mathbf{R_3(A_{31}, A_{32}, A_{33})}$. Suppose the view definitions are as following:
The basic views $\mathbf{V_1, V_2, V_3}$ are defined by one CQ each and each CQ only has one subgoal which is an RRS to assign the actual attributes and relations to the views:

$\mathbf{V_1 (X_1, X_2) :- CQ_{V1} (S_1(X_1, X_2), S_1 \rightarrow R_1, X_1 \rightarrow A_{11}, X_2 \rightarrow A_{12})}$

$\mathbf{V_2 (X_3, X_4, X_5) :- CQ_{V2} (S_2(X_3, X_4, X_5), S_2 \rightarrow R_2, X_3 \rightarrow A_{21}, X_4 \rightarrow A_{22}, X_5 \rightarrow A_{23})}$

$\mathbf{V_3 (X_6, X_7, X_8) :- CQ_{V3} (S_3(X_6, X_7, X_8), S_3 \rightarrow R_3, X_6 \rightarrow A_{31}, X_7 \rightarrow A_{32}, X_8 \rightarrow A_{33})}$

The intermediate view $\mathbf{V_4}$ is defined by one CQ which is abstractly represented as

$\mathbf{V_4 (X_3, X_8, X_9) :- CQ_{V4}}$

$\mathbf{(S_5(X_3, X_4, X_5) \wedge S_6 (X_6, X_7, X_8) \wedge S_7(X_3>5) \wedge S_8(X_5=X_6) \wedge S_9(X_9= X_4+ X_7))}$

The $\mathbf{CQ_{V4}}$ is realized by five subgoals as:

$\mathbf{S_5(X_3, X_4, X_5)}$:- $(\mathbf{S_5 \rightarrow V_2})$

$\mathbf{S_6}$ $(\mathbf{X_6, X_7, X_8})$:- $(\mathbf{S_6 \rightarrow V_3})$

$\mathbf{S_7\ (X_3)}$ **:-** $\mathbf{(>, 5)}$

$\mathbf{S_8\ (X_5)}$ **:-** $\mathbf{(=, X_6)}$

$\mathbf{S_9}$ $(\mathbf{X_9})$:- $(+, \mathbf{X_4, X_7})$

Where the RRSs $\mathbf{S_5}$ and $\mathbf{S_6}$ are assigned to the basic views $\mathbf{V_2}$ and $\mathbf{V_3}$ respectively. The ORSs $\mathbf{S_7}$ and $\mathbf{S_8}$ and the CRS $\mathbf{S_9}$ altogether put the constrains onto $\mathbf{CQ_{V4}}$

The root view **V** is defined by one CQ which is abstractly represented as:

$\mathbf{V(X_1, X_9)}$:- $\mathbf{CQ_V(S_{10}(X_1, X_2)} \wedge \mathbf{S_{11}\ (X_3, X_8, X_9)} \wedge \mathbf{S_{12}(X_2{=}X_8))}$

The $\mathbf{CQ_V}$ is realized by three subgoals as:

$\mathbf{S_{10}(X_1, X_2)}$:- $(\mathbf{S_{10} \rightarrow V_1})$

$\mathbf{S_{11}}$ $(\mathbf{X_3, X_8, X_9})$:- $(\mathbf{S_{11} \rightarrow V_4})$

$\mathbf{S_{12}\ (X_2)}$ **:-** $\mathbf{(=, X_8)}$

Where the RRSs $\mathbf{S_{10}}$ and $\mathbf{S_{11}}$ are assigned to the basic view $\mathbf{V_1}$ and the intermediate view $\mathbf{V_4}$ respectively. The ORSs $\mathbf{S_{12}}$ put the constrains onto the CQ

By substituting the subgoals and variables for the corresponding actual relations and attributes according to the assignments, the conjunctive queries can be transformed into local queries which can be directly evaluated against the local database of the data source. For example the $\mathbf{CQ_V}$ in the above example can be transformed into a SQL query as following:

SELECT *R1.A11* **AS** *X1***,** *R2.A22+R3.A32* **AS** *X9*

FROM *R1, R2, R3*

WHERE *R1.A12=R3.A33* **AND** *R2.A21>5* **AND** *R2.A23=R3.A31*

Different data sources may employ different DBMSs to manage their local databases thus they may support different query languages. Even if all the DBMSs support SQL, the syntax can vary slightly between DBMS and DBMS. The transformation of a conjunctive query into a local query is not the focus of this research thus will not be discussed further in this thesis.

After all the basic views are evaluated, they can be treated as actual relations (also referred to as *materialized views* in other research [37]). Thus the intermediate views whose direct successors are basic views can be evaluated in exactly the same way as the basic views. By applying the view evaluation process iteratively, the root view can be finally evaluated thus producing the required results data to respond the service invocation call.

5.5.3 Result Constructing

This step is responsible for generating the results data for the partial service plans and the service plan from the results data of the service invocation calls. As introduced in section 4.4.3, the data an IPU provides is in the form of a unary or a binary relation. Thus the results data produced from a service invocation call are a set of *tuples* stored as the extension of the relation and the variable list in the service invocation call specifies the head of the relation. As described in section 5.5.1, a resulting query is in the form of conjunctive query where the query corresponds to a partial service plan and each subgoal relation in the query corresponds to a service invocation call in the partial service plan. Thus the resulting query can be evaluated through substituting its subgoal relations for the results data relations produced by the corresponding service invocation calls. As a result, after all the service invocation calls in a partial service plan are executed, the results data for the partial service plan can be formed through collecting and composing the results data produced by the service invocation calls.

Since the results data is in the form of a relation, the set of results data of the service invocation calls of a partial service plan can be stored in a temporary relational database. And the set of the relation heads constitutes the schema of the database. The corresponding resulting query can be evaluated through transforming the query into a local query which can be executed directly against this temporary database in exactly the same way as described in the previous section. Hence the results data of a partial service plan can be produced which is in the form of an n-ary relation where the results data is the extension of the relation and the head of the partial service plan specifies the head of the relation.

As described in section 5.4.3.2, each resulting query is a rewriting of the original user query. The resulting queries have identical heads to the original user query and the original user query can be answered by the union of the answers of the resulting queries. Thus after the results data of all the partial service plans in a service plan are produced, they can be stored in the temporary relational database. Since all the results data relations have identical heads, they can be combined together by applying the union operations to the results data relations which may be realized through composing a local query. As a consequence, the final results data for answering the original user query can be constructed through executing the local query against the temporary database. The final results data is also in the form of an n-ary relation and the tuples in the extension of the relation represent the information instances which can be obtained from the participating data sources to fulfil the user information needs.

To sum up, the role of information supplying (see section 3.2) in SLEDI is played through the following process:

1. End users specify their information needs as user queries in terms of the application domain ontology and organization structure.
2. Through employing the technique of answering queries using views, each user query is rewritten into a set of resulting queries where each resulting query only consists of the global definition of the IPUs.
3. The corresponding service invocation calls, partial service plans and service plans are generated according to the involved IPUs, resulting queries and the user queries, in a bottom-up fashion.
4. The service plans are executed through sending invocation calls to the appropriate IPSs which stand at the participating data sources sites. The IPSs construct the results data through composing local queries according to the local definition of the IPUs to respond to the service invocation calls.
5. The results data of the service plans and partial service plans are constructed through collecting and composing the results data produced by the service invocation calls in a bottom up fashion.
6. The final results data of the service plans are returned to the end users to answer the user queries hence to fulfil the users' information needs.

5.6 Summary

This Chapter has presented the process of query processing in SLEDI which delivers information supplying service to end users. The procedure of rewriting user queries into resulting queries is introduced. The operations of converting the resulting queries into service plans, generating corresponding partial service plans and service invocation calls, executing the plans to obtain the results data and constructing the final results data are delineated in detail.

Chapter 6 introduces the automatic evolution handling process which is responsible for modifying the IDS with respect to the various types of evolution. The modification is carried out through firstly identifying the evolution occurring, and then identifying the parts of the IDS which are affected by the evolution and finally applying automatic modifications onto the parts. The details of the process are described.

Chapter 6 Evolution Handling

6.1 Introduction

Chapter 5 presented the details of user query processing. The algorithm for user query rewriting and the process of transforming the queries into service plans were described. Then the operations of executing the service plans and constructing the results data were introduced.

This chapter delineates the automatic evolution handling process in this research which completes the description of the SLEDI method. The organizational and schematic evolutions are addressed. The organizational evolution are identified and then handled by adding a new function into the user query processing. And the schematic evolution are identified and then handled through analysing the local definitions of the corresponding IPUs and automatically modifying the IPUs based on a set of predefined processes.

6.2 Overview of Evolution Handling

As introduced in section 1.3.1, software maintenance is the most expensive stage of the software life cycle. According to the IEEE standards [43], the maintenance process can be defined as having seven stages including: Modification Identification, Analysis, Design, Implementation, Testing, Accept Testing and Delivery. Thus when evolution occurs in a data integration system, the maintenance programmers may conduct the maintenance work based on their knowledge through the following processes:

1. Identifying the evolution
2. Identifying the components of the systems which are affected by the evolution
3. Modifying the components accordingly to keep it functioning properly

In the database integration systems where data integrating is accomplished through hard wiring on the local schemas of the participating data sources, a huge quantum of hard-coded queries may come into existence when the amount of data sources is large. The evolution occurring in the data sources may result in having to analyse and rectify all the existing hard wiring queries which involve a large amount of human effort. This maintenance work can rapidly become unmanageable when evolution occurs frequently.

Since the SLEDI is designed to integrate data from autonomous and *evolving* data sources, the *evolution handling* is one of the main research issues addressed in this thesis. Hence mitigating the human effort in the maintenance work is one of the targets the SLEDI intends to reach. As the data integrating in SLEDI is achieved by describing the data sources through the IPU mechanism, automatic assistance can be employed for evolution handling. The evolution handling process is a simulation of the maintenance work carried out by human effort hence is constituted of the three processes mentioned above.

The evolution covered in this research can be classified into three different types: *organizational evolution*, *schematic evolution* and *system level evolution*. Organizational evolution refers to the changes happening in the organizational structure of the system, the schematic evolution concerns alterations occurring in the local schemas of the data sources and the system level evolution refers to the modifications applied onto the narrative information of the data sources such as the names and URLs for describing the data sources.

As introduced in section 4.5.2, in order to allow fine-grained control from the end users, the participating data sources are also described by the aspects which characterize them. The description categorizes the data sources into the organizational structure. Since the data sources are frequently evolving, the description may change from time to time. As a consequence, to acquire the results data which reflect the latest state of the system, existing user queries may require evolution handling as an organizational structure the queries have been composed against may have evolved. The process of handling the organizational evolution will be described in detail in section 6.3

The alteration arising in the local schema of the participating data sources can result in a great deal of maintenance work, especially in the hard wired data integration system. The data integration in SLEDI is established on the IPU mechanism where each IPU is correlated to the participating data sources through its local definition. Since the local definitions of IPU are defined as virtual views over the local schemas and realized through structure data in the form of DAG (see section 4.4.4.3), the schematic evolution can be handled by employing the automatic assistance to search the structured data and apply modifications accordingly without human intervention. The detailed delineation of the process will be described in section 6.4

The narrative information of the participating data sources such as the names and the URLs of the local databases may also change when the data sources evolve. This system level evolution may also impact the data integration system and hence requires maintenance work to take place. Compared to the hard wired data integration system, the SLEDI is based on SOA and all the IPUs provided by the data sources are accommodated through the IPSs. Thus the maintenance effort for handling the system level evolution can also be decreased thanks to the characteristic of loose coupling, autonomy and discoverability of the services that SOA intrinsically supports. The handling of system level evolution will be covered in the next chapter.

6.3 Organizational Evolution Handling

As introduced previously, user query can specify the targeting data sources it intends to query against through the **Q(org)** with respect to the organizational structure. Then the targeting data sources can be filtered out based on processing the **Q(org)** according to the organizational structure. When organizational evolution occurs frequently, the organizational structure may come into multiple versions and only the latest version may be accessible. As a consequence, the version of the organizational structure that some existing user queries are composed against may not be the current version when the user queries are being answered. The discrepancies between different versions of the organizational structure may cause obstacles to obtaining the proper results data for answering the user queries. Hence the evolution handling process is required to tackle the discrepancies and properly filter out the targeting data sources for answering the user queries.

In SLEDI, the organizational structure is actualized through a set of DAGs where each DAG represents a classification to characterize the data sources from an aspect; each non-leaf node of the DAG represents a virtual group to describe a class in the classification and each leaf node represents a data source which is classified as an instance of the class where its direct predecessor represents. Thus the discrepancy between two adjacent versions of organizational structure can be described as the transformation between two DAGs and the transformation can be correlated to the evolution occurring which caused the alteration of the organizational structure. As a consequence, the relationship between the original version of the organizational structure the user queries are composed against and the current version when the queries are being answered can be established and automatic assistance can be employed to filter out the targeting data sources properly despite the discrepancies created by the organizational evolution.

The automatic assisted evolution handling for the organizational evolution is carried out through the processes of Organizational Evolution Identification and Organizational Evolution Solving. By analysing the organizational evolution occurring, the process of Organizational Evolution Identification records each evolution and correlates them with the transformations which represent the corresponding alterations in the organizational structure. Hence the Organizational Evolution Identification keeps tracks of all the evolution occurring in the organizational structure as a set of *records*. Then the process of Organizational Evolution Solving takes into place, through examining the **Q(org)** of the user query which should be answered with respect to the records, the process can determine whether the user query is affected by the evolution. If not affected, no further evolution handling action is needed and the user query is answered through the query answering process which has been described in chapter 5. On the contrary, if the user query is affected, the evolution solving process rewrites the **Q(org)** into an equivalent formula with respect to the current version of the organizational structure by accessing the records. Then the rewritten user query is answered by the query answering process. The details of the processes are described in the following sections.

6.3.1 Organizational Evolution Identification

As introduced previously, Evolution Identification plays the role of identifying the evolution occurring in the organizational structure. Since the organizational structure is actualized through the structure of DAG where the root nodes, non-leaf nodes and leaf nodes correspond to the classifications, virtual groups, and data sources (see section 4.5.2), the organizational evolution covered in this research as introduced in section 2.3.2 can be described with respect to the organizational structure hence can be categorized into three different levels, and each level involves several different types of atomic evolution. They are described in details below:

6.3.1.1 The Types of Organizational Evolution

1) **Data Source Level**
   - *Data Source Addition*: a new data source is added to the organizational structure as a new leaf node
   - *Data Source Rename*: a current data source changes its name
   - *Data Source Removal*: a current data sources drops out of the organizational structure
2) **Group Level**
   - *Group Addition*: a new group is added to the organizational structure as a new non-leaf node
   - *Group Rename*: a current group changes its name
   - *Group Removal*: a current group drops out of the organizational structure
   - *Group separation*: a current group is split into two or more groups and all the data sources belonging to the original group are distributed into the new groups.
   - *Group Aggregation*: two or more current groups are aggregated together into a new group and all the data sources belonging to original groups are put into the new group
3) **Classification Level**
   - *Classification Addition*: a new classification is added to the organizational structure as a new DAG
   - *Classification Rename*: a current classification changes its name
   - *Classification Removal*: a current classification drops out of the organizational structure
   - *Classification separation*: a current classification is split into two or more classifications and all the groups belonging to the original classification are distributed into the new classifications.

- *Classification Aggregation*: two or more current classifications are aggregated together into a new classification and all the groups belonging to the original classifications are put into the new classification

Although the organizational evolution can appear in various ways, by defining the atomic types of organizational evolution above, each organizational evolution occurring in the organizational structure can be represented by one of the atomic organizational evolution. It is worth mentioning that the atomic organizational evolution defined above separate the concerns of the organizational evolution into different levels. For example, the *separation* and *aggregation* evolution in the group level only concerns evolution of the virtual groups which require that the data sources belonging to the groups remain unchanged during these atomic organizational evolutions. And the atomic organizational evolutions in the classification level follow the same fashion.

From the process of query processing described in chapter 5, only the data source filtering sub-process may be affected by the organizational evolution (see section 5.3.2). Since the sub-process filters out the targeting data sources set through extracting the names of the classifications and virtual groups in the Atomic Targets specified by the **Q(org)**, and then traversing the organizational structure to filter out all the data sources which are the successors of the groups. It is obvious that some of the atomic organizational evolution defined above will not affect the query processing thanks to the design of the organizational structure. For example, for the atomic organizational evolution in the data source level, being the leaf nodes of the DAGs, no matter the addition, rename or removal of the data sources happing, the groups and classifications remains unchanged thus the targeting data sources will be properly filtered out according to the current version of the organizational structure. Also for the addition to the group and classification level, since the newly-added group and classification do not exist in the **Q(org)** of any existing user queries, the query processing will not be affected.

6.3.1.2 The Model of Organizational Evolution

Since the Organizational Evolution Identification takes the responsibility of recording the organizational evolution occurring and correlates them with the corresponding alterations of the organizational structure, a formal model is created in this research to describe the atomic organizational evolution and the corresponding alterations. As some of the atomic organizational evolution will not affect the query processing, only those which may affect the query processing are recorded. The model is defined below:

Atomic Organizational Evolution : (*aoetype* : **Atomic Organizational Evolution Type**, *aoe* : **Atomic Organizational Evolution Description**)

Atomic Organizational Evolution Type : String

$\forall$X : **Atomic Organizational Evolution Type**, X $\in$ {*"Group Rename", "Group Removal", "Group separation", "Group Aggregation", "Classification Rename", "Classification Removal", "Classification separation", "Classification Aggregation"* }

Since different types of the atomic organizational evolution may trigger different kinds of alterations onto the organizational structure, the atomic organizational evolution descriptions require different models to represent hence are modelled separately below:

Atomic Organizational Evolution Description : (*classification* : **String**, *group name* : **String**, *altered group name* : **String**) (where *aoetype* $\in$ {*"Group Rename"* , *"Group Aggregation"*})

Atomic Organizational Evolution Description : (*classification* : **String**, *group name* : **String**) (where *aoetype* = *"Group Removal"*)

Atomic Organizational Evolution Description : (*classification* : **String**, *group name* : **String**, { *altered group name* : **String** }) (where *aoetype* = *"Group separation"*)

Atomic Organizational Evolution Description : (*classification* : **String**, *altered classification name* : **String**) (where *aoetype* $\in$ {*"Classification Rename"* , *"Classification Aggregation"*})

Atomic Organizational Evolution Description : (*classification* : **String**) (where *aoetype* = *"classification Removal"*)

Atomic Organizational Evolution Description : (*classification* : **String**, { *altered classification name* : **String** }) (where *aoetype* = *"classification separation"*)

And then, the organizational evolution can be modelled as:

Organizational Evolution { *ae* : **Atomic Evolution** }

When organizational evolution occurs, the organizational structure is modified accordingly to reflect the evolution. Since the evolution handling is focused on generating proper answers for the user queries, how to modify the organizational structure is trivial for this research hence is not covered. The purpose of the evolution identification is to keep the tracks of all the organizational evolution ever occurring in the IDS, thus it is assumed that every time organizational evolution occurs, the process of evolution identification is applied and the *Organizational Evolution* data model keeps all the records up to the current version of the organizational structure. Consequently, further evolution handling actions can be taken based on the records which will be discussed in detail in the next section.

6.3.2 Organizational Evolution Solving

As discussed in section 6.2, the automatic assisted evolution handling is a simulation of human effort which consists of the three processes. Through defining the atomic organizational evolution and representing them in the data model above, the organizational evolution have been identified. Thus the next stages are identifying the parts of the IDS which are affected by the evolution and modifying them accordingly. These targets are achieved by the process of *organizational evolution solving* in this research. And the process consists of the following steps:

1. Assess whether the user query is affected by the organizational evolution through parsing the classification and groups specified in the **Q(org)** and accessing the evolution records
2. Taking appropriate actions based on the assessment in the previous step. If the query is not affected, pass the query to the query processing to acquire the results data. If the query is affected, rewrite the **Q(org)** into a corresponding formula through accessing the evolution records and the current version of the organizational structure before passing the query to the query processing.

Since the user queries may require evolution solving before going to query processing, the evolution solving is introduced as the first process into the query processing. This means every user query which requires an answer will first be processed by the evolution solving process and the process is realized through the following steps:

1. Let ***R*** be the organizational evolution records built up in the organizational evolution identifying
2. Let ***AT*** be the atomic targets list in the **Q(org)** of the user query
3. If ***AT*** ≠ Ø, for each element *atr* in ***AT***, attempt to find the element *r* in ***R*** that the *classification* of the atomic organizational evolution description of *r* is identical with the *Classification root name* of *atr*. If *r* is found, let *ty* be the atomic organizational evolution type of *r*
    i. If *ty* ∈ { "*Classification Removal*", *"Group Removal"* }, reject the user query
    ii. If *ty* ∈ { *"Classification Rename"* , *"Classification Aggregation"* }, rewrite the *Classification root name* of *atr* as the *altered classification name* of the atomic organizational evolution description of *r*
    iii. If *ty* ∈ { *"Group Rename"* , *"Group Aggregation"* }, if the *Group name* of *atr* is identical with the *group name* of the atomic organizational evolution description of *r*, rewrite the *Group name* of *atr* as the *altered group name* of the atomic organizational evolution description of *r*.
    iv. If *ty* = "*classification separation*", let *grp* be the *Group name* of *atr* and ***ACN*** be the list of the *altered classification name* of the atomic organizational evolution description of *r*. For each element *ac* in the ***ACN***, add a new element *atrn* in AT, then set the value of *ac* and *grp* into the *Classification root name* and *Group name* of *atrn* respectively. Then delete the atr from AT
    v. If *ty* = "*Group separation*", if the *Group name* of *atr* is identical with the *group*

name of the atomic organizational evolution description of *r*, let ***AGN*** be the list of the *altered group name*, For each element *ag* in the ***AGN***, add a new element *atrn* in AT, then set the value of *classification* of *r* and *ag* into the *Classification root name* and *Group name* of *atrn* respectively. Then delete the *atr* from AT

4. output ***AT*** as the rewritten **Q(org)** of the user query

As the organizational evolution records keep track of all the organizational evolution occurring, through comparing the classifications and groups specified in the **Q(org)** with their counterparts in the records, the process of organization evolution solving can determine whether the user query is affected by the evolution. Through traversing the records, the equivalence in the current organizational structure of the classifications and groups specified might be found. In the case of rename and aggregation in the classification and group level, the classifications and groups specified in the **Q(org)** are replaced by their corresponding counterparts in the current version of the organizational structure. In the case of separation in the classification and group level, since each classification and group specified in the **Q(org)** may correspond to two or more classifications and groups, extra atomic targets are added to the **Q(org),** as appropriate. And under the circumstances of classification and group removal, the original user query is rejected since the targeting data sources the user query intendeds to query against are no longer available. As a consequence, the **Q(org)** of the user queries are modified accordingly with respect to the current version of the organizational structure and the semantics of the user query are preserved. The process of the organizational evolution solving can be modelled as a function:

Organizational Evolution Solving :

User Query (Q(org), Q(onto)) → **User Query ($\mathbf{Q(org)^{rewriting}}$, Q(onto))**

6.4 Schematic Evolution Handling

As introduced in section 1.3.1 and section 6.2, the schematic evolution refers to the evolution occurring in the local schema of the participating data sources. Schematic evolution handling is one of the central evolution issues the SLEDI is designed to tackle, as the alterations arising in the local schema may result in large amount of maintenance work. Compared to the hard wired data integration approach, The IPU mechanism of SLEDI not only provides a solution for integrating the data from the participating data sources, but, more importantly, it also builds the foundation for introducing automatic assistance to handle the schematic evolution. Hence human effort involved in the maintenance work may be mitigated.

In SLEDI, the participating data sources expose the data they are willing to share through the IPU mechanism. The IPU consists of Global Definition and Local Definition where the Local Definition describes how the data that IPU provides are constructed through specifying the virtual views over the local schema of the participating data sources. Thus when schematic evolution occurs in the data sources, alterations may arise in the local schema, and, as a consequence, the virtual views specified by the local definition of the IPUs might become invalid, thus the IPUs may not be able to construct data properly over the evolved local schema. Since the virtual views are realized through DAGs, by introducing automatic assistance to examine the schematic evolution and the DAGs, the evolution might be handled without human intervention and the IPUs can be modified accordingly in order to be compatible with the evolved local schema.

Similarly with the organizational evolution handling, the schematic evolution handling of SLEDI is delivered through the process of Schematic Evolution Identification and Schematic Evolution Solving. Once the schematic evolution occurs in a data source, the process of schematic evolution identification identifies the type of the evolution and describes the evolution with the corresponding alterations in the local schema the evolution caused. Then the process of schematic evolution solving can search the local definitions of the IPUs the data source provided to filter out those IPUs whose local definitions are affected by the schematic evolution. After the affected IPUs are identified, through analysing the schematic evolution and the local definitions of the IPUs, appropriate actions are taken to automatically modify the affected IPUs in order to make them still semantically and syntactically valid for the IDS. In some circumstances, human intervention may be required and some affected IPUs may have to be disposed of. The processes are delineated in detail in following sections.

6.4.1 Schematic Evolution Identification

As introduced previously, the first step of evolution handling identifies the evolution occurring, thus further actions can be determined based on it. The schematic evolution identification identifies the types of schematic evolution occurring in the data sources with the corresponding alterations onto the local schemas the evolution caused. Since there are no materials to define a complete delineation of the possible schematic evolution for the time being and the database schemas involved in this research are relational schemas, the schematic evolution covered in this research are defined from the manipulations provided by the relational DBMS for the administrators to conduct onto the database schemas. Thus the schematic evolution can be categorized into three different levels and each level involves several types of schematic evolution.

6.4.1.1 The Types of Schematic Evolution

The three levels of schematic evolution are *Attribute Level*, *Relation Level* and *Database Level* which refer to the evolution causing alterations to the attributes, relations and databases of the local schema respectively. They are listed below:

Attribute Level:

- *Attribute Addition*: a new attribute is added to a relation
- *Attribute Removal*: an existing attribute is dropped from a relation
- *Attribute Rename*: an existing attribute changes its name
- *Attribute Domain Change*: an existing attribute changes its domain (i.e. data type)
- *Attribute Decomposition*: an existing attribute is divided into two or more attributes
- *Attribute Aggregation*: two or more attributes are aggregated into one attribute

Relation Level:

- *Relation Addition*: a new relation is added to a database
- *Relation Removal*: an existing relation is dropped from a database
- *Relation Rename*: an existing relation changes its name
- *Relation Decomposition*: an existing relation is divided into two or more relations
- *Relation Aggregation*: two or more relations are aggregated into one relation

Database Level:

- *Database Addition*: a new database source is added to the IDS
- *Database Removal*: an existing database source is dropped from the IDS

Through the descriptive definitions above, the various types of schematic evolution are listed with respect to the alterations they cause onto the database schema. As described in section 4.5.1, a participating data source of the IDS shares its data through connecting the local definition of the IPUs with its local schema; hence the results data the IPUs provide can be constructed via composing the local queries based on the local definitions and executing the queries against the local database. Once the schematic evolution occurred in the local database, the local schema may be altered. Thus two versions of the local schemas may come into existence. Since the local definition is defined with respect to the original version of the local schema, the local queries may not be able to run against to the currently local schema which is in fact the evolved version.

Although various schematic evolutions are listed above, some types of evolution may not introduce impact onto the local definitions of the existing IPUs. For example, in the case of an addition to the attribute and relation level, although the newly-added attributes and relations may require human effort to integrate them into the IDS by following the algorithm described in chapter 4, they did not exist when the existing IPUs were constructed, thus all the existing local definitions would not be affected. The same situation applies to the addition of databases; although the addition of a database may also affect the organizational structure (see section 6.3.1.1), for all the existing IPUs, no impact would be made. For other types of schematic evolution, the IPUs may be affected hence may have to be modified in order to work properly.

6.4.1.2 The Model of Schematic Evolution

Although the schematic evolution discussed above can be intuitively understood by human maintenance programmers, the lack of precise definition makes it difficult for employing automatic assistance into the evolution handling. Thus a formal model is created in this research to describe the schematic evolution. Through correlating each type of the schematic evolution with the corresponding alteration onto the schema, the schematic evolution can be identified and further actions can be determined. Similar with the organizational evolution, only those evolution which may affect the IPUs are modelled:

Schematic Evolution: (*ipsname* : **String,** *sevt* : **Schematic Evolution Type**, *se* : **Schematic Evolution Description**)

Since each schematic evolution occurs in a specific data source and each data source shares its data through an IPS, the first property of the schematic evolution model is to specify in which data source the schematic evolution occurred by indicating the name of the IPS. The second property indicates the type of the schematic evolution and the third property specifies the details of the schematic evolution.

Schematic Evolution Type : String

∀X : **Evolution Type**, X ∈ {*"Attribute Removal", "Attribute Rename", "Attribute Domain Change", "Attribute Decomposition", "Attribute Aggregation", "Relation Removal", "Relation Rename", "Relation Decomposition", "Relation Aggregation" , "Database Removal"* }

Different types of schematic evolution cause different alterations onto the schema thus require different data to describe the details. They are modelled separately below:

1) **Schematic Evolution Description** : (*re* : **Relation,** *att* : **Attribute**) (where *sevt="Attribute Removal"*)

The first property *re* refers to the relation in the schema which the removed attribute belongs to and the second property *att* refers to the attribute which is removed by the schematic evolution.

2) **Schematic Evolution Description** : (*re* : **Relation,** *attori* : **Attribute,** *attevo* : **Attribute**) (where *sevt="Attribute Rename"*)

The first property *re* refers to the relation in the schema where the evolved attribute belongs. The second and third properties refer to the attribute before and after the evolution respectively.

3) **Schematic Evolution Description**: (*re* : **Relation,** *attori* : **Attribute,** *attevo* : **Attribute,** *opn* : **Operation**) (where *sevt="Attribute Domain Change"*)

In this research, attribute domain change refers strictly to an attribute which only changes its domain (i.e. data type) during the evolution; the semantic of the attribute remains unchanged.

Consequently, every instance of the attribute has two types of representation data; one in the form of the original attribute and another in the form of the evolved attribute. Thus there is a one to one mapping between the instances of the original attribute and the evolved attribute. Similarly with the attribute rename; the first three properties of the attribute domain change refer to the relation the evolved attribute belongs to, the attribute before and after the change. The fourth property is an *operation* (recall section 4.4.4.4). The operation describes how to convert the instances of the evolved attribute into the corresponding instances of the original attribute.

4) **Schematic Evolution Description** : (*re* : **Relation,** *attori* : **Attribute,** {*attevo* : **Attribute**}**,** {*opn* : **Operation**}, {(*varop* : **Variable**, *attevo* : **Attribute**)}) (where *sevt=“Attribute Decomposition”*)

The attribute decomposition refers to an original attribute decomposed into two or more evolved attributes where all the original and evolved attributes reside in the same relation and the instances of the original attribute can be constructed through converting the instances of the evolved attributes. Similarly with the attribute domain change; the first three properties refer to the relation, the original attribute and list of the evolved attributes. The fourth property refers to the list of operations for constructing the instances of the original attribute from the evolved attributes and the fifth property is a list of pairs where each pair correlates an attribute with the corresponding variable in the operation.

5) **Schematic Evolution Description**: (*re* : **Relation,** {*attori* : **Attribute**}, *attevo* : **Attribute,** {*opn* : **Operation**}, {(*varop* : **Variable**, *attevo* : **Attribute**)}) (where *sevt=“Attribute Aggregation”*)

The attribute aggregating refers to two or more original attributes aggregated into one evolved attribute. Similarly with the attribute decomposition, all the original and evolved attributes belonging to the same relation and the instances of the original attributes can be formulated from the instances of the evolved attribute. The first property refers to the relation, the second property refers to the list of the original attributes, the third property refers to the evolved attribute and the fourth property refers to the list of operations for constructing the instances of the original attributes from the instances of the evolved attribute. Each operation in the list corresponds to one original attribute. And the fifth property refers to the correlations between the attributes and the variables in the operations.

6) **Schematic Evolution Description** : (*re* : **Relation**) (where *sevt=“Relation Removal”*)

The relation removal only has one property *re* which refers to the relation removed from the schema.

7) **Schematic Evolution Description** : (*reori* : **Relation,** *reevo* : **Relation**) (where

sevt=*"Relation Rename"*)

The relation rename has two properties where the first property *reori* refers to the original relation before the evolution and the second property *reevo* refers to the evolved relation after the evolution.

8) **Schematic Evolution Description** : (*reori* : **Relation,** {*reevo* : **Relation**}, {*rbv* : **View**}, *riv* : **View**) (where *sevt*=*"Relation Decomposition"* $\wedge$ *rbv.vtype* = *"basic"* $\wedge$ *riv.vtype* = *"intermediate"*)

The relation decomposition refers to an original relation separated into two or more relations. More precisely, all the attributes in the original relation are distributed into the evolved relations. Since all the attributes are preserved through the evolution, the instances of the original relation can be constructed through applying relational operations over the instances of the evolved relations. In other words, only those evolutions where the original relation can be rebuilt without loss of information are considered in this research. As introduced in section 4.4.4.3, the relational operations over the evolved relations can be described through the view definitions. The model of relation decomposition consists of four properties. The first property refers to the original relation before the evolution; the second property refers to the list of all the evolved relations after the evolution; the third property refers to the list of basic views connected to the evolved relations and the fourth property refers to the intermediate view which specifies how to compose the instance data of the original relation from the instance data of the corresponding basic views of the evolved relations.

9) **Schematic Evolution Description** : ({*reori* : **Relation**}**,** *reevo* : **Relation**, { *reori* : **Relation,** *riv* : **View** }, *rbv* : **View**) (where *sevt*=*"Relation Aggregation"* $\wedge$ *rbv.vtype* = *"basic"* $\wedge$ *riv.vtype* = *"intermediate"*)

The relation aggregation refers to two or more original relations aggregated into one evolved relation. Similarly with the relation decomposition, all the attributes belonging to the original relations are preserved during the evolution and the instances of the original relation can be rebuilt from the instances of the evolved relations. The model of relation aggregation consists of four properties, the first property refers to the list of the original relations; the second property refers to the evolved relation; the third property refers to the list of pairs of the original relation and intermediate view, where the intermediate view represents how to construct the instance data of the original relation it corresponds to from the instances data of the basic view of the evolved relations; the fourth property refers to the basic view correlating to the evolved relation.

10) **Schematic Evolution Description** : () (where *sevt*=*"Database Removal"*)

The database removal refers to the evolution of the entire database removed from the IDS. Since every data source only uses one database, the database removal results in all the IPUs the data source provides are not available any more thus need to be removed. As the corresponding IPS of the IPUs are already specified in the *ipsname*, no further detailed data is required in the model of the evolution description.

When the schematic evolutions occur in data sources, the corresponding database schemas are altered accordingly. How to amend the local schema properly according to the schematic evolution is trivial for this research hence is not discussed. The process of evolution identification plays the role of recording the schematic evolution occurring with the necessary detailed data in order to build the basis from which further actions can be taken. Through the formal model defined above, every single schematic evolution is identified and represented. Each schematic evolution is considered in isolation with no connection to another. After schematic evolution is identified, the process of schematic evolution solving takes place. The process analyses the IPUs of the corresponding data source where the schematic evolution occurred, filters out those affected IPUs and modifies them accordingly. The detail of the process is described in the next section.

6.4.2 Schematic Evolution Solving

After the schematic evolution is identified, the impact the evolution have on the IDS can be examined and the appropriate actions can be taken to handle the evolution. This task is carried out through the process of schematic evolution solving. Similarly with the organizational evolution, the schematic evolution identification provides detailed data and the schematic evolution solving processes the data to accomplish the evolution handling. However, unlike when all the organizational evolution are identified and recorded together, each schematic evolution is identified and solved individually and immediately. From the IDS point of view, each schematic evolution can be considered as an event. Once the event happens, the evolution is identified immediately and is solved by the process of schematic evolution solving before another schematic evolution occurs.

As introduced previously, the evolution handling is a simulation of human effort on maintenance work. Once a schematic evolution is identified, the actions of identifying the affected parts of the IDS and modifying those parts can be taken. Since the only parts of the IDS which may be affected by schematic evolution are the IPUs, the process of schematic evolution handling needs to pinpoint all the affected IPUs and modify them accordingly. The process consists of the following steps:

1. Identifying the affected IPUs: Through parsing the data in the formal model which represents the schematic evolution, the altered attributes, relations and the database relating to the evolution can be identified. Then all the IPUs connected with the database can be identified and the IPUs whose local definition is affected by the altered attributes and

relations can be filtered out.

2. Modifying the affected IPUs: A set of processes are defined and associated with each type of the schematic evolution. For each of the affected IPUs filtered out from the previous step, automatic modification onto the local definition of the IPU is applied based on the processes. Under some circumstances, the IPUs may have to be disposed of.

In order to precisely delineate the process of schematic evolution handling, some assumptions about the IDS and the evolution are explicitly made. It is assumed that all the IPUs of the IDS are valid before a schematic evolution occurs. By means of valid IPU, it refers to the global definition properly declares what data the IPU provides with respect to the application domain ontology; and local definition is properly defined with respect to the local schema. The local definition complies with the validation rules described in section 4.4.4.5 and can be used to generate local queries to construct the proper results data for the IPU from the local database.

6.4.2.1 Affected IPU Identification

Once a schematic evolution is identified, the process of schematic evolution solving determines the affected IPUs by taking the schematic evolution data model and the list of IPSs of the IDS (see section 4.5.1) as input, and then filters out all the affected IPUs as the output. Thus the identification of affected IPUs can be considered as a function:

Affected IPU Identification: (*se*: **Schematic Evolution**, {*ips*: **IPS**}) → {*ipu* : **IPU**}

The affected IPU identification is carried out through the following steps:

1. Let *ipsen* be the *ipsname* of the schematic evolution *se*; ***IPS*** be the list of all the IPSs; *sevt* be the schematic evolution type of *se*; *re* be the relation in the schematic evolution description of *se* and ***AI*** be an empty list.
2. Attempt to find the element *ips* in ***IPS*** where the *ipsname* of *ips* is identical with *ipsen*.
3. If *ips* is found, put all the IPUs of *ips* in a list ***IPU***
4. If *sevt* ≠*"Database Removal"* and ***IPU*** ≠ Ø, for each element *ipu* in ***IPU***
    i. Find all the basic views of the local definition of *ipu* and put them into a list ***BV*** (see section 4.4.4.4)
    ii. For each element *bv* in the BV, find all the relations in the dependency list of *bv* and put them into a list ***DEP***
    iii. Attempt to find the element *dep* in ***DEP*** where *dep* is identical with *re*. If *dep* is found, put *ipu* into ***AI*** and remove duplicates.
5. Output ***AI***.

As illustrated above, the relations in the dependency list of the basic views of a local definition indicates there is a connection between the relations and the IPU which the local definition belongs to. Since the schematic evolution data model specifies the relations which have been altered, through comparing the altered relations and the relations involved in the local definition of an IPU, whether the IPU is affected by the schematic evolution can be determined. If the schematic evolution is database removal, no comparison is needed as all the IPUs connected with the database require removal. After the process of affected IPU identification, the affected IPUs are filtered out into a list and output for further processing.

6.4.2.2 Affected IPU Modification

After the list of affected IPUs is generated, the modification onto them can be applied. Generally, since the schematic evolution introduces the discrepancies between the schema the local definition of the affected IPUs are defined over and the schema the local database currently has after the schematic evolution, the purpose of the modification is to make the two schemas consistent again through modifying the local definition of the affected IPUs. As the different types of schematic evolution alter the local schema in different ways hence cause different impact onto the local definition of the affected IPUs, the process of affected IPU modification is applied through a set of predefined processes with respect to the various types of schematic evolution defined in section 6.4.1.

Although the affected IPU modification intends to modify the local definition of the IPUs to be consistent with the current local schema in order to continuingly construct proper results data for the IPUs, it may not be achievable under some circumstances. Recall the validation rules described in section 4.4.4.5, the rules introduce a set of constraints the local definition of IPU must apply for generating results data properly. Thus the validity of the local definition of IPU can be defined:

1. A Local Definition of IPU is considered *valid* if it complies with the validation rules and all of the *views* in the definition are valid.
2. A View is *valid* if it complies with the validation rules and all the *conjunctive queries* in its definition are valid.
3. A Conjunctive Query is *valid* if it complies with the validation rules and all the *subgoals* in its definition are valid.
4. A Subgoal is *valid* if it complies with the validation rules and all the *assignments* in its definition are valid.
5. An Assignment is *valid* if it complies with the validation rules.

Once the assignments, subgoals, conjunctive queries, views and local definitions are not valid any more after the modification, they have to be disposed of.

The general process of the affected IPU modification is to firstly apply the schematic evolution onto the relations in the dependency list of all the basic views. By doing so, those relations become consistent again with the current local schema after the schematic evolution occurs. And then, the definitions of the basic views are examined to determine whether they are still valid and modifications are applied accordingly if they are not valid. If the basic views are not valid after the modification, they are disposed of. After that, the same examining, modifying and disposal processes are applied onto all the intermediate views and then the root view. Finally, all the view definitions in the local definition are either modified to be valid or disposed of, then the corresponding local definition becomes valid again with respect to the evolved local schema or can be disposed of.

The process of affected IPU modification takes the schematic evolution data model and the list of affected IPUs from the affected IPU Identification as its input to perform the modification. In order to precisely describe the modification process, some clarifications are described below:

1. Two attributes A and B are said to be identical if the name of A equals the name of B and the data type of A also equals the data type of B.
2. Two relations R1 and R2 are said to be identical if the name of R1 equals to the name of R2 and the attributes in R1 are identical with the attributes in R2.
3. Two variables V and V' are said to be identical if the name of V equals the name of V' and the data type of V also equals the data type of V'.

It is also assumed that once the modifications of the affected IPUs are undertaken, it takes effect immediately onto the IPSs in the IDS (see section 4.5.1). The unique name assumption is also applied, which means any item in the IDS such as IPU, local definition, view, variable, attribute etc. uses a unique name to represent itself. There are no two items using same literals for their names. The processes of applying the modification are described in details below with respect to the different types of schematic evolution:

1) Modification process for attribute removal

If the type of the schematic evolution is *"Attribute Removal"*, extract the relation *r*e and the attribute *att* out from the schematic evolution model; let ***AI*** be the list of affected IPUs and ***DELVAR*** and ***DELV*** be empty lists, then

1. For each element *ipu* in the ***AI***, let *ld* be the local definition of *ipu*; *rv*, ***IV***, ***BV*** be the root view, the list of intermediate views and the list of basic views of *ld* respectively.
2. If ***BV*** ≠ Ø, for each element *bv* in ***BV***, let ***CQ*** be the list of conjunctive queries of *bv* and ***R*** be the dependency list of *bv*.
3. If ***R*** ≠ Ø, for each element *r* in ***R***, if *r* is identical with *r*e, attempt to find attribute *att'* in *r* where *att'* is identical with *att*. If *att'* is found, remove *att'* from *r*.
4. If ***CQ*** ≠ Ø, for each element *cq* in ***CQ***, let ***SBG*** be the list of subgoals of *cq*. And for each element *sbg* in ***SBG***, let *sbgtp* be the subgoal type of *sbg* and *asi* be the assignment of the *sbg*.

i. For every *sbg* whose *sbgtp* = "*regular*", if the relation *re'* in the *asi* is identical with *re*, attempt to find the attribute *att'* of *re'* which is identical with *att*. If *att'* is found, remove *att'*. In the list of variable and attribute pairs of *asi*, attempt to find the attribute *att'* which is identical with *att*. If *att'* is found, put the variable *asvar* which is paired with *att'* into ***DELVAR*** and remove duplicates. Then remove the pair of *asvar* and *att'* from *asi*.

ii. For every *sbg* whose *sbgtp* = "*ordered*", attempt to find element *v'* in ***DELVAR*** which is identical with the variable *v* in the head of *sbg* (*sbg* has exactly one variable in its head when *sbgtp* = "*ordered*", see section 4.4.4.4). If *v'* is found, remove the *sbg* and the corresponding *cq*.

iii. For every *sbg* whose *sbgtp* = "*converting*", let ***OP*** be the list of operation of *asi*. For each element *op* in ***OP***, attempt to find element *v'* in ***DELVAR*** which is identical with any of the variables in *op*. If *v'* is found, remove *op*. If ***OP*** = Ø , remove the *sbg* and the corresponding *cq*.

5. Until all *sbg* in ***SBG*** are processed, check whether the corresponding *cq* is valid against the validation rules. If *cq* is not valid, remove the *cq* from ***CQ***.
6. Until all *cq* in ***CQ*** are processed, check whether the corresponding *bv* is valid against the validation rules. If *bv* is not valid, put the *viewname* of *bv* into ***DELV*** and remove duplicates. Then remove *bv* from ***BV***.
7. If ***IV*** ≠ Ø, for each element *iv* in ***IV***, let ***ICQ*** be the list of conjunctive queries of *iv* and ***DPL*** be the dependency list of *iv*.
8. If ***DPL*** ≠ Ø, for each element *dpl* in ***DPL***, attempt to find the element *viewname'* in ***DELV*** which the *viewname'* is identical with the *viewname* of *dpl* (*dpl* can only be view when *iv* is an intermediate view, see section 4.4.4.4). If *viewname'* is found, remove *dpl* from ***DPL***.
9. If ***ICQ*** ≠ Ø, for each element *icq* in ***CQ***, let ***ISBG*** be the list of subgoals of *icq*. And for each element *isbg* in ***ISBG***, let *isbgtp* be the subgoal type of *isbg* and *iasi* be the assignment of the *isbg*.
10. For every *isbg* whose *isbgt* = "*regular*", let *vie* be the view in *iasi* and attempt to find the element *viewname'* in ***DELV*** which the *viewname'* is identical with the *viewname* of *vie*. If *viewname'* is found, remove *icq* from ***CQ***.
11. Until all the *icq* in ***CQ*** are processed, check whether the corresponding *iv* is valid against the validation rules. If *iv* is not valid, put the *viewname* of *iv* into ***DELV*** and remove duplicates. Then remove *iv* from ***IV***.
12. Repeat from step 7 iteratively until no more *iv* can be removed.
13. Attempt to find an identical match between the elements in ***DELV*** and the *viewname* of views in the dependency list of *rv*. If found, remove the corresponding view from the dependency list.
14. Check whether the conjunctive queries in the *rv* are valid against the validation rules and

remove the invalid ones.

15. Check whether the *rv* is valid against the validation rules. If not valid, remove the *ipu* and repeat from step 1 until all the *ipu* in the ***AI*** are processed.

The process above solves the attribute removal through firstly removing the attributes from the basic views of the local definition for each of the affected IPUs. Then it validates the basic views against the validation rules and invalid basic views are removed from the local definition. If the corresponding variables of the removed attribute are involved in CRS or ORS in a basic view, the view is considered as invalid as the semantics of the view are changed. After the basic views are validated, each intermediate view can be validated as soon as all the views which are its direct successors have been already validated. Through validating the views iteratively, the root view can be finally validated. If the root view is not valid, it cannot construct results data properly from the evolved local schema and the IPU is removed.

2) Modification process for attribute rename

If the type of the schematic evolution is *"Attribute Rename"*, extract the relation *r*e, the original attribute *attori* and evolved attribute *attevo* from the schematic evolution model; let ***AI*** be the list of affected IPUs then,

1. For each element *ipu* in the ***AI***, let *ld* be the local definition of *ipu*; ***BV*** be the list of basic views of *ld*.
2. If ***BV*** ≠ Ø, for each element *bv* in ***BV***, let ***CQ*** be the list of conjunctive queries of *bv* and ***R*** be the dependency list of *bv*.
3. If ***R*** ≠ Ø, for each element *r* in ***R***, if *r* is identical with *r*e, attempt to find attribute *att'* in *r* where *att'* is identical with *attori*. If *att'* is found, replace *att'* with *attevo*.
4. If ***CQ*** ≠ Ø, for each element *cq* in ***CQ***, let ***SBG*** be the list of subgoals of *cq*. And for each element *sbg* in ***SBG***, let *sbgtp* be the subgoal type of *sbg* and *asi* be the assignment of the *sbg*.
5. For every *sbg* whose *sbgt* = "*regular*", if the relation *rela* in the *asi* is identical with *r*e, in the list of variable and attribute pair of *asi*, attempt to find the attribute *asatt* which is identical with *attori*. If *asatt* is found, replace *asatt* with *attevo*.
6. Check whether *bv* is valid against the validation rules. If not valid, remove *bv*.
7. Validate all the views in *ld* iteratively and remove the invalid views, then repeat from step 1 until all the *ipu* in the ***AI*** are processed.

Recall section 4.4.4.3 and section 4.4.4.4; every actual relation in the local schema is connected to the local definition of IPUs through a basic view. Each relation is mapped to a regular type of subgoal (i.e. RRS) in the definition of the basic view. Through the list of paired attributes and variables in the assignment of the subgoal, each attribute is correlated with a variable. Thus the attribute rename can only affect the definition of basic views as all the direct predecessors of the basic view use the variables instead of the actual relation and attributes in their definition. The above process solves the attribute rename through firstly updating the relations in the dependency lists of the basic views to be consistent with the evolved local schema. Then it alters the correlation of the evolved attribute so that the variable correlated with the original attribute is redirected and correlated with the evolved attributes. And then validate the basic views. Finally all the views in the local definition are validated and the affected IPUs are updated to be valid with respect to the evolved local schema.

3) Modification process for attribute domain change

If the type of the schematic evolution is *"Attribute Domain Change"*, extract the relation *r*e, the original attribute *attori*, the evolved attribute *attevo* and the operation *opn* out from the schematic evolution model; let ***AI*** be the list of affected IPUs then,

1. For each element *ipu* in the ***AI***, let *ld* be the local definition of *ipu*; ***BV*** be the list of basic views of *ld*.
2. If ***BV*** ≠ Ø, for each element *bv* in ***BV***, let ***CQ*** be the list of conjunctive queries of *bv* and ***R*** be the dependency list of *bv*.
3. If ***R*** ≠ Ø, for each element *r* in ***R***, if *r* is identical with *r*e, attempt to find attribute *att'* in *r* where *att'* is identical with *attori*. If *att'* is found, replace *att'* with *attevo*.
4. If ***CQ*** ≠ Ø, for each element *cq* in ***CQ***, let ***SBG*** be the list of subgoals of *cq*. And for each element *sbg* in ***SBG***, let *sbgtp* be the subgoal type of *sbg* and *asi* be the assignment of the *sbg*.
5. For every *sbg* whose *sbgt* = "*regular*", if the relation *rela* in the *asi* is identical with *r*e, in the list of variable and attribute pair of *asi*, attempt to find the pair whose attribute *asatt* is identical with *attori*. If the pair is found, let *varori* be the variable of the pair and replace the pair with (*varevo*, *attevo*) where the *varevo* is a new variable has same data type with *varori*,
    i. Add a new element *sbgn* into ***SBG***, set its *sbgtp* as "*converting*"; the variable in its head as *varori*; and the operation in its assignment as *opn*
    ii. Replace the *convopd* of *opn* with *varevo*
6. Check whether *bv* is valid against the validation rules. If invalid, remove *bv*
7. Validate all the views in *ld* iteratively and remove the invalid views, then repeat from step 1 until all the *ipu* in the ***AI*** are processed.

The model of attribute domain change uses *opn* to describe the operation for constructing the results data of the original attribute from the results data of the evolved attribute. The variable in the head of the *opn* and the *convopd* of the opn are mapped to the original attribute and the evolved attribute respectively. Through adding a new converting type of subgoal into the conjunctive query which is connected to the relation the evolved attribute belongs to and replacing the original attribute (and its corresponding variable) with the evolved attribute (and its corresponding variable), the original attribute in the conjunctive query can be rebuilt over the evolved local schema. Similarly with the attribute rename, the attribute domain change also only affects the basic views. The above process solves the attribute domain change by firstly updating the relations in the dependency lists, and then rewriting the conjunctive queries to rebuild the original attributes. Finally the basic views and the local definitions of the IPUs are validated.

4) Modification process for attribute decomposition

If the type of the schematic evolution is *"Attribute Decomposition"*, extract the relation *r*e, the original attribute *attori*, the list of evolved attributes ***ATTEVO***; the list of operations ***OPN*** and the list of paired variables and attributes ***VA*** out from the schematic evolution model; let ***AI*** be the list of affected IPUs.

Since the step 1, 2, 4 and 6 are completely the same as the process for solving attribute domain change, only the step 3 and 5 are described below:

3. If ***R*** ≠ Ø, for each element *r* in ***R***, if *r* is identical with *r*e, attempt to find attribute *att'* in *r* where *att'* is identical with *attori*. If *att'* is found, for each element *attevo* in ***ATTEVO***, add *attevo* into *r* and remove duplicates remove *attori*.
5. For every *sbg* whose *sbgt* = "*regular*", if the relation *rela* in the *asi* is identical with *r*e, in the list of variable and attribute pair of *asi*, attempt to find the attribute *asatt* which is identical with *attori*. If *asatt* is found, Let *varori* be the variable correlated with *attori*.
   i. For each *va* in ***VA***, if the attribute of *va* ≠ *attori*, add *va* into *asi* and remove duplicates. If the attribute of *va* = *attori*, let *opv* be the corresponding variable.
   ii. For each element *opn* in ***OPN***, add a new element *sbgn* into ***SBG*** and remove duplicates, set its *sbgtp* as "*converting*" and the operation in its assignment as *opn*. If the variable in the head of *opn* = *opv*, set the variable as *varori*;
   iii. Remove the variable and attribute pair in *asi* where the attribute = *attori*

Similarly with solving the attribute domain change, the process rebuilds the original attribute from the evolved attributes through modifying the local definitions by updating the dependency list of the basic views, adding new converting type of subgoals and correlating the evolved attributes with the appropriate variables based on the correlations specified in the schematic evolution description of the attribute decomposition. And finally validates the local definitions.

5) Modification process for attribute aggregation

If the type of the schematic evolution is *"Attribute Aggregation"*, extract the relation *r*e, the list of original attributes ***ATTORI***, the evolved attribute *attevo*; the list of operations ***OPN*** and the list of paired variables and attributes ***VA*** out from the schematic evolution model; let ***M*** and ***P*** be two empty lists and ***AI*** be the list of affected IPUs then,

As with the attribute decomposition, only steps 3 and 5 of the process are described:

3. If ***R*** ≠ Ø, for each element *r* in ***R***, if *r* is identical with *r*e, add *attevo* into *r* and for each element *attori* in ***ATTORI***, attempt to find attribute *att'* in *r* where *att'* is identical with *attori*. If *att'* is found, put *att'* into ***M*** and remove duplicates. Then remove the *att'* from *r*.

5. For every *sbg* whose *sbgt* = "*regular*", if the relation *rela* in the *asi* is identical with *r*e,
    i. Attempt to find the element *va* in ***VA*** that the attribute of *va* is identical with *attevo*, if *va* is found, add *va* into *asi* and remove duplicates
    ii. For each element *m* in ***M***, attempt to find the pair of variable and attribute in *asi* where its attribute *att'* is identical with *m*. If the pair is found, put it into ***P*** and remove duplicates. Then remove the pair from *asi*.
    iii. For each element *p* in ***P***, let *var'* and *att"* be the variable and attribute of *p*. Attempt to find the element *va* in ***VA*** where the attribute of *va* is identical with *att"*. If *va* is found, reset *att"* of *p* as the variable of *va*.
    iv. For each element *opn* in ***OPN***, add a new element *sbgn* into ***SBG***, set its *sbgtp* as "*converting*" and the operation in its assignment as *opn*. Attempt to find element *p* in ***P*** that the second variable *var"* of *p* is identical with the head variable of *sbgn*, if *var"* is found, reset the head variable of *sbgn* as the first variable *var'* of *p*.

Similarly with solving the attribute decomposition, the above process modifies the local definition through updating the relations in the dependency list to be consistent with the evolved local schema, adding new converting types of subgoals based on the operations specified in the schematic evolution model. And then correlating the attributes with the appropriate variables based on the assignments in the original basic views and the schematic evolution description. Thus the original attributes are rebuilt from the evolved database schema. Finally, the local definition of the affected IPUs are updated and validated.

6) Modification process for relation removal

If the type of the schematic evolution is *"Relation Removal"*, extract the relation *r*e from the schematic evolution model; let ***AI*** be the list of affected IPUs then,

1. For each element *ipu* in the ***AI***, let *ld* be the local definition of *ipu*; ***BV*** be the list of basic views of *ld*.
2. If ***BV*** ≠ Ø, for each element *bv* in ***BV***, let ***CQ*** be the list of conjunctive queries of *bv* and ***R*** be the dependency list of *bv*.
3. If ***R*** ≠ Ø, for each element *r* in ***R***, if *r* is identical with *r*e, remove *r* from ***R***.
4. If ***CQ*** ≠ Ø, for each element *cq* in ***CQ***, let ***SBG*** be the list of subgoals of *cq*. And for each element *sbg* in ***SBG***, let *sbgtp* be the subgoal type of *sbg* and *asi* be the assignment of the *sbg*.
5. For every *sbg* whose *sbgtp* = "*regular*", if the relation *re'* in the *asi* is identical with *re*, remove the *cq*.
6. Check whether *bv* is valid against the validation rules. If not valid, remove *bv*.
7. Validate all the views in *ld* iteratively and remove the invalid views, then repeat from step 1 until all the *ipu* in the ***AI*** are processed.

Similar with the attribute removal, the above process solves the relation removal through firstly removing the relations in the dependency list of the basic views, updating the basic views and checking them against the validation rules. Invalid basic views are removed. Then all the views in the local definition are updated accordingly and validated. If the root view of an IPU is not valid after the modification, the IPU is removed.

7) Modification process for relation rename

If the type of the schematic evolution is *"Relation Rename"*, extract the original relation *reori* and the evolved relation *reevo* from the schematic evolution model; let ***AI*** be the list of affected IPUs then,

1. For each element *ipu* in the ***AI***, let *ld* be the local definition of *ipu*; ***BV*** be the list of basic views of *ld*.
2. If ***BV*** ≠ Ø, for each element *bv* in ***BV***, let ***CQ*** be the list of conjunctive queries of *bv* and ***R*** be the dependency list of *bv*.
3. If ***R*** ≠ Ø, for each element *r* in ***R***, if *r* is identical with *reori*, replace *r* with *reevo*.
4. If ***CQ*** ≠ Ø, for each element *cq* in ***CQ***, let ***SBG*** be the list of subgoals of *cq*. And for each element *sbg* in ***SBG***, let *sbgtp* be the subgoal type of *sbg* and *asi* be the assignment of the *sbg*.
5. For every *sbg* whose *sbgt* = "*regular*", if the relation *rela* in the *asi* is identical with *reori*, replace *rela* with *reevo*.
6. Check whether *bv* is valid against the validation rules. If not valid, remove *bv*.
7. Validate all the views in *ld* iteratively and remove the invalid views, then repeat from step 1 until all the *ipu* in the ***AI*** are processed.

Similarly with attribute rename, the above process firstly updates the relations in the dependency list of the basic views, and then modifies the basic views through replacing the original relation in the assignment of regular subgoals with the evolved relations. Then it validates the basic views and the local definition of the IPUs.

8) Modification process for relation decomposition

If the type of the schematic evolution is *"Relation Decomposition"*, extract the original relation *reori*, the list of evolved relations ***REEVO***, the intermediate view *riv* and the list of basic views ***RBV*** from the schematic evolution model; let ***AI*** be the list of affected IPUs then,

1. For each element *ipu* in the ***AI***, let *ld* be the local definition of *ipu*; ***IV*** and ***BV*** be the list of intermediate views and the list of basic views of *ld* respectively.
2. If ***BV*** ≠ Ø, for each element *bv* in ***BV***, let ***CQ*** be the list of conjunctive queries of *bv* and ***R*** be the dependency list of *bv*.
3. If ***R*** ≠ Ø, attempt to find element *r* in ***R*** where *r* is identical with *reori*, if *r* is found, remove *r* from ***R***. For each element *reevo* in ***REEVO***, add *reevo* into ***R*** and remove duplicates.
4. If ***CQ*** ≠ Ø, for each element *cq* in ***CQ***, let ***SBG*** be the list of subgoals of *cq*. And for each

element *sbg* in ***SBG***, let *sbgtp* be the subgoal type of *sbg* and *asi* be the assignment of the *sbg*.

5. For every *sbg* whose *sbgt* = "*regular*", if the relation *rela* in the *asi* is identical with *reori*,
    i. Let *viewname'* be the view name of *bv*.
    ii. Remove *bv* and for each element *rbv* in ***RBV***, add *rbv* into ***BV*** and remove duplicates.
    iii. Add *riv* into ***IV*** and replace the view name of *riv* as *viewname'*.
6. Validate all the views in *ld* iteratively and remove the invalid views, then repeat from step 1 until all the *ipu* in the ***AI*** are processed.

The above process solves the relation decomposition through firstly updating the relations in the dependency list of basic views according to the original relation and the list of evolved relations represented in the schematic evolution model. Then it removes the basic view connected with the original relation and adds new basic views correlated with the evolved relations specified in the schematic evolution description. It then adds the intermediate view described in the description into the local definition. Since each actual relation is correlated to a basic view (see section 4.4.4.5), each evolved relation is connected to the local definition through adding a new basic view and the original relation is connected to an intermediate view which is defined over the added basic views.

9) Modification process for relation aggregation

If the type of the schematic evolution is *"Relation Aggregation"*, extract the list of original relations ***REORI***, the evolved relation *reevo*, the list of paired original relation and intermediate view ***RRIV*** and the basic view *rbv* from the schematic evolution model; let ***AI*** be the list of affected IPUs.

As steps 1, 2, 4 and 6 of the above process for solving relation aggregation are completely the same as the process of solving relation decomposition, only step 3 and 5 are described.

3. If ***R*** ≠ Ø, for each element *reori* in ***REORI***, attempt to find element *r* in ***R*** where *r* is identical with *reori*, if *r* is found, remove *r* from ***R***. Add *reevo* into ***R*** and remove duplicates.
5. For every *sbg* whose *sbgt* = "*regular*", let *rela* be the relation in the *asi*, attempt to find element *reori* in ***REORI*** which is identical with *rela*. If *reori* is found.
    i. Let *viewname'* be the view name of *bv*.
    ii. Attempt to find element *rriv* in ***RRIV*** that the relation *reo* of *rriv* is identical with *rela*. If *rriv* is found, replace the name of intermediate view *riv* of *rriv* with *viewname'* then add *riv* into ***IV*** and remove duplicates.

    Add *rbv* into ***BV***.

As with solving relation aggregation, the process firstly updates the relations in the dependency list of basic views then adds a new basic view corresponding to the evolved relation. It then replaces the basic views correlated with the original relations with the corresponding intermediate views specified in the schematic evolution model. Hence every original relation is rebuilt from the evolved relation through the intermediate views.

10) Modification process for database removal

The modification process for database removal is different from the nine processes described above. Instead of analysing the affected views and modifying them accordingly, the database removal requires removing all the correlated IPUs from the IDS. Recall the process of affected IPU identification introduced in section 6.4.2.1; the affected IPU list is empty when the schematic evolution is *"Relation Aggregation"*. The modification process is carried out through the follow steps:

1. Let *ipsname* and *sevt* be the IPS name and the evolution type of the schematic evolution model respectively; the ***IPS*** be the list of IPSs in IDS.
2. If *sevt* = "*Database Removal*", for each element *ips* in ***IPS***, attempt to find the name of *ips* which is identical with *ipsname*. If *ips* is found, remove *ips* from ***IPS***.

After all the processes for solving different types of schematic evolution are described in detail above, the modification of affected IPUs can be considered as an automatic process which takes the schematic evolution model and the affected IPU list as input and modifies the IPUs accordingly based on the predefined processes. When the schematic evolution is an addition to the attribute, relation and database level, human intervention is required to integrate the newly-added attributes, relations and databases into the IDS by following the process described in chapter 4. Other types of schematic evolution can be automatically handled without human intervention through the corresponding processes described above, although under the circumstance of removal to the attribute and relation level, the affected IPUs may finally be disposed of.

6.4.3 Schematic Evolution Handling Services

As introduced in section 6.3.2, the evolution handling for organizational evolution is realized through adding a new function right before the first step of the user query processing: the target data sources filtering. Different from handling organizational evolution, schematic evolution handling is realized through the evolution handling services as introduced in section 3.5.3. Conceptually, data sources identify the schematic evolution occurring in them; they represent the evolution by using the data model described in section 6.4.1.2 and send the model to the evolution handling service. The service takes the schematic evolution data model as its input and applies automatic modifications onto the IPUs of the IPSs based on the processes defined in section 6.4.2.2. The service consists of a Locator and a Modifier, the Locator actualizes the process of affected IPU identification and the Modifier actualizes the process of affected IPU modification. In practice, multiple instances of the evolution handling services may be implemented for enhancing efficiency.

6.5 Summary

This chapter described the process for automatically handling the organizational evolution and schematic evolution. The process is a simulation of human effort. Through identifying the evolution, the affected parts of the IDS can be identified. Since the IDS is built on the organizational structure and the IPU mechanism which is realized through DAGs, automatic modifications can be applied based on the predefined processes.

Chapter 7 presents an experimental implementation of the SLEDI, various issues related to the design and the implementation are discussed, followed by a short evaluation.

Chapter 7 Experimental Implementation

7.1 Introduction

Chapter 6 presented the evolution handling in the SLEDI. Through delineating the processes of identifying and solving the various types of evolution, the description of the SLEDI solution and all the algorithms were completed.

This chapter presents the experimental implementation of the SLEDI solution. Various technical issues of the implementation are discussed including the design of each service, the system architecture and the developing and testing environments.

7.2 Services and Processes of the SLEDI

As introduced previously, the SLEDI follows the DaaS approach. The data models and algorithms of the three processes: data source describing, query processing and evolution handling which are described in chapter 4, 5 and 6, are realized through a set of *services*. This section illustrates the constituent components of the services and how the three processes are accomplished through the collaborations among the services.

7.2.1 Services

As introduced in section 3.5, the services of SLEDI conceptually include Information Provision Service (IPS), Registry Service (RS), Broker Service (BS) and Evolution Handling Service (EHS). Each conceptual service plays a certain role in the general architecture of SLEDI and cooperates with the other services to achieve the data integration from autonomous and evolving data sources.

7.2.1.1 Information Provision Service

Chapter 4 described the IPUD algorithm which organizes the data provided by a data source into a set of IPUs. Through the global definitions and the local definitions, the IPUs build up a *mapping* between the application domain ontology and the local database schema of the data source. As the IPUs are accommodated by an IPS, an abstract data model is created which contains the mapping and the local database schema. And the data model is realized as the metadata of the IPS which is illustrated in the figure 7-1 below:

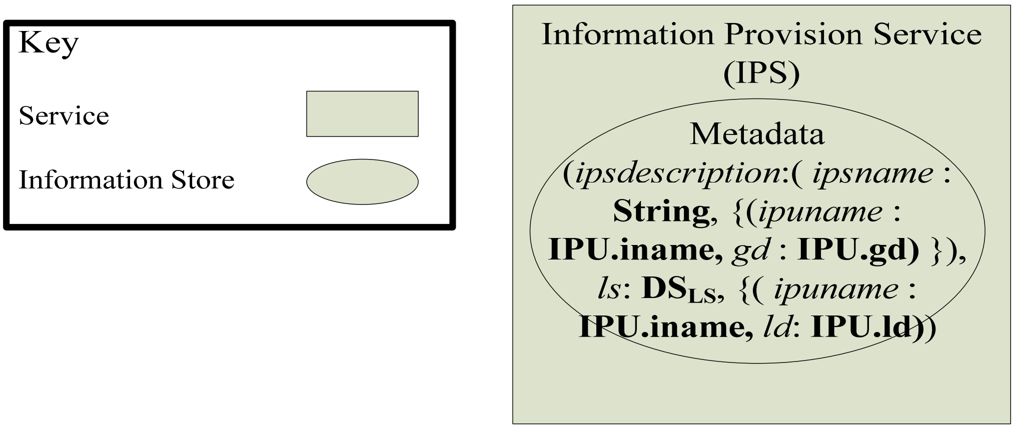

Key
Service
Information Store
Information Provision Service (IPS)
Metadata
(ipsdescription:(ipsname : String, {(ipuname : IPU.iname, gd : IPU.gd) }), ls: DSLS, {(ipuname : IPU.iname, ld: IPU.ld))

Figure 7-1 Metadata of the IPS

The metadata can be regarded as a conceptual data store and may be realized through databases technologies such as a relational database, xml file and etc. The global definitions explicitly describe what data this IPS provides, thus need to be exposed for the Broker Service to access. On the other hand, the local database schema and the local definitions are used to construct results data to respond to the service invocations sent from the Broker Service hence are only accessed by the IPS itself. Thus the IPS publishes its service description which contains the global definitions into the central Registry Service for the Broker service to access.

7.2.1.2 Registry Service

Registry is a central role in SOA (see section 1.3.2). It maintains a central repository for service provider (e.g. IPS) to describe what service it provides and how others can invoke the service through the service description. The service consumer (e.g. BS) accesses the repository to discover the services which satisfy its requirements and interacts with the services through message exchanging. In SLEDI, the Registry Service contains a *registry* and an interface where the registry is a conceptual data store and the interface manages the access activities conducted onto the registry from other services. The data elements of the registry include Application Domain Ontology, Organizational Structure, IPS Service Descriptions and Organizational Evolution which are illustrated in the figure 7-2 below:

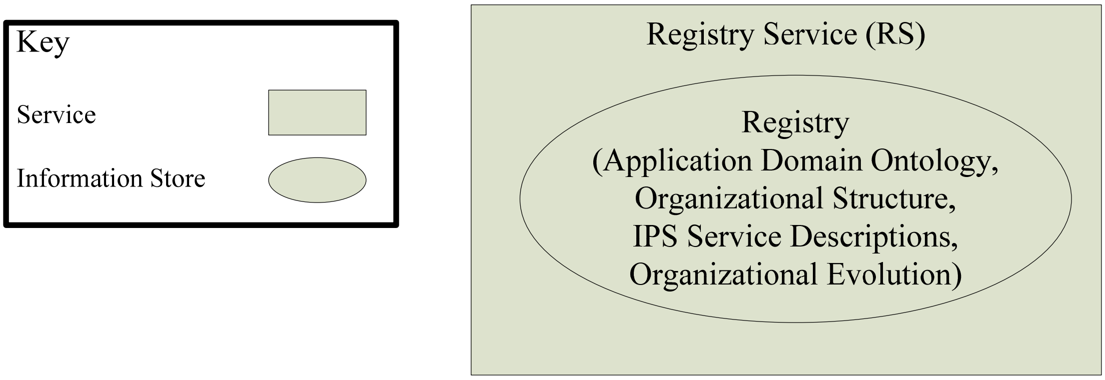

Key
Service
Information Store
Registry Service (RS)
Registry
(Application Domain Ontology,
Organizational Structure,
IPS Service Descriptions,
Organizational Evolution)

Figure 7-2 Registry of the Registry Service

As introduced previously, the application domain ontology models the salient real world objects hence provides an explicit representation of the application domain (see section 4.3). The organizational structure describes how the participating data sources are organized in the structure of DAG. (See section 4.5.2). The organizational evolution keeps all of the atomic organizational evolutions which have ever occurred (see section 6.3.1). Thus by accessing the data elements of the registry, the Broker Service can find the relevant IPSs and communicate with them to construct proper results data hence answering the queries from end users.

7.2.1.3 Broker Service

The Broker Service plays the role of the user interface of the SLEDI. It firstly accepts the queries from end users, then accesses the data elements of the registry introduced in the previous section to discover the relevant IPSs who can provide the answers to the queries. Then it generates the service invocations by accessing the service descriptions of the IPSs and executes the service invocations through exchanging the messages with the IPSs. Finally, the Broker Service collects the results data of the service invocations and combines them together to construct the final results data to answer the queries.

Conceptually, the Broker Service may have to access all the data elements of the registry for answering a single user query. This may lead to frequent data exchanging between the Broker Service and the Registry Service, whilst retaining all the data elements in the registry separates the data and the processing activities of user query answering. Hence only one copy of the data elements needs to be maintained in the SLEDI. Furthermore, the Broker Service and the Registry Service may be implemented through multiple instances and the implementation can be easily replaced with a different version.

7.2.1.4 Evolution Handling Service

As introduced in Chapter 6, the organizational evolutions are handled through firstly identifying the organizational evolutions occurring and recording them into the organizational evolution data element of the registry. And then, the organizational evolutions are solved through the sub-process of query processing to rewrite the user queries which are composed against an old version of organizational structure into equivalent queries with respect to the current version of the organizational structure. Thus the organizational evolution is actually carried out by the Broker Service during the query processing.

The schematic evolutions are handled by firstly identifying each schematic evolution which occurs. The schematic evolution identifying is carried out through the administrator of the data source to describe the evolution using the data model of schematic evolution (see section 6.4.1.2). Then, through accessing the metadata of the IPS the data source provides, the IPUs whose local definition and global definition are affected by the evolution can be located. Finally, the schematic evolution is solved through applying modifications on the metadata. Hence the Evolution Handling Service of the SLEDI is actually specialized in solving the schematic evolution. Each data source hosts an EHS which is invoked by the administrator and communicates with the IPS and RS to accomplish the schematic evolution handling.

7.2.2 Processes

The data source describing process results in the establishing of the IPSs. The process of query processing and evolution handling are carried out through the internal processing of the BS, IPSs, RS and EHSs and the collaborations among them.

7.2.2.1 Query Processing

Chapter 5 introduced the process of query processing consisting of three sub-processes: Data Source Filtering, Query Rewriting and Result Generating. Since the query processing of data integration from distributed data sources is a research area covers many research issues, and the focus of this research is to provide a solution dealing with the evolution of the participating data sources during the data integration, only the parts which are relevant to the evolution are considered. The components of the services and the collaborations among the services for achieving the query processing are illustrated in the Figure 7-3 below:

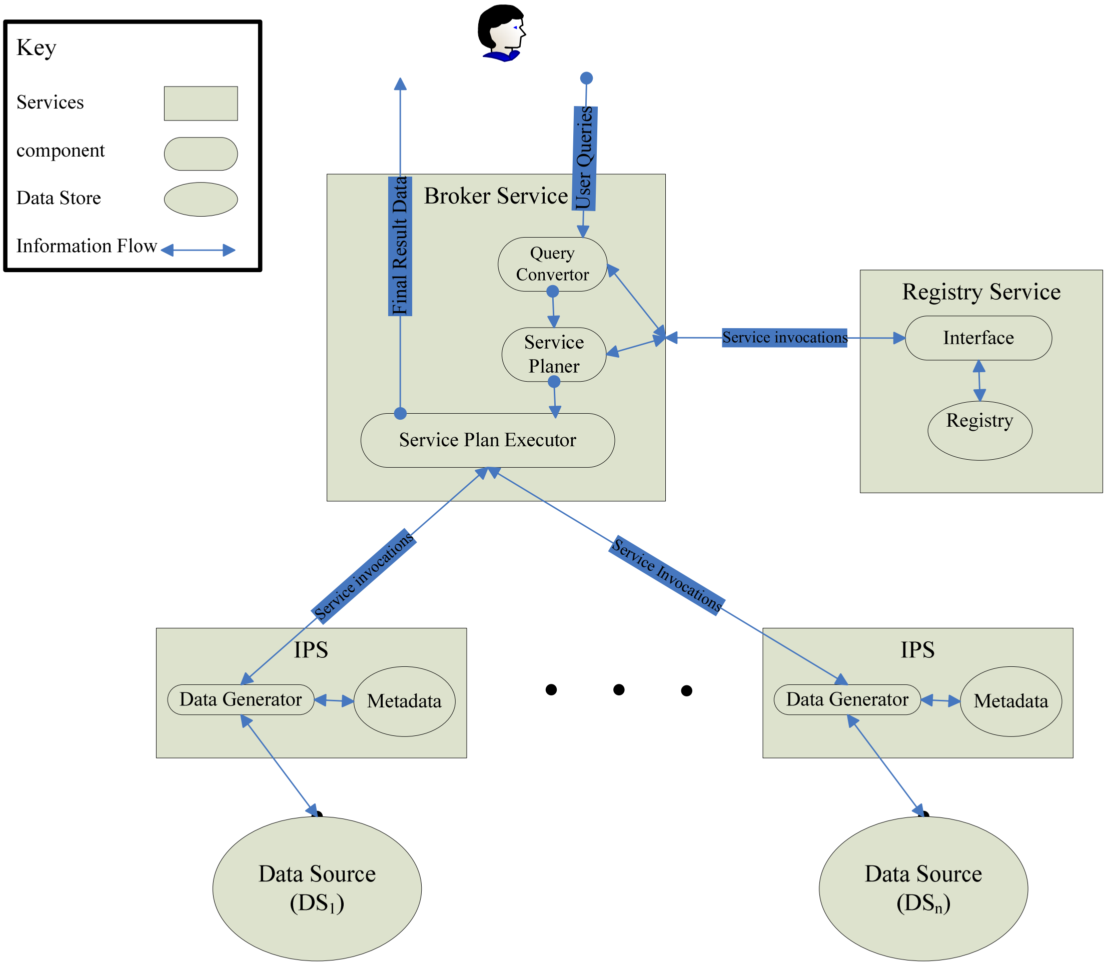


Figure 7-3 Service collaborations for query processing

Each sub-process of the query processing is delivered through one or more components of the services and may involve collaborations among the services as shown above:

1. Data Source Filtering: this sub-process is realized by the Query Convertor component of the Broker Service and the collaboration between the Broker Service and the Registry Service. Once a user query is passed to the Broker Service, the Query Convertor parses the query, communicates with the Registry Service to find out the information of organizational evolution and organizational structure, then the IPSs provided by the targeting data sources specified in the user query are determined.
2. Query Rewriting: this sub-process is also carried out by the Query Convertor and the collaboration between the Broker Service and the Registry Service. Through communicating with the Registry Service, the global definitions in the metadata of the IPSs which are filtered out in the previous sub-process are collected. Then the user query is further rewritten into subqueries in terms of the collected global definitions by using the query rewriting technique.
3. Result Generating: this sub-process is achieved through the Service Planner and the Service Plan Executor components of the Broker Service, the Data Generator component of the IPS

and the collaborations among the Broker Service, Registry Service and the IPSs. After the subqueries are produced in the previous sub-process, the Service Planner communicates with the Registry Service to convert each global definition of a subquery into a service invocation through parsing the service descriptions of the IPS which contains the IPU that the global definition belongs to. Then the Service Plan Executor sequentially executes the service invocations by sending messages to the corresponding IPS. When the IPS receives an invocation, the Data Generator parses the message to determine which IPU the invocation requires, and then accesses the metadata to find out the corresponding local definition and the local database schema to compose a local database query. The Data Generator then conducts the local query onto the database of the data source to generate the results data and send the results data back to respond to the Service Plan Executor. Finally, the Service Plan Executor collects the results data of each service invocation and combines them together to answer the user query.

The conceptual services can certainly be extended by having multiple instance implementations in practice.

7.2.2.2 Schematic Evolution Handling

Chapter 6 introduced the process of Schematic Evolution Handling which consists of two sub-processes: Evolution Identification and Evolution Solving. The components of the services and the collaborations among the services for accomplishing the Schematic Evolution Handling are illustrated in the figure 7-4 below:

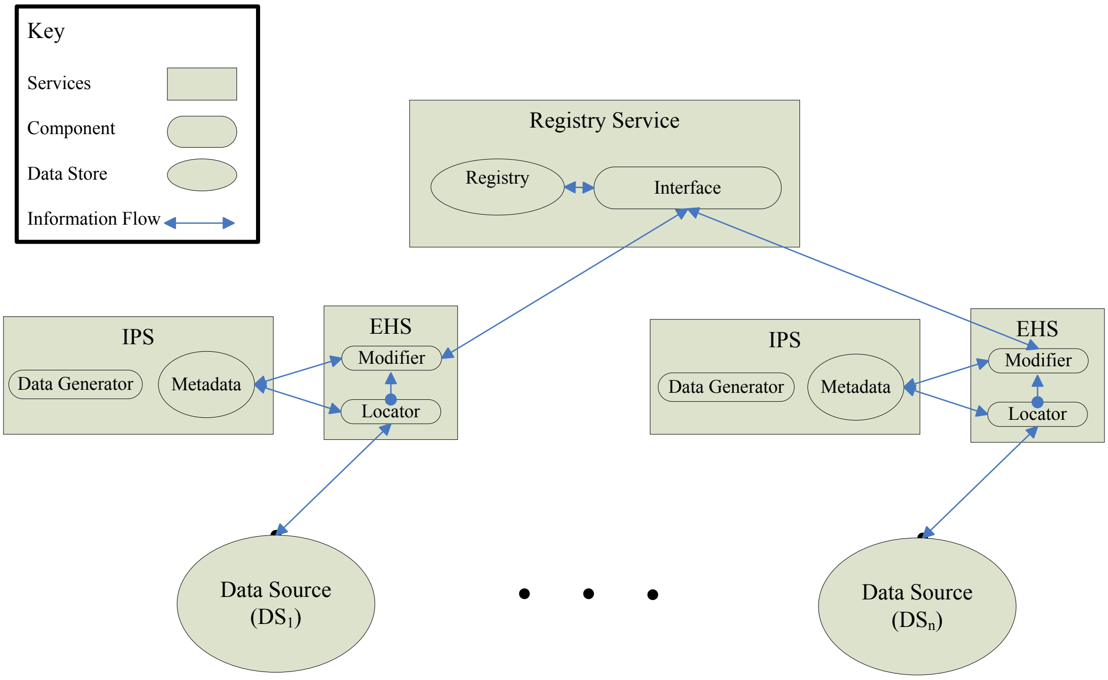


Figure 7-4 Service collaborations for Schematic Evolution Handling

1. Schematic Evolution Identification: this sub-process is delivered by the invocation of the EHS from the data source administrator. Once a schematic evolution occurs in the database of a participating data source, the administrator describes the evolution using the data model of schematic evolution described in section 6.4.1.2 and then invokes the EHS by sending messages containing the description of the evolution and the updated local database schema.
2. Schematic Evolution Solving: this sub-process is carried out by the Locator and Modifier components of the EHS and the collaborations among the EHS, IPS and the Registry Service. Once the EHS is invoked, the Locator parses the message and accesses the metadata of the IPS to determine the local definitions and global definitions which are affected by the evolution based on the algorithm described in section 6.4.2.1. After that, the Modifier conducts modifications onto the metadata by firstly updating the local database schema and then modifies the affected local and global definitions according to the algorithm. If a global definition is modified, the Modifier is then sending messages to the Registry Service to refresh the registry in order to update the IPS description.

Conceptually, the administrator of the data source invokes the EHS for handling the schematic evolution. The EHS then modifies the metadata of the IPS and may communicate with the Registry Service if the service description of the IPS is changed. The EHS may be implemented by multiple instances in practice.

The four conceptual services may be implemented variously in different contexts to better fulfil the system requirements. One conceptual service may be implemented by multiple services instances and two or more conceptual services may be combined into a single service instance. For example, in the experimental implementation of this research, the EHS is segmented into two parts, one part is combined with the Registry Service and another part is combined with the IPS to produce Data Service (DS). The details will be illustrated in the following sections.

7.3 Experimental System

Since the SLEDI is a solution for integrating data from autonomous and evolving data sources, it is impractical to implement a complete data integration system based on the SLEDI in a single research by a single researcher, because it may involve research issues which fall beyond the boundaries of this research such as the security issues of data integration, query translation, query optimization and etc. Consequently, an experimental implementation of SLEDI is implemented for conducting the evaluation of this work. The methods presented in the preceding chapters are embodied in the experimental system. The system is referred to as Service Late Binding Enabled Data Integration System (SLEDIS) for convenience.

As introduced previously, the experimental system is in the mental health application domain.

Since the SLEDI is a service based solution, the experimental system is developed by employing the SOA architecture and Web Service technologies. Thus the Service-Oriented design and development method introduced in [68] is used for developing the SLEDIS. The conceptual services presented in section 7.2.1 can be roughly considered as the result of the service design phase which explicitly describes the business process each conceptual service intends to accomplish. The following phase of the development is the service specifying. Through considering the service coupling, service cohesion and service granularity, it explicitly designates the structural and behavioral specifications of the services such as input and output messages, port types and operations. Thus some conceptual services are coupled as a single service and some processes are combined as a single operation. Consequently, the phase of service specifying results in the abstract definition of the WSDL of each service. The following parts of the WSDL are described:

- definition of all service operations
- definition of input and output messages of each operation
- definition of associated XSD schema types used to represent message payloads

7.3.1 Specification of the Services

7.3.1.1 Broker Service

As mention previously, the Broker Service intends to accomplish query processing through three sub-processes: Data Source Filtering, Query Rewriting and Result Generating, while the Organizational Evolution Solving is embedded in the Data Source Filtering. During the designing of the experimental system, the three sub-processes are combined into one operation: *QueryAnswering*. Because the communications between services are realized through SOAP messages, the data types of the input and output messages of the operation are defined based on XML Schema Definition (XSD). Table 7-1 shows the result of the service specifying of the Broker Service.

Operation	Input (Request Message)		Output(Response Message)	
	Message Name	Type	Message Name	Type
QueryAnswering	QueryRequest	xsd:String	QueryResponse	xsd:complexType

Table 7-1 design of the Broker Service

It is observed that the three sub-processes are working sequentially while the input of the Query Rewriting and Result Generating are dependent on the output of their preceding sub-processes; hence the sub-processes are unlikely to be accessed by other operations individually. By applying the principle that a service should be designed with the characteristics of loose coupling, reusability and autonomy, the operation QueryAnswering is produced through combining the three sub-processes. The input and output messages of the QueryAnswering are also defined in XSD schema. The input message is an xsd:string which is a primitive data type defined by W3C [98]. The message describes a query raised by end users in the form of a conjunctive query over the relations in the application domain ontology. The output message is in the format of xsd:complexType to store the results data for answering the input user query.

One of the problems of a data-intensive service is exchanging the results data of a query between services. The results data cannot be encapsulated into SOAP directly because the results data is normally in the form of relation instances like a two-dimensional table, whereas the messages between the services are in the form of SOAP. Therefore, the results data must be transformed so that it can be carried by SOAP messages. The *DataSet* is a data format which is supported by the .NET Framework and can be used to store the relation instances. More importantly, it can be serialized into XML format and loaded into the body of SOAP messages. Therefore, the results data of each query is stored in the format of DataSet and automatically serialized as the QueryResponse message of the Broker Service. Since the serialization process is automatically managed by .NET Framework, the details of the xsd:complexType of the output message are not discussed further.

7.3.1.2 Data Service

As introduced previously, each data source provides an Information Provision Service at its own site, and the schematic evolution handling process is triggered by the administrator of a data source and carried out through modifying the metadata of the corresponding IPS. By applying the same principle used in the design of the Broker Service, a Data Service is designed for each of the participating data sources. The Data Service combines the functions of the IPS and the schematic evolution handling process of the EHS. Consequently, the metadata in the IPS is adopted into the Data Service. The operations of *InformationProvision* and *SchematicEvolutionHandling* are produced to implement the functions respectively. Table 7-2 shows the result of the service specifying of the Data Service.

Operation	Input (Request Message)		Output(Response Message)	
	Message Name	Type	Message Name	Type
InformationProvision	InfoRequest	xsd:String	InfoResponse	xsd:complexType
SchematicEvolutionHandling	SEHRequest	xsd:complexType	SEDResponse	xsd:String

Table 7-2 design of the Broker Service

The operation of InformationProvision is similar to the QueryAnswering operation of the Broker Service. Both of them take a query described in the string data type as the input message and return the results data in the format of xsd:complexType for answering the query. The difference is that the input message of the Broker Service represents the user query whereas the input message of a Data Service represents a subquery resulting from the query rewriting sub-process which is carried out by the Broker Service. The InformationProvision operation of a Data Service is normally invoked by the Broker Service and the QueryAnswering operation of the Broker Service is usually invoked by an end user.

The operation of SchematicEvolutionHandling is directly derived from the schematic evolution handling process. The input message is a complex type defined in XSD schema which represents the schematic evolution description. Due to the limited space of the thesis, only partial definition of the complex type is shown below, all the trivial details such as name space definitions are omitted. This style is also applied to other complex type definitions in the rest of this chapter.

```
<xsd:element name=”SEHRequest”>
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name=”SchematicEvolutionType” type=”xsd:string”/>
         <xsd:element name=”SchemaEvolutionDescription” type=”xsd:string”/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

The first element of the input message is a string to describe the type of the schema evolution (e.g. “Attribute Removal”), while the second element of the input message is a string that represents the schema evolution description (e.g. “(Relation1, Attribute1)”). The output message is also a string that represents the status of the execution of this operation (e.g. “successful” or “fail”).

7.3.1.3 Registry Service

The Registry Service is derived directly from the conceptual Registry Service. It maintains its metadata (i.e. the registry) and provides an Interface for other services to communicate with. The interface is composed through three operations: *GetReg*, *GetIPSDescription* and *IPSEvolution*. Table 7-3 shows the result of the service specifying of the Registry Service.

Operation	Input (Request Message)		Output(Response Message)	
	Message Name	Type	Message Name	Type
GetReg	RegRequest	xsd:String	RegResponse	xsd:complexType
GetIPSDescription	IPSDRequest	xsd:String	IPSDResponse	xsd:complexType

IPSEvolution	IPSERequest	xsd:complexType	SEDResponse	xsd:String

Table 7-3 design of the Broker Service

As mentioned previously, the metadata of the Registry Service maintains the four data items: application domain ontology, organizational structure, organizational evolution, and IPS descriptions. The operation of GetReg is produced for other service to get access to the first three data items as these data items are normally required to be fully retrieved instead of partially retrieved. The operation is usually invoked by the Broker Service. The input message of the operation is a string to specify which data item the Broker Service intends to retrieve. The output message is a complex type which represents the retrieved data item to respond to the invocation.

The operation of GetIPSDescription is designed for retrieving the location of the IPS and the global definition of the IPUs which are associated with the IPS. The input message of the operation is a string to represent the name of the IPS that the Broker Service intends to retrieve. The output message is a complex type which represents the retrieved IPS description. The details of the complex type for each data item will be discussed in later sections.

The operation of IPSEvolution is derived from part of the schematic evolution solving process. As introduced in section 6.4.2.2, the global definition of the IPUs may require modification under some circumstances. Since the global definition of the IPUs are embedded in the IPS descriptions and the registry maintains all the IPS descriptions. The IPSEvolution operation is designed for modifying the IPS descriptions. The input message of the operation is a complex type which is shown below

```
<xsd:element name=” IPSERequest”>
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name=”IPSName” type=”xsd:string”/>
         <xsd:element name=”IPUName” type=”xsd:string”/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Since the only modification action applied onto a global definition is to delete it (see section 6.4.2.2), the first and the second elements of the input message are both strings to respectively specify the names of the IPS and the IPU which are associated with the global definition requiring deletion.

7.3.2 Architecture of the SLEDIS

The previous section introduced the service design of the SLEDIS by following the Service-Oriented design method. Since the service design only cover the abstract definition of the services, this section describes the design of the internal components and process of each services based on the Object-Oriented design method. Figure 7-5 shows the architecture of the SLEDIS in terms of its databases, services and the clients. The diagram demonstrates the internal classes and process of the Broker Service, Data Service and Registry Service. The input and output messages of each service and the messages exchanging activities are omitted

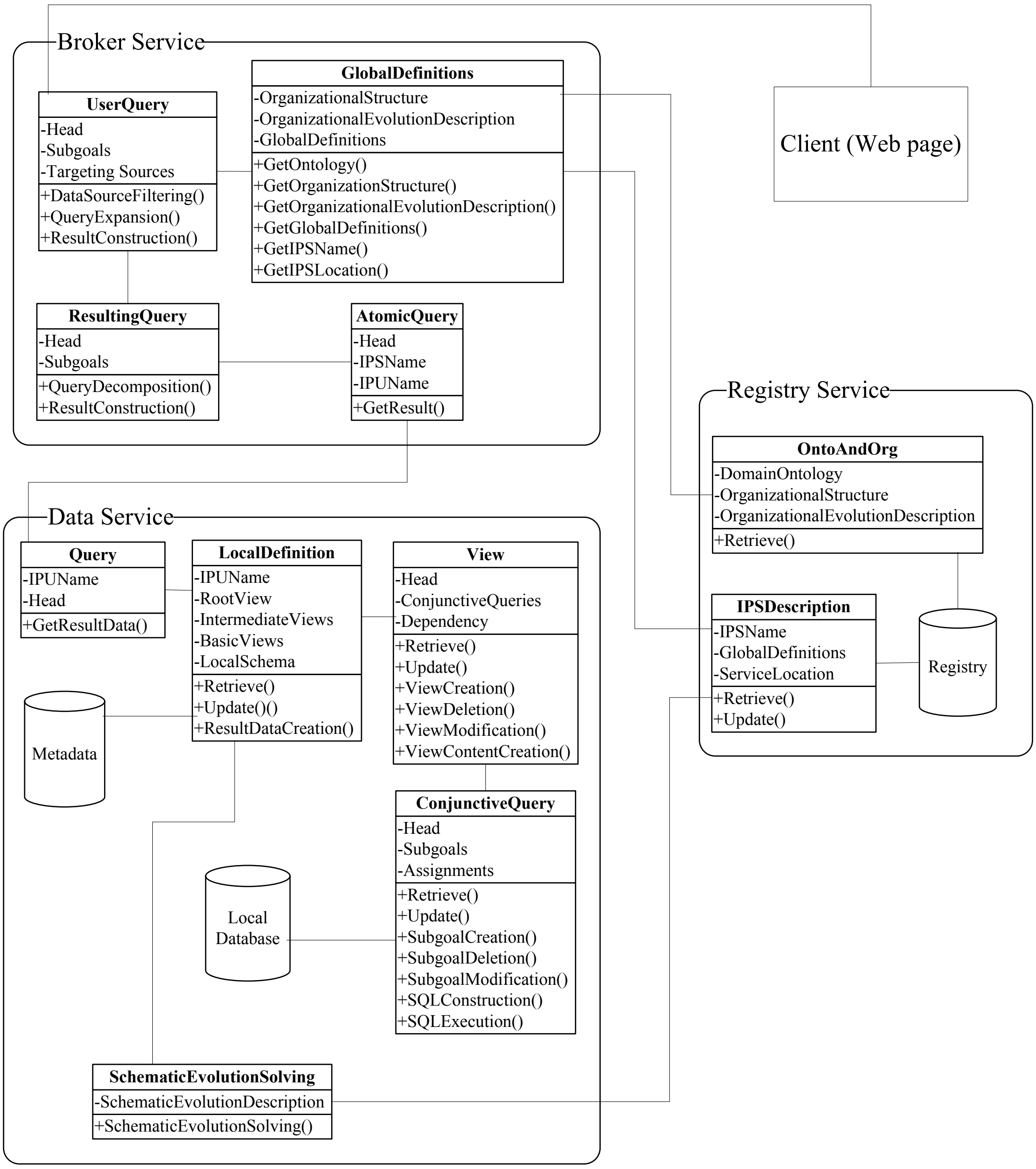


Figure 7-5 Architecture of the SLEDIS showing classes and databases

This architecture reflects the design of the services in SLEDIS.

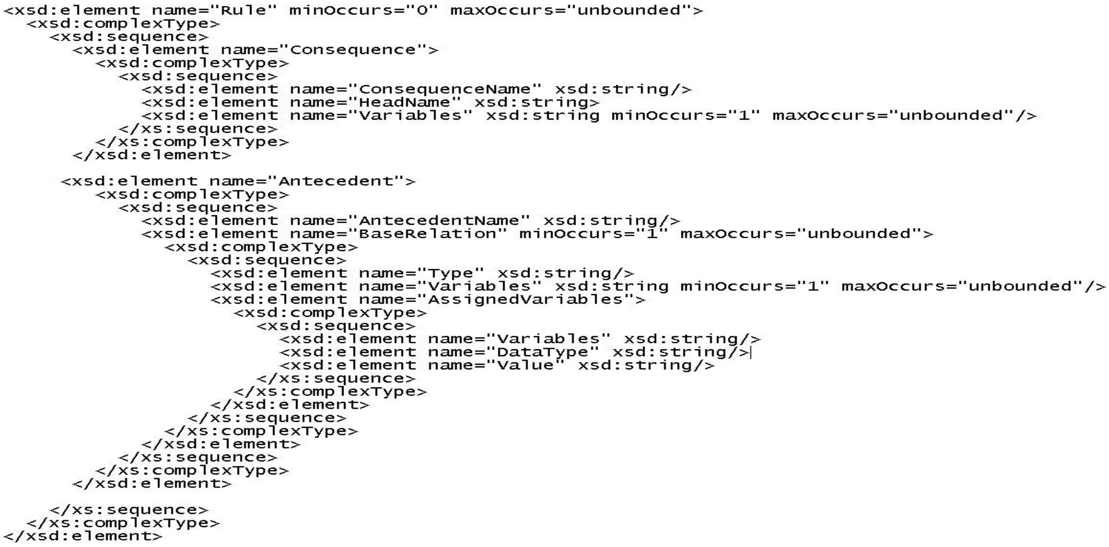
- In the Broker Service, the classes of UserQuery, GlobalDefinitions, ResultingQuery and AtomicQuery encapsulate the sub-processes of Data Source Filtering, Query Rewriting and Result Generating.
- In the Data Service, the classes of Query, LocalDefinitions, View and ConjunctiveQuery encapsulate the results data construction for answering the subqueries delivered to the Data Service. The SchematicEvolutionSolving class and the LocalDefinition class encapsulate the part of the schematic evolution solving process which is performed for modifying the local definitions.
- In the Registry Service, the OntoAndOrg class and the IPSDescription class provide the information for the Broker Service to perform the query processing. The IPSDescription class encapsulates the part of the schematic evolution solving process which is performed for modifying the global definitions.

In the implementation, a web page is developed to accept user queries and display the results data obtained from the Broker Service. The Metadata in the Data Service (i.e. the metadata in the IPS) and the Registry in the Registry Service are implemented in XML data format. The local database represents the actual DBMS of the data source. The above design of the SLEDIS has also been influenced by a number of considerations:

1. Service Reusability: All the Data Services have the unified operations which receive subqueries delivered from the Broker Service in the format of a string. Hence when new databases need to be published as Data Services, no programs are required.
2. Service Loose coupling: The Broker Service implements most of the business logic of the query processing while maintaining no metadata. Hence the Broker Service can be implemented by multiple instances and each instance can be easily substituted by a different implementation.
3. Optimization of results data construction: Data Service not only maintains the local definitions, but also builds materialized views to help the results data construction.

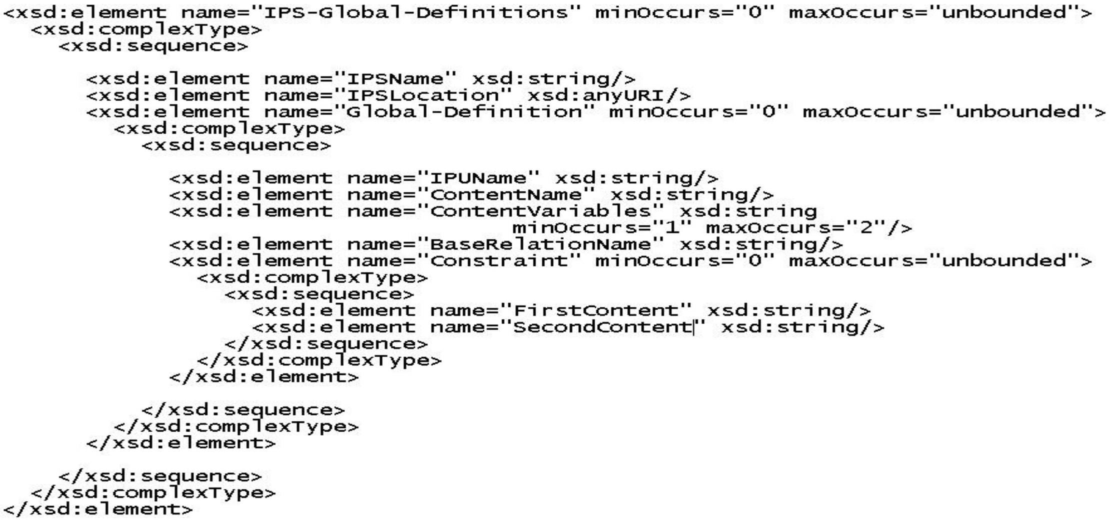
7.3.3 Metadata structure and management

As introduced previously, both Data Service and Registry Service have their own metadata which contains a number of data elements. The metadata are realized in XML format through XML files and managed at the sites of the corresponding services. The structure of the metadata is illustrated through the XSD schemas in the following figures. Due to the limited space of the thesis, only some parts of the metadata structure are depicted in the following figures for demonstration purpose.



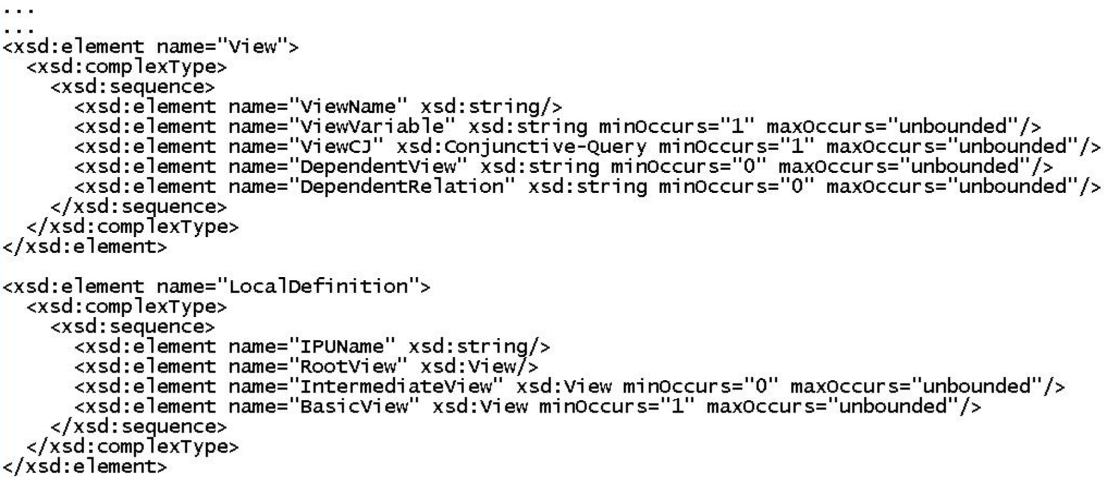
```
<xsd:element name="Rule" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Consequence">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="ConsequenceName" xsd:string/>
            <xsd:element name="HeadName" xsd:string>
            <xsd:element name="Variables" xsd:string minOccurs="1" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xsd:element>

     <xsd:element name="Antecedent">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="AntecedentName" xsd:string/>
            <xsd:element name="BaseRelation" minOccurs="1" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="Type" xsd:string/>
                  <xsd:element name="Variables" xsd:string minOccurs="1" maxOccurs="unbounded"/>
                  <xsd:element name="AssignedVariables">
                    <xsd:complexType>
                      <xsd:sequence>
                        <xsd:element name="Variables" xsd:string/>
                        <xsd:element name="DataType" xsd:string/>|
                        <xsd:element name="Value" xsd:string/>
                      </xs:sequence>
                    </xs:complexType>
                  </xsd:element>
                </xs:sequence>
              </xs:complexType>
            </xsd:element>
          </xs:sequence>
        </xs:complexType>
      </xsd:element>

    </xs:sequence>
  </xs:complexType>
</xsd:element>
```

Figure 7-6 The Rule component of the application domain ontology



```
<xsd:element name="IPS-Global-Definitions" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>

      <xsd:element name="IPSName" xsd:string/>
      <xsd:element name="IPSLocation" xsd:anyURI/>
      <xsd:element name="Global-Definition" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>

            <xsd:element name="IPUName" xsd:string/>
            <xsd:element name="ContentName" xsd:string/>
            <xsd:element name="ContentVariables" xsd:string
                                     minOccurs="1" maxOccurs="2"/>
            <xsd:element name="BaseRelationName" xsd:string/>
            <xsd:element name="Constraint" minOccurs="0" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="FirstContent" xsd:string/>
                  <xsd:element name="SecondContent|" xsd:string/>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>

          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 7-7 Global Definition



```
...
...
<xsd:element name="View">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="ViewName" xsd:string/>
      <xsd:element name="ViewVariable" xsd:string minOccurs="1" maxOccurs="unbounded"/>
      <xsd:element name="ViewCJ" xsd:Conjunctive-Query minOccurs="1" maxOccurs="unbounded"/>
      <xsd:element name="DependentView" xsd:string minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="DependentRelation" xsd:string minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="LocalDefinition">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="IPUName" xsd:string/>
      <xsd:element name="RootView" xsd:View/>
      <xsd:element name="IntermediateView" xsd:View minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="BasicView" xsd:View minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 7-8 Local Definition

As introduced in chapter 4, the application domain ontology contains the *terminolog*y part and the *rule* part. The terminology constitutes description logic expressions and is realized through

Web Ontology Language (OWL). The OWL is a specification developed and recommended by W3C to represent DL based ontology and RDF/XML based serialization of the ontology. The details of OWL can be found in [66]. The rule part of the application domain ontology, organizational structure and organizational evolution is realized as XML format and the XSD of the rule part is depicted in figure 7-6. Similarly, the global definition and local definition of the IPUs are also implemented in XML format and the XSD schemas of them are depicted in figures 7-7 and 7-8 respectively.

The software tool XML Notepad 2007 is used to build the metadata. Although various XML editing software tools are available for helping to build the metadata, they can only provide simple help with editing the metadata, the content of the metadata must be constructed and validated manually before being realized into the XML files. The metadata constructing would need to be rectified if the system were released for large-scale use. However, the current situation is acceptable for the evaluation purpose.

7.3.4 Development and testing environment

As introduced previously, the experimental implementation system SLEDIS implements the SLEDI solution to integrate participating databases in the mental health application domain. The SLEDIS implements the Broker Service, Registry Service, and Data Services and runs on Microsoft Windows XP professional operating system. The development of the SLEDIS was undertaken in the following environment:

- Integrated Development Environment (IDE): Microsoft Visual Studio 2008
- Programming Language: C# 2008
- Programming Platform: .Net Framework 3.5
- Web Service developing toolkit: ASP.Net 2008
- Web Server: Internet Information Service (IIS) 5.1
- Database Management System: Microsoft SQL Server 2005

The IDE is supported by the .Net framework which is a software framework released by Microsoft. It includes a large library of code solutions and a Common Language Runtime (CLR) infrastructure to manage the execution of the programs written in various programming languages that .Net framework supports. The IDE was chosen for several reasons:

- It is currently a major IDE for supporting developing web service based software solutions.
- It supports a number of programming languages (e.g. C#) for the developer's convenience.
- It provides a development and runtime environment which major components of the web service such as WSDL and SOAP message can be easily realized.
- Database connectivity is very well supported.
- Easy testing and debugging are supported by the friendly user interface of the IDE.

C# is a programming language developed within the .Net initiative by Microsoft hence is well supported by the .Net framework. It supports multiple programming diagrams such as Object-Oriented Programming (OOP), hence can be used to realize the design of the services as introduced in section 7.4.2. ASP.Net is a web application framework developed based on the CLR to allow programmers to build web related applications such as web sites and web services. It allows programmers writing ASP.Net code using any of the programming languages supported by .Net such as C#. Furthermore, ASP.Net supports the processing of the web service components such as WSDL and SOAP hence allowing easy development and deployment of the web services.

In addition, the SLEDIS was tested under the following hardware environments:

- Computer: Two desktop PCs with the same hardware configuration
- Processor: Pentium 4 (3.20 GHz)
- RAM: 2 GB
- Hard Drive: 320GB

The two PCs were connected through a local Ethernet network running at 10 megabytes per second (10Mbps). One PC was used to host all the Data Services and publish the service descriptions into the registry. Another PC was used to host the Broker Service and the Registry Service.

7.3.5 Implementation of the Services

This section introduces the implementation of the services through describing the designed classes and methods of each service, as illustrated in figure 7-5 in section 7.3.2

7.3.5.1 Registry Service

In a typical scenario of web service interaction, the Registry maintained by the Registry Service only hosts the service descriptions of the services (i.e. the data services in the case of SLEDI). The Registry in SLEDI also contains the application domain ontology, organizational structure and organizational evolution. Thus the Registry Service also needs to provide assistance for other services to access that information. The following classes are designed:

- *OntoAndOrg*: this class represents the data elements of the application domain ontology, organizational structure and the organizational evolution stored in the Registry. It provides a method for retrieving the data elements and will be invoked by the Broker Service.
- *IPSDescription*: this class represents the description of Data Service stored in the Registry. It is responsible for retrieving and updating the descriptions. The retrieving will take place when it is invoked by the Broker Service to obtain location and global definitions of a Data

Service. The updating will be performed when it is invoked by a Data Service when the name, location and global definitions of the Data Service require modification.

Only one Registry Service is implemented; other services can perform the actions of service publishing and discovering through invoking the Registry Service. The invocation is realized through knowing the service description of the Registry in advance and binding to the service in hard coded programs.

7.3.5.2 Broker Service

The Broker Service may have multiple instances in practice and the following classes are designed:

- *GlobalDefinitions*: this class represents the data elements retrieved from the Registry Service. It has two roles to play. The first role is to retrieve the application domain ontology, organizational structure and organizational evolution for the data source filtering process to use. The second role is to retrieve the service descriptions of the relevant Data Services determined by the data source filtering.
- *UserQuery*: this class represents a user query received from an end user through the user interface (a simple web page in this case). It has three methods of performing the processes of data source filtering, query expansion and result construction.
- *ResultingQuery*: this class represents a resulting query produced by the query expansion. It is responsible for decomposing the resulting query into subqueries and the results construction of the resulting query.
- *AtomicQuery*: this class represents a subquery produced by the resulting query decomposition, where each subquery corresponds to a global definition. It is responsible for invoking the corresponding Data Service to obtain the results data of the subquery.

As the case study aims to examine whether SLEDIS can solve the evolution problems, the parallel query processing is not considered in the implementation. In addition, the query rewriting algorithm described in chapter 5 is implemented without considering the optimization. After the resulting queries are produced, each resulting query is decomposed into subqueries where each subquery corresponds to the global definition of a single IPU and is referred to as an atomic query. The atomic queries are evaluated sequentially through invoking the Data Service the IPU corresponds to. The results data of each resulting query is obtained through combining the results data of all its subqueries into a temporal data table. After the resulting queries are evaluated in turns, the final results data of the user query can be constructed in the same way. Although this may increase the data transferred between the Broker Service and the Data Services, it enables the investigator to check whether each Data Service is able to provide the correct results data especially in the circumstances of schematic evolution occurring.

7.3.5.3 Data Service

Each participating database implements one Data Service and all the Data Services have the same service interfaces and classes, as introduced previously. The following classes are designed:

- *LocalDefinition*: this class represents the local definition of an IPU stored in the metadata of the Data Service. The result data corresponds to the local definition is maintained as a materialized view in the local database. This class is responsible for retrieving the view data to respond to the requests from the Query class and updating the view data to respond to the requests from the SchematicEvolutionolving class.
- *Query*: this class represents the atomic query sent from the Broker Service through service invocation. It is responsible for accessing the corresponding local definition based on the IPU name specified in the atomic query and sending the results data back to respond to the service invocation.
- *View*: this class represents a view defined in the local definition. It is responsible for maintaining the definition and the content data of the view.
- *ConjunctiveQuery*: this class represents a conjunctive query of a view. It is responsible for maintaining the definition and the results data of the conjunctive query. The results data is produced through constructing an SQL query based on the definition of the conjunctive query and evaluating the SQL query against the local database.
- *SchematicEvolutionovling*: this class is responsible for implementing the schematic evolution handling processes. It locates affected local definitions based on the schematic evolution descriptions provided by the local database administrator. It then applies the modifications based on the algorithms described in section 6.4.2.2

As introduced previously, the complete translation of an arbitrary conjunctive query into an SQL query, which can be evaluated directly against the local databases, can be very complex hence is not suitable for this single case study. Therefore, each view in the local definition is materialized and maintained in the local database. As a consequence, each subgoal of a conjunctive query is assigned to either a schema relation or a materialized view. Thus constructing a "select-from-where" SQL query from the conjunctive query is much easier to undertake. Although it requires more space in the local database to store the views and the views may need to be rebuilt after local definition is modified, the atomic query sent from the Broker Service can be answered more efficiently as the results data are already maintained in the root view of the corresponding local definition. Furthermore, it is easier for the investigator to check whether the views are still providing the correct results data after the schematic evolution handling has taken place.

7.4 Test Data

After the experimental system is successfully developed, the SLEDIS can be applied into a specific application domain to further conduct the evaluation of the solution. As introduced in section 2.6, the mental health application domain demonstrates the data integration requirements with the evolution and heterogeneity problems, as defined in chapter 1. The patient information is held by different data sources such as hospitals and social services. The data sources realize the information in different database schemas and store the information in their own local databases. In addition, the data sources are organized by their geographical locations. Applying the SLEDIS into the application domain aims to allow end users to find the patient information from the various entities. Thus whether the SLEDI solution can solve the evolution and heterogeneity problems it is designed for can be examined through the case study which will be discussed in details in the next chapter.

In order to conduct the case study, the test data from the mental health domain is designed for the case study. It is assumed that every data source realizes the patient information in its own database schema. The patient information is represented in the E-R model and illustrated in the following graphs:

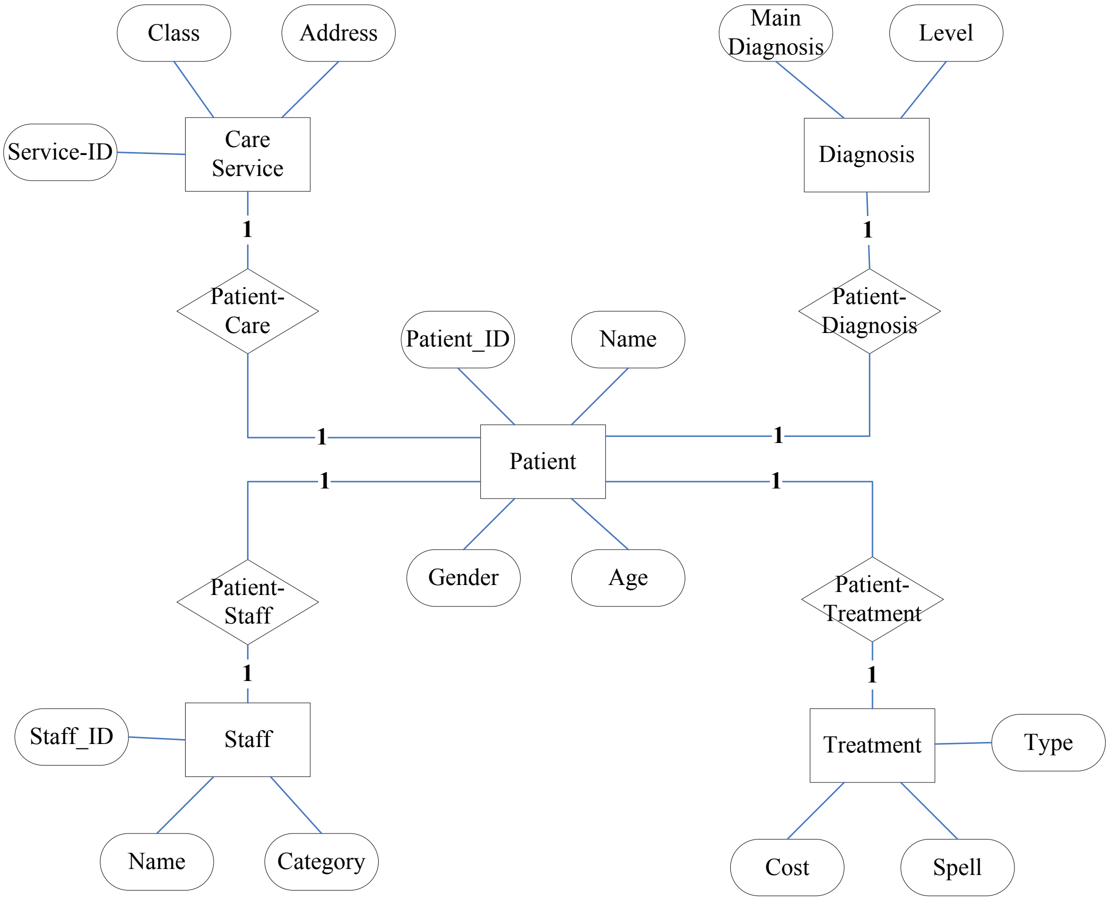


Figure 7-9 The E-R model of Patient Information

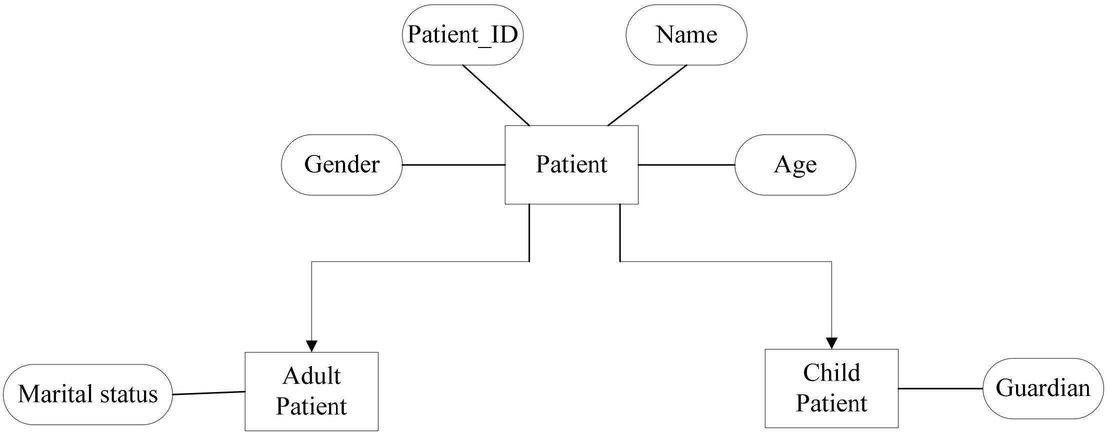


Figure 7-10 Subclasses of Patient Information

As shown in figure 7-9, the patient information involves five entities, four relationships and a number of attributes. A patient may attend a care service and the care service may be a hospital or a social service represented by the attribute “class” of the care service. A patient may be assigned to a staff who may be a doctor or a social worker. In addition, a patient may be associated with a diagnosis and a treatment. The diagnosis identifies the illness of the patient through the main diagnosis and the level of severity. A treatment has an attribute spell to describe the treatment activities have been taken by the patient in a certain period of time. The patient entity has two subclasses indicating whether a patient is either an adult patient or a child patient, as shown in figure 7-10.

The patient information is realized in the application domain ontology and a number of different local database schemas. The details will be illustrated in the next chapter. Although the test data is much simpler than the practical situation, it is able to cover the heterogeneity and evolution problems defined in this thesis thus is enough for the case study. The design of the test data is influenced by a number of considerations:

- The information modelled (i.e. the patient information) should be general information in the application domain which can be understood by the data sources in the domain and may be managed in the participating databases.
- Each data source realizes the general information into its own local database schema differently. The local database schemas should be able to cover all the heterogeneities defined in section 1.2.3.
- An evolution plan should be designed to cover various types of evolution defined in this research and to perform each evolution onto different databases to cover many possible cases.
- A set of user queries should be devised to examine whether the relevant Data Service for

answering the queries can be correctly found and the results data can be correctly constructed especially after the evolution handling has taken place.

Due to the nature of the evolution problems, the evolutions of the participating databases are manually conducted. Plain text log files are also used to store the outputs of each process and to compare them with the pre-defined correct answers in order to obtain the results for the measures. The details of the test data and the case study will be further discussed in the next chapter.

7.5 Evaluation of the Implementation

This section discusses some of the issues arising from the implementation of the experimental system SLEDIS. The SLEDI solution itself will be evaluated in the next chapter.

7.5.1 Design Evaluation

The methods of using the Service-Oriented approach for designing the service specifications and the Object-Oriented approach for implementing the services have been proved effective. The system architecture has not been changed for any version of the SLEDIS although some of the methods of the classes have been modified over time. The Service-Oriented approach helps to clearly specify the operations and messages of the services correctly hence reducing the modification effort afterwards. The Object-Oriented approach helps to accurately design the internal structure of each service. Because the SLEDIS is implemented using C# which is an Object-Oriented programming language, the programming effort are also reduced thanks to the design. Furthermore, the design also mitigates the programming effort for expanding the system through adding more data services.

There were some disadvantages, in particular, the implementation of the query processing. The implementation of the query rewriting sub-process considers little optimization; a resulting query produced from the query rewriting algorithm generates a ResultingQuery object. The large number of the objects may use more computing resources. However, it made the program elegant and easy to check whether the correct resulting queries are produced. In addition, each atomic query is answered through accessing the corresponding data service and all the data services are accessed sequentially. It may to some extent increase the network communication costs, especially when the number of the IPUs provided by a data service and/or the amount of the data services becomes very large. Nevertheless, since the case study is focusing on the evolution problem, not on the performance, the current situation is acceptable.

Each data service maintains a number of materialized views according to the local definitions of the IPUs it provides. It may increase the spatial cost of the local database especially when the local definitions of the IPUs involve more views and/or the number of the IPUs is large. However, it saves the computational cost on the database connections when obtaining the results data of the IPUs. Furthermore, no validation check on input data is provided, hence the investigator needs to ensure the input data is valid. However, it was not a problem after a complete set of test data was produced.

7.5.2 Test and Validation

Each class was tested individually before being integrated into the system. Slight modifications were made onto the program code of each class during the testing in order to visualize the input and output data. Since the interactions between the classes are sequential (i.e. the output of a class is used as the input of another class), the classes were tested in sequence. Consequently, there were very few problems found during integrating the classes into the system. In addition, each service was also tested though checking its input and output messages. The checking was mainly undertaken through verifying whether the output of a class or a service was as expected. For example, the resulting queries produced from a user query and the data services found relevant to a resulting query were compared with a manually performed analysis. The modified views in the local definition conducted by the evolution handling process were compared to the results produced by hand.

The UserQuery class and the GlobalDefinitions class of the Broker Service, and the LocalDefinition class of the Data Service were checked through using the debugging tools provided by the IDE due to their complexity. It helped to display the state of the various fields at the appropriate points during the executions. Single-step debugging was also used to verify the correctness of the implementation of the algorithms. SQL queries were also printed out to ensure the correctness before its execution. Although it is not feasible to test the QueryExpension of the UserQuery class with a large number of data services, the experiments with the well designed typical test data are enough to confirm the correctness of the query expansion even with a large number of data services.

7.6 Summary

This chapter has presented the design of the architecture and the services of the experimental system (i.e. the SLEDIS). Various technical issues relating to the design and implementation of the SLEDIS were discussed and a short evaluation of the implementation was also presented.

Chapter 8 presents an extensive evaluation of the SLEDI solution. The algorithms of the SLEDI are evaluated through verifying the results of the case study. The main focus is on the capability of solving the evolution and heterogeneity problems. Various other characteristics of the SLEDI are also discussed.

Chapter 8 Evaluation

8.1 Introduction

Chapter 7 described an experimental implementation system SLEDIS, various issues related to the implementation of the system, such as the design of the architecture, the services and the metadata structure were discussed.

This chapter presents an extensive evaluation which covers various characteristics of the SLEDI solution. The evaluation is conducted through firstly defining the research questions and corresponding propositions of the case study and then obtaining the results of the measures to examine the propositions. The results are then discussed.

8.2 Case Study

For the purpose of evaluating the SLEDI solution presented in this research, a single case study has been conducted in the Mental Health application domain based on the experimental implementation system. Case study is a research method which is commonly applied in social science research and can be adopted in software engineering field [48, 47]. Some pioneering researches have presented general guidance on how to apply the case study method appropriately to software engineering research [48].

8.2.1 Context and Analysis Unit

Since the purpose of this case study is to evaluate the SLEDI solution for integrating data from autonomous and evolving data sources, the objective is to investigate whether the SLEDIS is able to solve the evolution problems defined in this thesis while integrating data from distributed and heterogeneous databases. It is assumed there are various entities in the mental health application domain such as hospitals, social services and etc. Each entity provides partial information about patients such as their names, addresses, diagnoses and etc. The entities are organized by their geographic locations and each entity is considered as a data source which publishes its database as a Data Service (DS) and registers the DS into the registry. Thus, the case study is a single project case study with a single analysis unit. The context and analysis unit of the case study is:

- Context: mental health application domain
- Analysis Unit: the experimental implementation system (SLEDIS)

8.2.2 Research Questions and Propositions

In order to conduct the case study effectively, the research questions and propositions of the case study have to be clearly defined. The objective of the SLEDI is to provide a solution which is able to integrate data from distributed and heterogeneous databases while reducing the maintenance costs caused by the evolution of the participating databases. The main research questions of the case study are defined below:

- How and why the IPU mechanism can integrate distributed databases with the heterogeneities defined in section 1.2.3
- How and why the IPU mechanism and the metadata of the IPS can help to solve the evolution problems defined in this thesis
- How and why the evolution handling process can help to solve the evolution problems defined in this thesis
- How and why the service based solution (i.e. the SLEDI) can help to solve the evolution problems defined in this thesis

Based on the research questions, the propositions of the case study can be further defined below:

1. The heterogeneity problems defined in section 1.2.3 can be solved in the data integration through the IPU mechanism and the query processing.
2. The IPU mechanism and the metadata of the IPS and BS can reduce the modification effort caused by the schematic evolution occurring in the participating databases.
3. The organizational evolution handling in the query processing can reduce the modification effort made to the user queries caused by the organizational evolution.
4. The schematic evolution occurring in one participating database will not affect the metadata of the IPSs provided by other data sources hence the system can still work properly.
5. The EHS can reduce the cost of the modification effort caused by the schematic evolution.
6. The service late binding based on the SOA and web service can help to reduce the maintenance costs caused by the evolution defined in the thesis.

Based on the propositions, the measures for verifying the propositions are defined in the six tables from 8-1 to 8-6 below. Each table corresponds to one proposition.

Measure	Explanation
Correctness of the result.	The correctness of the results data constructed for answering a user query.

Table 8-1 Measure for Proposition 1

Measure	Explanation
Number of user queries explicitly specifies the local database schemas.	The number of user queries which explicitly specify the actual relations and attributes of the local database schemas they intend to query against.
Number of hard coded queries in mappings.	The number of hard coded queries established for integrating the local database schemas of the participating data sources.

Table 8-2 Measures for Proposition 2

Measure	Explanation
Number of user queries explicitly specifies the participating databases.	The number of user queries which explicitly specify the actual names of the local database schemas they intend to query against.
Number of the affected user queries.	The number of existing user queries which are affected by the evolution which occurs.

Table 8-3 Measures for Proposition 3

Measure	Explanation
Number of databases considered.	The number of participating databases which need to be considered when a single schematic evolution occurs.

Table 8-4 Measure for Proposition 4

Measure	Explanation
Number of IPUs which require modification effort.	The number of the IPUs whose local definition and/or global definition require modification when a schematic evolution occurs.
Human intervention.	Whether an evolution handling process requires human intervention.
The effort of identifying the IPUs affected.	The work to find the IPUs which are affected by a schematic evolution.

Table 8-5 Measures for Proposition 5

Measure	Explanation
The effort of modification on the IPUs.	The work to modify the IPUs which are affected by a schema evolution.

Table 8-6 Measure for Proposition 6

The case study is carried out with the aim of answering the research questions through verifying the propositions as defined above. Consequently, if the propositions are well supported by the results of the case study, the SLEDI solution constructed in this thesis is generally considered to be successful, hence the aim of this research is achieved.

8.3 Overview of the Evaluation

The evaluation begins with one of the most important properties of the SLEDI: the capability of solving the evolution and heterogeneity problems in the data integration. Since the evolution problems are set in the context of integrating data from schematic heterogeneous databases as defined in chapter 1, the capability of solving the heterogeneity problems is evaluated first. Then the capability of solving evolution problems in the data integration is evaluated, which is the focus of this research. The issues relating to the query processing are also discussed. The verification of the propositions is illustrated in detail with the relevant data evidences throughout the evaluation. Furthermore, some general characteristics of the SLEDI solution such as scalability, expandability, language independence and application domain independence are also discussed.

As introduced in section 7.4, a set of test data is designed for undertaking the case study. The patient information was presented in the E-R model in figures 7-9 and 7-10. Initially, it is assumed there are four data sources denoted as $\mathbf{DS_1}$, $\mathbf{DS_2}$, $\mathbf{DS_3}$ and $\mathbf{DS_4}$. The corresponding databases of the data sources are denoted as $\mathbf{DB_1}$, $\mathbf{DB_2}$, $\mathbf{DB_3}$ and $\mathbf{DB_4}$. Each data source model the patient information into its local database schema differently. Although only four data sources are involved, the local schemas of the data sources are designed to cover all the heterogeneity problems defined in section 1.2.3. Each data source publishes a Data Service for evaluating the capability of solving the evolution and heterogeneity problems, and then more data sources are added hence more data services are published to evaluate the scalability of the system. In addition, the patient information is also modelled in the application domain ontology and the four data sources are organized into a hierarchy based on their geographical locations. The realization of patient information in the application domain ontology is shown below for demonstration purposes (e.g. using conjunctive query instead of the real xml data to illustrate the *Rule* component in the application domain ontology). Due to the limited space of the thesis, only some parts are illustrated:

Atomic Concepts: {…PatientID, Name, Gender, Age, Adult, Child, MaritalStatus, Guardian, MainDiagnosis,…}
Roles: {… HasPatientID, HasName, HasGender, HasAge, HasClassification, HasMartialStatus,

HasGuardian, …}

Composite Concepts: {…

Adult ⊑ Classification; Child ⊑ Classification; Adult ∩ Child ⊑ ⊥;

Patient := (=1HasPatientID.PatientID) ⊓(=1HasName.Name) ⊓ (=1HasGender.Gender) ⊓ (=1 HasAge.Age) ⊓ (=1HasClassification.Classification);

AdultPatient **:=** Patient ⊓ ∀HasClassification.*Adult* ⊓ (=1HasMartialStatus. MaritalStatus);

ChildPatient **:=** Patient ⊓ ∀HasClassification.*Child* ⊓ (=1 HasGuardian. Guardian);

…}

Rules: {… ;

Patient-Info(X_PatientID, X_Name, X_Gender, X_Age) **:-** Patient(X) ∧ HasPatientID(X, X_PatientID) ∧ HasName(X, X_Name) ∧ HasGender(X, X_Gender) ∧ HasAge(X, X_Age);

…}

Table 8-7 application domain ontology (partial)

The application domain ontology is designed by following the rule introduced in section 4.3.3.3: that it should be able to represent the general information precisely while allowing the data sources to easily describe their data.

The local schemas of the data sources are illustrated below:

Local schema of DB$_1$:

Patient(*PID*(String), *name*(String), *gender*(String), *age*(Int), *main-diagnosis*(String), *level*(String))

Patient-Assignment(*PID*(String), *Care_Service_ID*(String), *class*(String), *address*(String), *Staff_ID*(String), *Staff_name*(String), *Staff_category*(String))

Treatment (*PID*(String), *type*(String), *Spell*(String), *cost*(Double))

Local schema of DB$_2$:

Patient(*Patient_ID*(String), *first_name*(String), *middle_name*(String), *last_name*(String), *gender*(String), *age*(Int), *guardian*(String))

Care-Service(*Patient_ID*(String), *Care_Service_ID*(String), *class*(String), *address*(String))

Diagnosis(*Patient_ID*(String), *main-diagnosis*(String), *level*(String))

Treatment (*Patient_ID* (String), *type*(String), *Spell_ID*(String), *cost*(Double))

Spell(*Spell_ID*(String), *Jan*(String), *Feb*(String)…)

Local schema of DB$_3$:

Child-Patient(*PID*(String), *name*(String), *gender*(String), *age*(int), *guardian*(String))

Adult-Patient(*PID*(String), *name*(String), *gender*(String), *age*(Int), *Marital_status*(String))

Diagnosis(*PID*(String), *main-diagnosis*(Int), *level*(String))

Local schema of DB$_4$:

Patient_info(*PID*(String), *name*(String), *gender*(String), *age*(Int), *guardian*(String), *Marital_status*(String), *classification*(String))

Diagnosis(*PID*(String), *main-diagnosis*(String), *level*(String), *Staff_ID*(String), *Staff_name*(String), *Staff_category*(String))

Treatment (*PID*(String), *type*(String), *Spell_ID*(String), *cost*(Double))

Spell(*Spell_ID*(String), *month*(String), *Activity*(String))

Table 8-8 four typical local database schemas

In addition, DS_1 and DS_2 are located in *Durham* which further belongs to *County Durham*. DS_3 and DS_4 are located in *Newcastle*. The *organizational structure* is illustrated below (recall section 4.5.2). The data service corresponding to each data source is denoted by the name of the data source here for simplicity:

Classification: (*rootname* : Geographical Location, (*group*: County Durham), (*group*: Newcastle))

Group : (*gname* : County Durham, (*group*: Durham))

Group : (*gname* : Durham, (*Data Service*: DS_1), (*Data Service*: DS_2))

Group : (*gname* : Newcastle, (*Data Service*: DS_3), (*Data Service*: DS_4))

Table 8-9 organizational structure

It can be seen from the above local schemas; **DS_1** may be a clinic which holds the patients information with the diagnoses information in one relation and does not distinguish between an adult patient and a child patient. **DS_2** may be a social service specializing in child services which only holds child patient information. **DS_3** may be a research centre which distinguishes between the adult and child patient information and **DS_4** may be a hospital which puts the adult patient and child patient into one relation. The Treatment of a patient is described by the treatment activities the patient has had (i.e. the spell). In this case study, it is assumed there are 65 kinds of activities and a patient only has one activity in each month hence the spell of a patient is represented as a digital code (i.e. 1 to 65) for a month. Though it is much simpler than the practical situation, it is enough to demonstrate the main features of the SLEDIS. The above test data are used in the evaluation and the details will be discussed in later sections.

8.4 Capability of solving heterogeneity problems

As introduced in chapter 1, this research aims to provide a solution for integrating data from heterogeneous and frequently evolving participating databases to provide a unified vision for end users to utilize. Thus before solving the evolution problems, the heterogeneity problems must be solved first. For evaluating the capability of solving heterogeneity problems, the experimental system SLEDIS in the case study should be able to successfully integrate participating databases with various schematic heterogeneities as defined in section 1.2.3. As introduced in chapter 4, the

IPUD algorithm based on the IPU mechanism is designed for solving the heterogeneity problems, hence a research question was defined in section 8.2.2 as follows:

- How and why the IPU mechanism can integrate distributed databases with the heterogeneities defined in section 1.2.3.

For answering the research question, a proposition is also defined in section 8.2.2:

Proposition 1

> The heterogeneity problems defined in section 1.2.3 can be solved in the data integration through the IPU mechanism and the query processing

The Proposition 1 is investigated through integrating data from the four pre-designed participating databases (e.g. DS_1, DS_2, DS_3 and DS_4) in the SLEDIS. In principle, various heterogeneity problems have been covered by the four typical databases and each database constructs its IPUs without concerning other participating databases. If the heterogeneity problems in the four typical databases can be solved, then these heterogeneity problems can be solved by the SLEDI solution no matter how many participating databases are involved. The heterogeneities problem in a participating database is considered as “solved” if all of the following statements are **TRUE**:

- For each IPU constructed from the database, the head of the root view of its local definition is identical with the head of the content of its global definition
- For each local definition, a set of valid view definitions are constructed based on the local schema of the databases using conjunctive queries.
- The database can be found by the SLEDIS for answering user queries based on the global definition of the IPUs it constructed
- The results data for answering the user queries can be obtained as long as it exists in the database.

8.4.1 Various Heterogeneity Problems

As introduced in the previous section, there are four data sources where each data source holds one database. Each database models the patient information in its local database schema differently in order to cover the various heterogeneity problems. Each data source constructs a set of IPUs and publishes a data service to accommodate the IPUs. The metadata of each data service is stored in an xml file residing at the site of the corresponding data service. The local schema of each database was created individually and populated with a set of data tuples to further investigate whether the SLEDIS is able to produce correct results data for answering a set of pre-defined user queries. In order to examine the capability of solving the heterogeneity problems, it is helpful to discuss whether the four typical local database schemas have covered all types of the heterogeneities defined in section 1.2.3. The following table shows where each type of the heterogeneities is covered:

Heterogeneity type		Covered	Example
Attribute Level	Naming discrepancy	YES	The PatientID is modelled to an attribute *PID*(String) in $\mathbf{DB_1}$ but modelled to an attribute *Patient_ID*(String) in $\mathbf{DB_2}$.
	Attribute domain discrepancy	YES	1) The MainDiagnosis is modelled to an attribute *main-diagnosis*(String) in $\mathbf{DB_2}$ but modelled to an attribute *main-diagnosis*(int) in $\mathbf{DB_3}$. 2) The same instance of MainDiagnosis is represented as “schizophrenia” in the attribute *main-diagnosis*(String) in $\mathbf{DB_2}$ but represented as “schiz” in the attribute *main-diagnosis*(String) in $\mathbf{DB_4}$.
	Attribute granularity discrepancy	YES	The Name is modelled to a single attribute *name*(String) in $\mathbf{DB_1}$ but modelled to three attributes:*first_name*(String),*last_name*(String) and *middle_name*(String) in $\mathbf{DB_2}$.
Relation level	Naming discrepancy	YES	The ChildPatient is modelled in the relation **Child-Patient** in $\mathbf{DB_3}$ but modelled in the relation **Patient** in $\mathbf{DB_2}$
	Relation granularity discrepancy	YES	The Patient-Diagnosis is modelled in the relation **Patient** in $\mathbf{DB_1}$ but modelled in two relations **Patient_info** and **Diagnosis** in $\mathbf{DB_4}$.
Schema level	Abstraction level discrepancy	YES	The AdultPatient and ChildPatient are modelled as two separate relations **Adult-Patient** and **Child-Patient** respectively in $\mathbf{DB_3}$ but are modelled as a single relation **Patient_info** in $\mathbf{DB_4}$ with an attribute *classification*(String) to indicate whether an instance is an adult or a child patient.
	Schematic discrepancy	YES	1) The spell is modelled as two attributes *month*(String) and *Activity*(String) in $\mathbf{DB_4}$ while each treatment activity represented by a code string (i.e. 1 to 65) corresponds to the month. 2) The spell is modelled as 12 attributes *Jan*(String), *Feb*(String)… in $\mathbf{DB_2}$ while each attribute represents the corresponding activity code. 3) The spell is modelled as a single attribute *Spell*(String) in $\mathbf{DB_1}$ while representing the 12 activity codes in one string separated by comma.

	Missing item discrepancy	YES	1) The Guardian is modelled as an attribute *guardian*(String) in **DB$_4$** but it is not modelled as any attribute in **DB$_1$** 2) The Treatment is modelled as a relation **Treatment** in **DB$_1$** but it is not modelled as any relation in **DB$_3$**
	Relation isomorphism discrepancy	YES	The relation **Patient_info** in **DB$_4$** has an attribute *classification*(String) to indicate whether an instance is an adult or a child patient, but the relation **Patient** in **DB$_1$** has no indication at all.

Table 8-10 heterogeneity problems coverage

8.4.2 Constructed IPUs

As introduced in chapter 4, the process of data source describing constructs the application domain ontology to represent the unified vision of the IDS and integrates data from the participating databases by establishing the mappings between the local schema of the databases and the application domain ontology. Consequently, the heterogeneity problems among the databases are solved by the mappings. The mappings are established through using the IPUD algorithm to construct a set of IPUs, more precisely, the global definitions and local definitions of the IPUs. As a matter of fact, some of the heterogeneity problems are solved by the global definitions and local definitions, while others are solved by the query processing.

As the application domain ontology and the local schemas of the participating databases have been created in the previous section, whether a set of IPUs can be established can be discussed. Moreover, after the various heterogeneity problems and their examples have been listed in table 8-10, whether and how the heterogeneity problems can be solved can also be discussed based on the constructed IPUs. The global definitions and local definitions of the IPUs constructed for the participating databases **DB$_1$**, **DB$_2$**, **DB$_3$** and **DB$_4$** are listed in the following table:

IPUs constructed from DB$_1$: **Global Definitions:**

Content: **V$_{11}$**(X):- Patient(X), **V$_{12}$**(X,Y):- HasPatientID(X,Y), **V$_{13}$** (X,Y):-HasName(X,Y), ……

Constraint: $\mathbf{V_{12}}(X,Y) \wedge \neg \mathbf{V_{11}}(X) \sqsubseteq \perp$, ……

Local Definitions: Root Views:

V$_{11}$(X):-CQ1(Subgoal1: (RRSassignment (**BV$_{11}$**; X→PatientID)

V$_{12}$(X,Y):-CQ1(Subgoal1: (RRSassignment (**BV$_{11}$**; X→PatientID, Y→PatientID)

V$_{13}$(X,Y):-CQ1(Subgoal1: (RRSassignment (**BV$_{11}$**; X→PatientID, Y→Name)

……

Intermediate Views:

Basic Views:

BV$_{11}$(PatientID, Name, Gender, Age, MainDiagnosis, Level):-CQ1(Subgoal1: (RRSassignment (**Patient**; PatientID→*PID*, Name→*name*, Gender→*gender*, Age→*age*, MainDiagnosis→*main-diagnosis*, Level→*level*)))

BV$_{12}$(PatientID, ServiceID, ServiceClass, ServiceAddress, StaffID, StaffName, StaffCategory):-CQ1(Subgoal1: (RRSassignment (**Patient-Assignment**; PatientID→*PID*, ServiceID→*Care_Service_ID*, ServiceClass→*class*, ServiceAddress→*address*, StaffID→*Staff_ID*, StaffName→*Staff_name*, StaffCategory→*Staff_category*)))

BV$_{13}$(PatientID, TreatmentType, TreatmentSpell, TreatmentCost):-CQ1(Subgoal1: (RRSassignment (**Treatment**, PatientID→*PID*, TreatmentType→*type*, TreatmentSpell→*Spell*, TreatmentCost→*cost*)))

IPUs constructed from DB$_2$: **Global Definitions:**

Content: **V$_{21}$**(X):- ChildPatient(X), **V$_{22}$**(X,Y):- HasPatientID(X,Y), **V$_{23}$** (X,Y):-HasName(X,Y), ……

Constraint: **V$_{22}$**(X,Y)∧¬ **V$_{21}$**(X) ⊑⊥, ……

Local Definitions: Root Views:

V$_{21}$(X):-CQ1(Subgoal1: (RRSassignment (**Iv$_{21}$**; X→PatientID)

V$_{22}$(X,Y):-CQ1(Subgoal1: (RRSassignment (**Iv$_{21}$**; X→PatientID, Y→PatientID)

V$_{23}$(X,Y):-CQ1(Subgoal1: (RRSassignment (**Iv$_{21}$**; X→PatientID, Y→Name)

……

Intermediate Views:

Iv$_{21}$(PatientID, Name, Gender, Age, Guardian):-CQ1(Subgoal1: **BV$_{21}$**, Subgoal2: (CRSassignment (Name): (fname+mname+ lname)))

Iv$_{22}$(PatientID, TreatmentType, TreatmentSpell, SpellCost):-CQ1(Subgoal1: **BV$_{24}$**, Subgoal2: **BV$_{25}$,** Subgoal3: (CRSassignment (TreatmentSpell): (Jan+Feb+…)))

Basic Views:

BV$_{21}$(PatientID, fname, mname, lname, Gender, Age, Guardian):-CQ1(Subgoal1: (RRSassignment (**Patient**; PatientID→*Patient_ID*, fname→*first_name*, mname→*middle_name*, lname→*last_name* Gender→gender, Age→age, Guardian→*guardian*)))

BV$_{22}$(PatientID, ServiceID, ServiceClass, ServiceAddress):-CQ1(Subgoal1: (RRSassignment (**Care-Service**; PatientID→*Patient_ID*, ServiceID→*Care_Service_ID*, ServiceClass→*class*, ServiceAddress→*addres*)))

BV$_{23}$(PatientID, MainDiagnosis, Level):-CQ1(Subgoal1: (RRSassignment (**Diagnosis**, PatientID→*Patient_ID*, MainDiagnosis→*main-diagnosis*, Level→*level*)))

BV$_{24}$(PatientID, TreatmentType, spellID, TreatmentCost):-CQ1(Subgoal1: (RRSassignment (**Treatment**, PatientID→*Patient_ID*, TreatmentType→*type*, spellID→*Spell_ID*, TreatmentCost→*cost*)))

BV$_{25}$(spellID, Jan, Feb…):-CQ1(Subgoal1: (RRSassignment (**Spell**, spellID→*Spell_ID*, Jan→*Jan*, Feb→*Feb*, …)))

IPUs constructed from DB$_3$: **Global Definitions:**

Content: $\mathbf{V_{31}}$(X):- ChildPatient(X), $\mathbf{V_{32}}$(X):- AdultPatient(X), $\mathbf{V_{33}}$(X,Y):- HasPatientID(X,Y), $\mathbf{V_{34}}$ (X,Y):-HasName(X,Y), ……

Constraint: $\mathbf{V_{33}}(X,Y) \wedge \neg \mathbf{V_{31}}(X) \sqsubseteq \perp$, $\mathbf{V_{34}}(X,Y) \wedge \neg \mathbf{V_{32}}(X) \sqsubseteq \perp$,……

Local Definitions: Root Views:

$\mathbf{V_{31}}$(X):-CQ1(Subgoal1: (RRSassignment (**BV**$_{31}$; X→PatientID)

$\mathbf{V_{32}}$(X):-CQ1(Subgoal1: (RRSassignment (**BV**$_{32}$; X→PatientID)

$\mathbf{V_{33}}$(X,Y):-CQ1(Subgoal1: (RRSassignment (**BV**$_{31}$; X→PatientID, Y→PatientID)

$\mathbf{V_{34}}$(X,Y):-CQ1(Subgoal1: (RRSassignment (**BV**$_{32}$; X→PatientID, Y→Name)

……

Intermediate Views:

IV$_{31}$(PatientID, MainDiagnosis, Level):-CQ1(Subgoal1: **BV**$_{33}$, Subgoal2: (CRSassignment (MainDiagnosis): (&maindiagnosis)))

Basic Views:

BV$_{31}$(PatientID, Name, Gender, Age, Guardian):-CQ1(Subgoal1: (RRSassignment (**Child-Patient**; PatientID→PID, Name→name, Gender→gender, Age→age, Guardian→*guardian*)))

BV$_{32}$(PatientID, Name, Gender, Age, MaritalStatus):-CQ1(Subgoal1: (RRSassignment (**Adult-Patient**; PatientID→PID, Name→name, Gender→gender, Age→age, MaritalStatus→*Marital_status*)))

BV$_{33}$(PatientID, maindiagnosis, Level):-CQ1(Subgoal1: (RRSassignment (**Diagnosis**, PatientID→*PID*, maindiagnosis→*main-diagnosis*, Level→*level*)))

IPUs constructed from DB$_4$: **Global Definitions:**

Content: $\mathbf{V_{41}}$(X):- ChildPatient(X), $\mathbf{V_{42}}$(X):- AdultPatient(X), $\mathbf{V_{43}}$(X,Y):- HasPatientID(X,Y), $\mathbf{V_{44}}$ (X,Y):-HasName(X,Y), ……

Constraint: $\mathbf{V_{43}}(X,Y) \wedge \neg \mathbf{V_{41}}(X) \sqsubseteq \perp$, $\mathbf{V_{44}}(X,Y) \wedge \neg \mathbf{V_{42}}(X) \sqsubseteq \perp$,……

Local Definitions: Root Views:

$\mathbf{V_{41}}$(X):-CQ1(Subgoal1: (RRSassignment (**IV**$_{411}$; X→PatientID)

$\mathbf{V_{42}}$(X):-CQ1(Subgoal1: (RRSassignment (**IV**$_{42}$; X→PatientID)

$\mathbf{V_{43}}$(X,Y):-CQ1(Subgoal1: (RRSassignment (**IV**$_{41}$; X→PatientID, Y→PatientID)

$\mathbf{V_{44}}$(X,Y):-CQ1(Subgoal1: (RRSassignment (**IV**$_{42}$; X→PatientID, Y→Name)

……

Intermediate Views:

IV$_{41}$(PatientID, Name, Gender, Age, Guardian):- CQ1(Subgoal1: **BV**$_{41}$, Subgoal2: (ORSassignment (Classification=”Child”)))

IV$_{42}$(PatientID, Name, Gender, Age, MaritalStatus):- CQ1(Subgoal1: **BV**$_{41}$, Subgoal2: (ORSassignment (Classification=”Adult”)))

IV$_{43}$(PatientID, MainDiagnosis, Level):-CQ1(Subgoal1: **BV$_{42}$**, Subgoal2: (CRSassignment (MainDiagnosis): (&maindiagnosis)))

Iv$_{44}$(PatientID, TreatmentType, TreatmentSpell, TreatmentCost):-CQ1(Subgoal1: **BV$_{43}$**, Subgoal2: **BV$_{44}$,** Subgoal3: (TreatmentSpell): Program(month, activity))

Basic Views:

BV$_{41}$(PatientID, Name, Gender, Age, Guardian, MaritalStatus, Classification):-CQ1(Subgoal1: (RRSassignment (**Patient_info**; PatientID→PID, Name→name, Gender→gender, Age→age, Guardian→*guardian*, MaritalStatus→*Marital_status*, Classification→*classification*)))

BV$_{42}$(PatientID, maindiagnosis, Level, StaffID, StaffName, StaffCategory):-CQ1(Subgoal1: (RRSassignment (**Diagnosis**, PatientID→*PID*, maindiagnosis→*main-diagnosis*, Level→*level*, StaffID→*Staff_ID*, StaffName→*Staff_name*, StaffCategory→*Staff_category l*)))

BV$_{43}$(PatientID, TreatmentType, spellID, TreatmentCost):-CQ1(Subgoal1: (RRSassignment (**Treatment**, PatientID→*PID*, TreatmentType→*type*, spellID→*Spell_ID*, TreatmentCost→*cost*)))

BV$_{44}$(spellID, month, activity):-CQ1(Subgoal1: (RRSassignment (**Spell**, spellID→*Spell_ID*, month→*month*, activity→*Activity*)))

Table 8-11 Global and local definitions of the IPUs

Due to the limited space of the thesis, only some of the global and local definitions are shown in the above table which can be used as examples to demonstrate how the heterogeneity problems are tackled. It is apparent that the view definitions in the above table (e.g. the definition of the basic, intermediate and root views) only show one possible construction of the global and local definitions. Different programmers may construct the view definitions differently for their own preference. In theory, the IPUD algorithm can be regarded as effective if at least one construction can solve the heterogeneity problems successfully.

It can be seen from table 8-11, the naming discrepancy in the attribute level and the relation level were solved through the *assignment* in the Regular Relation Subgoal (RRS) (see section 4.4.4.2) in the view definitions. For example, the attributes *PID*(String) in **DB$_1$** and *Patient_ID*(String) in **DB$_2$** are assigned to the same variable PatientID through the basic views **BV$_{11}$** and **BV$_{21}$** respectively. The relations **Patient** in **DB$_2$** and **Child-Patient** in **DB$_3$** are firstly assigned to **BV$_{21}$** and **BV$_{31}$** and then assigned to the root views **V$_{21}$** and **V$_{31}$** respectively, and then through the global definitions of the **V$_{21}$** and **V$_{31}$**, both of them are assigned to the ChildPatient relation in the application domain ontology. Hence the naming discrepancies were solved through assigning the heterogeneous attributes and relations to the corresponding elements in the application domain ontology.

The attribute domain discrepancy and the attribute granularity discrepancy in the attribute level were solved through the *assignment* in the Converting Relation Subgoal (RRS) in the view definitions. For example, the attribute *main-diagnosis*(int) in **DB_3** is converted into the string data type and assigned to the variable MainDiagnosis through the intermediate view **IV_{31}**. The three attributes: *first_name*(String), *middle_name*(String) and *last_name*(String) in **DB_2** are firstly combined into one attribute and then assigned to the variable Name through the intermediate view **IV_{21}**. The schematic discrepancy in the schema level was solved in a similar fashion. For example, through constructing **BV_{24}**, **BV_{25}** and **IV_{22}** for **DB_2** and constructing **BV_{43}**, **BV_{44}** and **IV_{44}** for **DB_4**, the data of patient treatment in the **DB_2** and **DB_4** were firstly converted in the same format as in the **DB_1** and then assigned to the corresponding local and global definitions.

The relation granularity discrepancy in the relation level was solved through constructing the local definitions and the corresponding global definitions. Since the *actual relations* in a local schema are mapped to a set of *virtual relations* which are the head relations of the local definitions and the content of the global definitions (see section 4.4.3), the global definitions represent the actual relations using the elements from the same application domain ontology. Consequently, the relation granularity discrepancy was solved. For example, the **Patient** and **Diagnosis** information is realized as one relation in **DB_1** but as two relations in **DB_4**, Both **DB_1** and **DB_4** using a set of virtual relations (i.e. **V_{11}**(X), **V_{12}**(X,Y)… and **V_{41}**(X), **V_{42}**(X,Y)…) to represent the information.

The abstraction level discrepancy in the schema level was solved through the *assignment* in the Ordered Relation Subgoal (ORS) in the view definitions and the construction of the local and global definitions. For example, by constructing two intermediate views, **IV_{41}** and **IV_{42}**, and the corresponding local and global definitions, **DB_4** extracts the adult and child patient information separately from its actual relation **Patient_info**. The missing item discrepancy and the relation isomorphism discrepancy in the schema level were not addressed explicitly by the construction of the local and global definitions. In fact, the two discrepancies are similar as both of them result from a lack of necessary data in the participating databases. However, they were considered in the query processing which will be discussed in the next section.

In principle, most of the heterogeneity problems were solved through construction of the local and global definitions of the IPUs. The correctness of the local and global definitions was tested before conducting further evaluations onto the system. As introduced in chapter 7, the views in the local definitions are materialized through constructing SQL queries based on the view definitions. This enables the investigator to conduct tests onto each local definition individually. All the views were tested through a set of queries which involved all the attributes in the views head and no errors were found. Thus the construction of the SQL queries from the view definitions was considered correct.

8.4.3 User queries and results

After the correctness of the local definitions was checked, the capability of solving heterogeneity problems was further evaluated through running a set of user queries against the SLEDIS to obtain the results data. The *correct answers* of the queries were manually built as *testing oracle* hence the system produced results data can be verified against it. In order to evaluate the capability thoroughly, the correct answer for each user query includes the *result tuples* and the *relevant databases*. The result tuples refer to all the results data which can be obtained from the participating databases for answering the user query which are in the form of relation tuples. The relevant databases refer to the participating databases which can provide result tuples for answering the user query even if they only provide data on some attributes of the tuples. The result of the evaluation is shown in the following table:

User Query	Number of Result Tuples		Number of Relevant Databases	
	Manually Produced	System Produced	Manually Produced	System Produced
Q1	4	3	6	4
Q2	3	3	4	4
Q3	3	3	3	3
Q4	2	2	3	3
Q5	3	2	4	2
Q6	2	2	4	4
Q7	4	4	4	4
Q8	4	4	5	5
Q9	3	2	4	3
Q10	2	2	2	2

Table 8-12 Result for evaluating heterogeneity problems solving capability

It can be seen from table 8-12, that the system produced results data tuples of 3 out of 10 user queries were less than the correct answers. Hence further investigations on the missing tuples and relevant databases were conducted through comparing the tuples and relevant databases produced by the system and the correct answers. It is observed that the missing tuples of the system producing results data are the tuples provided by the missing relevant databases. The missing relevant databases are the participating databases which do not provide some attributes required by the user queries. However, this is not surprising due to the PICSEL query rewriting algorithm adopted in this research as introduced in chapter 5. If a participating database fails to provide one or more attributes the user query requires, it may not be considered as a relevant database during the query processing. For example, a user query requires all information of child patient which has a conjunct *Guardian*(X), and a participating database provides child information except the *guardian* attribute, thus there is no HasGuardian(X,Y) or Guardian(X) in its global definitions. Consequently, the conjunct cannot be grounded to the global definitions of the database and the database is not considered as relevant, it results in other information such as name, age and etc are not obtained even though the database does provide those information. Therefore, the manually built correct answers were then modified by following the PICSEL query rewriting algorithm strictly and the system produced results data which became identical with the correct answers.

In fact, the above investigation demonstrates how the missing item discrepancy was addressed during the query processing. The relation isomorphism discrepancy was also addressed in the same way. For example, $\mathbf{DB_1}$ provides the patient information but fails to distinguish between the adult and child patient, hence it does not have global definitions representing adult or child patient specifically and it will not be considered as a relevant database if a user query requires precisely adult or child patient information. Through not considering the participating databases as relevant, the missing item discrepancy and relation isomorphism discrepancy were solved and the system can integrate data from the participating databases without errors.

8.4.4 Summary

The results of the test have shown that most types of heterogeneity problems defined in section 1.2.3 were successfully solved by the IPU mechanism, more precisely, by using the IPUD algorithm to construct the local and global definitions of the IPUs. Some of the heterogeneity problems were solved by the *assignment* used in the view definitions and some others were solved by constructing local and global definitions. It is worth mentioning that there is more than one possible ways to construct the local and global definitions; programmers may construct the view definitions based on their own preference with no strict rules on which set of views should be used for solving which specific type of heterogeneity problem.

The missing item discrepancy and relation isomorphism discrepancy were solved by not considering the participating databases as relevant during the query processing. This results in the system may not being able to produce the complete results data in some circumstances. However, this is due to the adoption of the PICSEL query rewriting algorithm hence is not an issue raised from constructing local and global definitions. A possible solution is the participating database may be to add extra attributes with null values and construct local and global definitions to represent the database providing the attributes, although only null values can be obtained. Consequently, the databases will be considered as relevant and the results tuples can be obtained from the database with null value in the extra attributes.

The local definitions and global definitions of the IPUs for the participating databases were successfully constructed, and the results data of the testing user queries produced by the system were verified to be correct. Thus, proposition 1 is well supported, the IPU mechanism is considered to be effective and the SLEDI has the capability of solving the heterogeneity problems defined in this thesis.

8.5 Capability of Solving Evolution Problems

After the capability of solving heterogeneity problems had been evaluated, further evaluations were conducted. As introduced in chapter 1, the main focus of this research is aimed at using the SLEDI solution to solve some evolution problems of the participating databases in the data integration. Thus a set of research questions, propositions and measures were defined in section 8.2.2 intending to thoroughly evaluate the capability of SLEDI on solving those evolution problems which have been defined in chapter 1. This section discusses whether each of the propositions was supported with reference to the results of the measures. Although the conceptual services IPS and EHS of a participating database were combined into a single DS as introduced in section 7.4.1, they were implemented as two different operations. Thus the propositions based on the conceptual services as listed above can still be evaluated through examining the corresponding operations.

8.5.1 Proposition 2

The IPU mechanism and the metadata of the IPS and BS can reduce the modification effort caused by the schematic evolution occurring in the participating databases

The measures for verifying the proposition are:

A. Number of user queries explicitly specifies the participating databases.
B. Numbers of user queries explicitly specifies the local database schemas.
C. Number of hard coded queries in mappings.
D. Number of the affected user queries.
E. Number of IPUs which require modification effort.
F. Human intervention.

In order to avoid bias in the testing, the user queries used for testing were designed by a colleague of the investigator (i.e. not the same programmer for developing and evaluating the system)

Number of total user queries in the testing: **10**
Number of user queries explicitly specifies the participating databases: **0**

Table 8-13 Results of Measure A

It can be seen from the above table that there is *no* user query to explicitly specify which participating databases the query intends to query against. We recall that all the participating databases are organized into the *organizational structure* through assigning each database into a virtual group and organizing the groups into classifications in the DAG structure. A user query can only specify its targeting databases through designating the names of the classification and groups in the *Q(org)* component of the query, even though the user knows exactly the names of the targeting databases. This allows the end users to specify the targeting databases through the virtual groups, instead of through the hard coded names of the targeting databases.

The user query will not be affected and no modification work is needed when the schematic evolution or system level evolution (e.g. changing the name or removal of the database) occurring in the targeting databases. Compared to the Federated database management system (FDBMS) which also follows the virtual data integration approach introduced in chapter 2, the SLEDI provides a more flexible way for end users to raise queries. The user queries in a FDBMS need to designate all the databases they intend to query against explicitly, thus when evolution occurs in the databases, user queries may have to be modified accordingly to indicate the evolved databases. From this comparison, SLEDI reduces the modification work onto the user queries caused by the database evolution hence proposition 2 is supported by the result of the measure A.

Number of total user queries in the testing: **10**
Numbers of user queries explicitly specifies the local database schemas: **0**

Table 8-14 Results of measure B

It can be seen that there is *no* user query to explicitly specify the local schemas of the databases it intends to query against. We recall that the SLEDI integrates the participating databases through establishing the mappings between the local schemas of the participating databases and the application domain ontology. The mappings are realized through the IPU mechanism and the metadata of the IPSs and BS. End users only see the virtual integrated database and raise queries as conjunctive queries over the relations in the application domain ontology in the *Q(onto)* component of the queries. Hence end users do not need to concern the local schemas of the participating databases.

This design allows end users to focus on the virtual integrated database instead of a set of participating databases and to query against the application domain ontology instead of against a set of local schemas. It is apparent that no modification work onto the existing user queries is required when the local schemas of the participating databases evolve as the application domain ontology remains stable. Compared to the loosely coupled FDBMS in which user queries need to explicitly include the local schemas of the participating databases, SLEDI reduces the modification work onto the user queries as the schematic evolution occurring in the participating database in the loosely couple FDBMS may result in having to modify the existing user queries. Consequently, the proposition 2 is supported by the result of the measure B.

Number of participating databases: **4**
Numbers of hard coded queries in mappings (lines of program): **0**

Table 8-15 Results of measure C

As introduced previously, one of the causes of high maintenance costs of the data integration system is that the data integration is achieved through hard coded queries. In other words, the mappings between local schemas of the participating databases and the application domain ontology are realized as hard coded programs. It can be seen from the above table that there is 0 line of program for realizing the mappings. This is because all the mappings (i.e. the local and global definitions) are constructed as DAG structured data and stored as the metadata of the IPSs and BS, hence no mappings are coded in the programs. The programs only focus on implementing the algorithms to access and manage the metadata (i.e. mappings) hence the evolution occurring in the participating databases may only result in modification to the metadata and no modification work is required to the programs.

The evolution occurring in the participating databases in a FDBMS may result in a large amount of modifications since the mappings are realized as hard coded programs. Modifying hard coded programs can be very complex and requires huge amounts of work, especially when the volume of the mappings is huge. Compared to the FDBMS, the SLEDI largely reduces the modification work to the mappings when the local schemas of the participating databases evolve hence proposition 2 is supported by the result of the measure C.

As introduced in chapter 6, the evolution problems the SLEDI solution is designed to tackle include three classes: *system level*, *schematic* and *organizational* evolution. Each class covers various types of evolution occurring in the participating databases. In order to examine how the evolutions were tackled by the SLEDIS, a set of evolutions were applied to the participating databases in the SLEDIS to obtain the results of the measure D, E and F. Different types of evolution may affect different components of the SLEDIS. For example, the schematic evolution may affect the local and global definitions but the organizational evolution will not. Even the same type of evolution may still impact SLEDIS differently depends on where the evolution occur. For example, removal of an attribute may only affect one basic view when it occurs on one attribute, A, but affect a set of views if it occurs on another attribute, B. In order to conducting the evaluation thoroughly, the evolution applied onto the SLEDIS not only covers all types of the evolution, but also triggers all the possible evolution handling processes. The results are listed in the following tables with respect to the different class of evolution:

Evolution type	Evolution Applied	User Query Affected	Local definition modified	Global definition modified	Human Intervention
Database Name Change	2	0	0	0	Yes
Service Name Change	2	0	0	0	Yes
Service URL Change	2	0	0	0	Yes

Table 8-16 Results of system level evolution

It can be seen that all the types of system level evolution have no impact on either user queries or on local and global definitions. We recall that the SLDEIS was developed based on the Web Service technology and the descriptive information of the participating databases were stored in the xml format registry separately with other information such as organizational structure and global definitions. Consequently, the system evolution will not affect the user queries and the local and global definitions. Although the evolution handling requires human intervention, it was simply done through manually modifying the corresponding data in the xml files representing the database name, service name and URL, where each modification was completed in no more than one minute. The advantages of employing the web service technology will be discussed in later in this section.

Evolution type	Evolution Applied	User Query Affected	Local definition modified	Global definition modified	Human Intervention
Attribute Addition	8	0	3	3	Yes
Attribute Removal	8	0	10	10	No
Attribute Rename	8	0	12	0	No
Attribute Domain Change	3	0	4	0	No
Attribute Decomposition	2	0	3	0	No
Attribute Aggregation	2	0	3	0	No
Relation Addition	2	0	8	8	Yes
Relation Removal	2	0	12	12	No
Relation Rename	2	0	12	0	No
Relation Decomposition	1	0	5	0	No
Relation Aggregation	1	0	5	0	No
Database Addition	1	0	9	9	Yes
Database Removal	1	0	8	8	No

Table 8-17 Results of schematic evolution

It can be seen from table 8-17 that the schematic evolution may only affect the local and global definitions and have no impact on the user queries. Some types of schematic evolution result in the modifications to both the local definitions and global definitions, while some other types only cause local definitions to be modified. This can be explained by recalling the data source describing process introduced in chapter 4; the local definitions are constructed as views over the local schemas and their heads are identical to the corresponding global definitions. Some schematic evolution may cause the views to become invalid hence the local definitions need to be modified in order to make the views valid again, thus the corresponding global definitions remain stable. On the other hand, some schematic evolution may result in the local definitions not being valid through modification, thus the local definitions and the corresponding global definitions have to be removed. Consequently, the number of global definitions modified is always less than or equal to the number of local definitions modified.

It may be realized from table 8-17 that the number of local definitions modified may be larger than the number of evolutions applied. This is not surprising since the local definitions are constructed in the DAG structure and one attribute or relation can be the successor of multiple views and one schematic evolution may lead to multiple local definitions requiring modification. There were only 3 out of 13 types of schematic evolution requiring human intervention which means the majority (10 out of 13 types) of the schematic evolution were handled automatically. The types of schematic evolution requiring human interventions are the addition of the attribute, relation or database. Although the newly-added elements may only require the construction of new IPUs (i.e. the local and global definitions) and have no impact on the existing IPUs, they need to be integrated into the system and hence will involve some manual effort.

Compared to the FDBMS where mappings between the local schemas and the application domain ontology are represented by hard coded queries; the SLEDI solution largely reduces the maintenance effort by representing the mappings through the IPU mechanism and structured metadata hence automatic evolution handling can be applied. Although there is still a small amount of schematic evolutions which involve human intervention, most types of schematic evolution were handled automatically hence the proposition 2 is supported by the result of the measures D, E and F.

8.5.2 Proposition 3

The organizational evolution handling in the query processing can reduce the modification effort made to the user queries caused by the organizational evolution

Since proposition 3 is focusing specifically on how SLEDIS handles the impact on the existing user queries brought about by the organizational evolution, the result of the measures D and F can also be used for verifying the proposition. The results of the measure on applying the organizational evolution onto the SLEDIS are listed in the following table:

Evolution type	Evolution	User Query	User Query	Human

	Applied	Affected	Modified	Intervention
Group Addition	2	0	0	No
Group Removal	2	8	0	No
Group Rename	2	8	0	No
Group Separation	1	4	0	No
Group Aggregation	1	8	0	No
Classification Addition	2	0	0	No
Classification Removal	2	10	0	No
Classification Rename	2	10	0	No
Classification Separation	1	10	0	No
Classification Aggregation	1	10	0	No

Table 8-18 Results of organizational evolution

It can be seen from table 8-18 that organizational evolution impacted on the existing user queries under most circumstances (8 out of 10 types organizational evolution). This can be explained by recalling the user query format introduced in chapter 5 where every user query includes a *Q(onto)* component to specify the targeting databases it intends to query against. The targeting databases were specified through designating the classifications and virtual groups the targeting databases belong to with respect to the organizational structure. Since the organizational evolution may introduce alterations to the classifications and/or the virtual groups specified in the *Q(onto)* component of the existing user queries, the user queries may be affected and hence require evolution handling.

Although the existing user queries were affected, it is apparent from table 8-18 that no modifications to the user queries were needed and no human intervention was required in the evolution handling. This can be explained by recalling the organizational evolution handling process introduced in chapter 6. The process automatically modifies the *Q(onto)* component of the existing user queries during the query processing based on the *organizational evolution* information stored in the registry, hence the *Q(onto)* becomes consistent again with the latest version of the organizational structure. Consequently, the targeting databases can still be determined correctly without modifying the existing user queries.

It may be realized from the table that the number of existing user queries which were affected may be larger than the number of organizational evolutions applied. This is because the same classifications and/or virtual groups may appear in multiple existing user queries. The addition of the classifications and/or virtual groups will not affect the existing user queries as they did not exist when the user queries were composed hence it is not possible for them to appear in the queries. The removal of the classifications and/or virtual groups may cause no targeting databases to be selected since the classifications and groups may no longer exist. Although it is not shown in the table, the results data obtained for the existing user queries were verified as expected with respect to the latest version of the organizational structure at the time when the queries were answered. Consequently, proposition 3 is supported by the results of the measures D and F.

8.5.3 Proposition 4

The schematic evolution occurring in one participating database will not affect the metadata of the IPSs provided by other data sources hence the system can still work properly.

Proposition 4 is focusing on whether schematic evolution occurring in one participating database will lead to modifications to the metadata of the IPSs (i.e. data services) provided by other participating databases. A set of schematic evolution were applied onto the participating database individually one at a time. Observing the metadata of which data services were modified by the SLEDIS, the result is shown in the following table:

Metadata modified to / Evolution Applied to	DS_1	DS_2	DS_3	DS_4
DB_1	Yes	No	No	No
DB_2	No	Yes	No	No
DB_3	No	No	Yes	No
DB_4	No	No	No	Yes

Table 8-19 Evolution applied and metadata modified among databases

It can be seen from table 8-19 that schematic evolution occurring in one database only leads to modifications to the metadata (i.e. the local and global definitions) of the data service provided by this database. The local and global definitions of other participating databases were not affected at all. Although not shown in the above table, the results data of the user queries were verified as expected under all the evolution applied. For example, when a database is removed from the SLEDIS, the data service hence all the local and global definitions the database provides, are removed. Under these circumstances, the results data obtained for the user queries are the results data which can be obtained from the other three participating databases left in the SLEIDS.

This can be explained by recalling the data source describing process introduced in chapter 4; every participating database describes the data it provides without being concerned with whether there are other participating databases involved in the system, what data other databases provide, or what the relationships are between the data provided by itself and the data provided by other participating databases. This design leads to the advantage that every participating database only maps its local schema to the application domain ontology. The relationships among the participating databases were handled at the query processing stage, not at the data source describing stage. Hence there is no information about the relationship of the participating database embedded in the mappings (i.e. local and global definitions). Schematic evolution occurring in one database can only affect the mappings of this database; other participating databases will be working as usual. Consequently, proposition 4 is well supported.

8.5.4 Proposition 5

The EHS can reduce the cost of the modification effort caused by the schematic evolution

As introduced in chapter 6, the evolution handling process includes organizational evolution handling and schematic evolution handling. The organizational evolution handling was implemented as part of the query processing while the schematic evolution handling was implemented as EHS. The EHS handles the schematic evolution by modifying the metadata of the corresponding IPS and the BS (i.e. local definitions and global definitions) through implementing the 10 pre-defined processes with respect to the schematic evolution type, as introduced in section 6.4.2.2. This design aims at using programs to automatically modify the structured metadata instead of human effort to revise the mappings for handling the various types of schematic evolution. Since proposition 5 is focusing on whether the EHS can reduce the modification effort of handling the schematic evolution, the investigation was conducted through examining the time costs for handling the various types of schematic evolution in the SLEDIS.

It can be seen from table 8-17 that most types of schematic evolution were handled automatically by EHS while the addition of the attribute, relation and database requires human intervention. In order to make the investigation thorough, the time costs of the EHS for handling each individual type of schematic evolution were examined. As a set of evolutions were applied for each type as listed in table 8-17, the average time for handling one schematic evolution of each type applied in one participating database were calculated. Since the cost of schematic evolution handling which requires human intervention may vary depending on what are the newly-added elements, only the costs of automatic evolution handling were counted. The results are shown in the following chart:

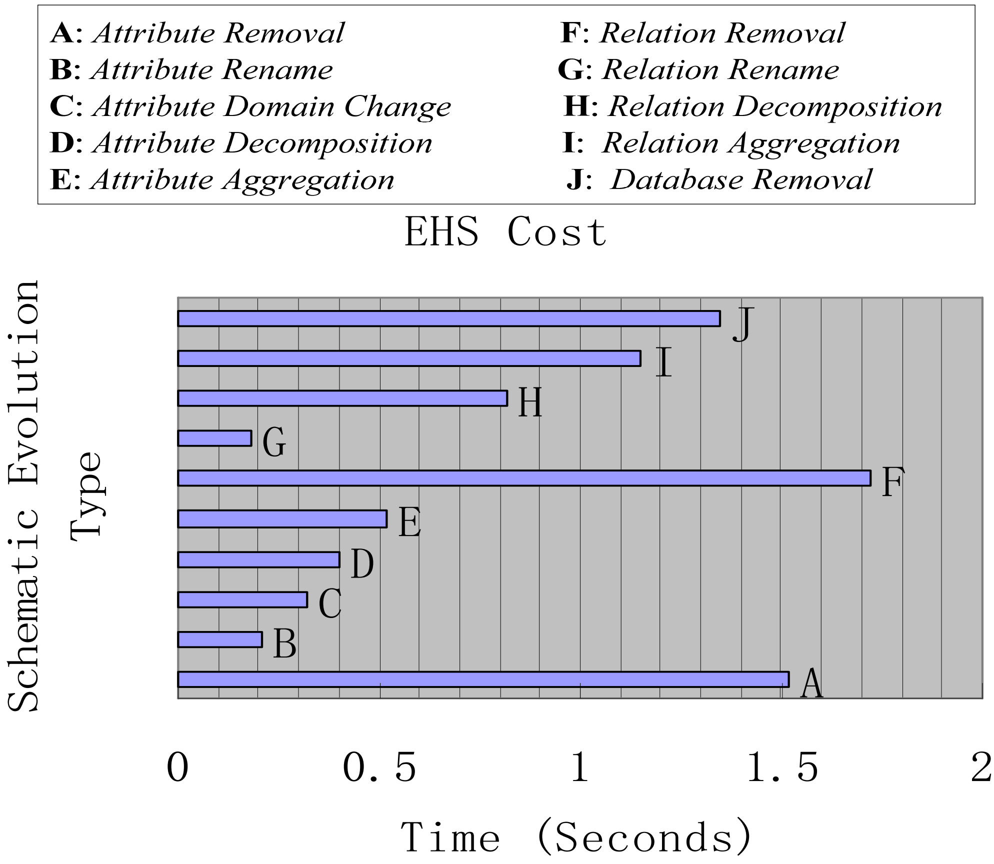


Figure 8-1 Time costs of EHS

It can be seen from figure 8-1 that the processes for handling the attribute rename and relation rename cost the *least* computational time. This is because the attributes and relations were all assigned to basic views, as introduced in chapter 4, thus when the name of attributes and/or relations changed, EHS only needed to modify the *assignments* in the basic views hence other parts of the local definitions remained unchanged. The processes for handling attribute domain change, attribute decomposition and attribute aggregation took slightly more time as the subgoals of the basic views also needed to be modified. Then, the processes tackling relation decomposition and relation aggregation took even more time as new basic views were added and other views in the local definition also needed to be modified. Finally, the processes for handling attribute removal, relation removal and database removal cost *most* computational time as the global definitions also had to be modified. This required communications between the IPS and BS hence cost more computational time. In summary, the average time for handling one schematic evolution was always less than 2 seconds, irrespective of the type of the schematic evolution.

In an FDBMS where the mappings between local schemas and application domain ontology are realized through hard coded queries, the handling of schematic evolution may require an understanding of and revising of the hard coded queries. Compared to the FDBMS, the SLEDI solution largely reduces the maintenance costs of handling the schematic evolution as most types of the schematic evolution can be handled through automatically modifying the structured metadata by the EHS. Since the EHS only takes less than 2 seconds for handling one schematic evolution, which no human effort can achieve, the proposition 5 is well supported.

8.5.5 Proposition 6

The service late binding based on the SOA and web service can help to reduce the maintenance costs caused by the evolution defined in the thesis.

As introduced previously, the SLEDI is a service based solution following the DaaS approach and is designed for reducing the maintenance costs of the data integration system through applying late binding to the data services. Since proposition 6 is focusing on whether the SOA and web service technology can help to reduce the maintenance costs, the investigation was conducted through examining whether the characteristics that SOA and web service intrinsically support, such as the service autonomy, discoverability, abstraction and loose coupling can facilitate SLEDI to achieve its design purpose.

As the participating data sources are in autonomous control, the owners of the data sources require full control over their databases. This is supported in SLEDI through the data source describing process as the owner of the participating data source plays the role of data service provider. They only expose the data they are willing to share through the global definitions. In other words, the external service consumers (i.e. the BS) can only access the data described by the global definitions instead of accessing the local schema directly hence data source owners have full control over what data can be accessed. In addition, the permission of accessing the local databases (e.g. the user name and password) was granted only to the data service, not to the external service consumers and the data services were situated at the sites of the corresponding data sources. This means the data source owners retain full control of their local databases and there is no extra constraint added onto the local databases hence the owners may still run their own applications on the databases and have complete rights to decide when they want to change their state such as joining in or dropping out from the system. Consequently, proposition 6 is supported.

Due to the discoverability of the services supported by the SOA and web service, each participating data source need only publish their data service descriptions into the central registry. Then the BS is able to find and access all the relevant data services when answering user queries through simply accessing the registry. In the handling of the evolutions occurring in local databases, the service description in the registry may only require modification when the global definition needs to be modified and the modifications made to the local definitions only will not affect the central registry. Hence the maintenance effort was reduced and proposition 6 is supported.

The abstraction of the service effects on the data services only needs to describe their global definitions, operations and locations in a standard way (i.e. service description), hence no further details of the local database and data service implementation were explicitly bound with participating data sources. Thus the service consumers can access the data services based on the simplified service descriptions and all the tedious work of achieving service communications was left to the web service implementation framework to finish, in this case, the .NET framework. Furthermore, the data sources can easily change the implementation of their data services thanks to the abstraction. Consequently, the maintenance effort was reduced and proposition 6 is supported.

The loose coupling of the services leads to each data service being independent of other data services. The data sources are not concerned with whether there are other participating data sources involved in the system when they publish their data services hence each data service has no information about other participating data sources to maintain. Thus the schematic evolution occurring in one database will not affect the data services provided by other participating databases which have been verified by proposition 4. In addition, the service consumers are not coupled with specific data services. Instead, the relevant data services were only located and accessed during the query processing hence the service consumer will not be affected by the evolutions occurring in the participating databases which further reduce maintenance effort. Consequently, the proposition 6 is supported.

Furthermore, based on the characteristics supported by the SOA and web service as described above, the service late binding applied in SLEDI results in relevant data services for answering user queries only being located and accessed at run time, instead of at design time. In other words, the data services only publish their service description in the central registry, and the service consumers do not know what data services are available and how to access them at design time. Thus when modifications were made to the data services for handling the evolution, no other parts of the system were affected, hence the impact on the system brought by the evolution was largely reduced. Consequently, proposition 6 is supported.

In summary, through supporting the service autonomy, discoverability, abstraction and loose coupling, the SOA and web service provided substantial help in applying the service late binding in the design and implementation of the SLEDI solution to largely reduce the maintenance costs caused by the evolution occurring in the participating databases. Therefore proposition 6 is well supported.

8.6 Scalability

As the SLEDI solution is designed for integrating data from autonomous and evolving data sources, it is important the SLEDI can still perform properly on reducing the maintenance costs as it is designed for when the number of the participating databases increases. Thus another important characteristic, the scalability of the SLEDI, was also evaluated during the case study. The evaluation was conducted through reproducing the four existing participating databases to join into the SLEDIS and then examining various aspects related to the scalability. As each participating database shares its data through constructing a set of local and global definitions, the relationship between the total number of the (pairs of) local and global definitions and the number of the participating databases was examined first. Then, since the schematic evolutions occurring in the participating databases were handled through examining and modifying the local and global definitions, the relationship between the number of affected local and global definitions and the number of the participating databases was also examined. The number of the affected local and global definitions was obtained through firstly applying a set of pre-defined schematic evolutions to the participating databases in the same way as introduced in the table 8-17, and then calculating the average number of the affected local and global definitions of one participating database when all the schematic evolutions were applied. The results are show in the figure 8-2 below:

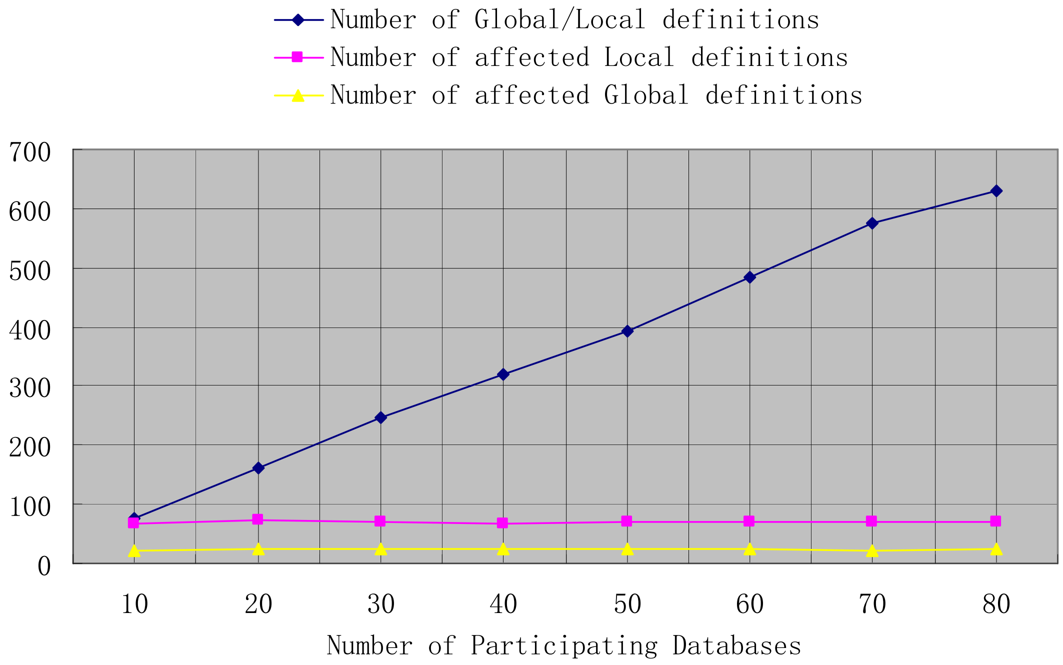


Figure 8-2 Relationship between the numbers of global/local definitions, affected global/local definitions and the number of participating databases

It can be seen that the number of global/local definitions had linear growth in the number of participating databases, while the numbers of the affected global and local definitions approximately remained stable, regardless of the number of the participating databases. The growth of the global/local definitions was as expected because every participating database provides a set of global and local definitions. The numbers generally remaining unchanged indicates that the amount of maintenance effort required did not increase when more databases joined in. Therefore, the SLEDI solution provides satisfying scalability at this stage.

And then, the relationship between the computational time cost of EHS for handling the schematic evolutions and the number of the participating databases was further examined. In order to make the evaluation thorough, the computational time cost was counted separately with respect to each type of schematic evolution through calculating the average time cost of EHS for handling one type of schematic evolution applied to one participating database. The results are show in the figure 8-3 below:

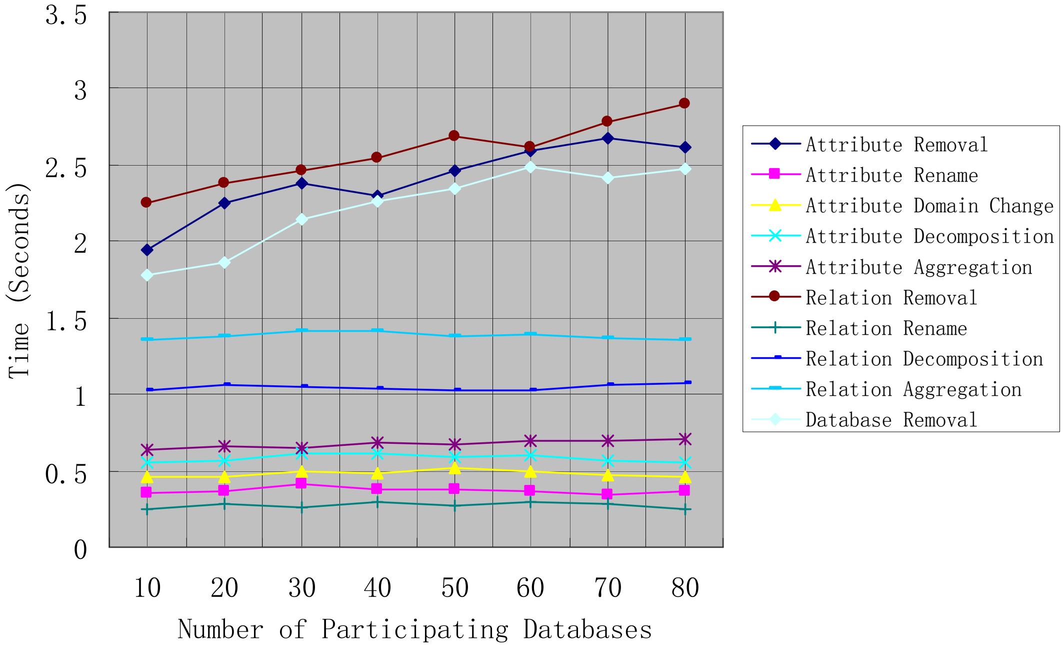


Figure 8-3 Relationship between the computational time costs for handling schematic evolutions and the number of participating databases

It can be seen that the computational costs for handling the attribute removal, relation removal and database removal were slightly raised in a merely linear fashion along with the increasing number of the participating databases. This is because the handling of the three types of schematic evolution may involve modifications to the global definitions. Since the increasing number of participating databases leads to a growth in the number of global definitions and all the global definitions were stored as metadata in the central registry, more time may be required for traversing the central registry to achieve the modifications. On the other hand, the time costs of handling the other types of schematic evolution generally remained unchanged, regardless of the increasing number of the participating databases. This was due to those schematic evolutions only requiring modification to the local definitions which only the evolved participating databases were involved in. The results further strengthen the evidence that SLEDI provides satisfying scalability.

In summary, the above results demonstrated that although the time costs of handling some types of schematic evolution were slightly raised when the number of participating databases increased, the number of global and local definitions affected by the schematic evolution and the time costs of handling most types of schematic evolution generally remained unchanged despite the growing number of participating databases. This differs from the traditional solution such as FDBMS, where all the relationships among the participating database have to be considered at design time. The complexity of the system may be dramatically escalated when the number of participating databases grows large. Therefore, the SLEDI solution supports good scalability.

8.7 Other Characteristics

In addition to the characteristics evaluated above, some other aspects of the SLEDI were also evaluated during the case study including the expandability, programming language independency and application domain independence of the SLEDI. The results are discussed as follows:

8.7.1 Expandability

As stated in section 1.2, all the data sources involved in this research are assumed to employ RDBMS to manage their data, in other words, all the participating databases are relational databases which realize the relational data model. Hence the databases which realize other types of data model, such as Object-Oriented databases, xml databases, flat files and legacy databases, were not considered in the research, thus cannot be directly integrated into the system through the data source describing process in the SLEDI. However, although the local definition of the IPU mechanism were particularly designed to build virtual views over relational database schemas, the database schemas were represented as a set of relations and the views were constructed in the form of conjunctive queries over the relations. The actual data in the databases were accessed through translating the conjunctive queries into local SQL queries which the databases can directly process. Since the conjunctive query and relation are abstract forms for representing data, the databases realizing data models other than the relational model, may still be integrated into the system through data source describing, if they can provide mechanisms for representing their data in the form of relations and programs to translate the conjunctive queries into local queries they can process.

The global definition of the IPU mechanism is constructed as a conjunctive query over relations in the application domain ontology, and the IPUs are accommodated through data services. They do not have dependency on the data model of the participating database thus other types of database may be integrated into the system through adding proper wrappers to produce data services. In conclusion, the SLEDI solution has the potential to integrate the types of participating databases other than relational database, although this capability has not been supported in the current stage. Further research is required to achieve this capability, not only on the data source describing for other database types, but also on the query processing and evolution handling problems newly-introduced by the database types.

8.7.2 Programming Language Independency

Although the experimental system SLEDIS was developed using the C# programming language and ASP.Net web service toolkit supported by the .Net framework, the SLEDI solution was designed by following SOA and the components of the SLEDI, such as data services, were designed as standard web services. This means in principle that the implementation of the SLEDI is not bound to any specific programming language because one of the main characteristics of the SOA and web service is the programming language independency. Each data service provider (e.g. participating database) only needs to describe the service (e.g. data service) it provides in a standard way (i.e. through WSDL), the service consumers will be able to access the services based on the description and communicate with the service through exchanging standard SOAP messages regardless of what programming language has been employed for the service providers to implement the services.

The two main platforms supporting the implementation of web service based applications currently available are .Net framework and J2EE. Both platforms provide a set of programming languages, toolkits and APIs to facilitate web service developing such as C#, java, Asp.Net and Axis. Thus the services implemented by J2EE may work together with the services developed by the .Net framework because all the services declare their operations and parameters in their service description, and both platforms are able to generate and process the service descriptions and the SOAP messages in the service invocations.

The language independency was also examined during the case study through implementing a data service in two versions where one version was developed by .Net and another version was developed by J2EE. The results show the .Net version worked well with other parts of the SLEDIS whereas the J2EE version encountered some problems. One problem was that the data of parameters and returning results were packed differently in the .Net service and in the J2EE service. This caused the service consumer (i.e. the BS) to be modified in order to correctly parse the results data obtained from the J2EE service. In conclusion, although the web service is supposed to have complete language independency, the current tools available did not achieve its full potential, although this situation may change with future releases of the tools.

8.7.3 Application Domain Independency

Although the case study was conducted in a single application domain and there is no cross domain data usage applied to the experimental implementation system, the SLEDI solution was designed to be abstract from particular applications. For example, the data source describing process constructs local and global definitions based on the assumption that all the participating databases were relational databases. Thus the databases can be from any application domain as long as they realized the relational data model. The application domain ontology was constructed based on the description logic and the user queries were represented in the format of conjunctive query. In addition, the schematic evolution examined was focusing on the evolution of attributes and relations in the relational data model which were also abstract from specific applications.

It can be seen from the constructed local and global definitions, as introduced in the early sections that all the views defined in the local definitions represented the information in an abstract fashion (i.e. conjunctive queries) although the information is all from the single application domain. Besides, the data models and algorithms of the SLEDI solution, such as the local and global definition data model, the IPUD algorithm, the evolution description model and the evolution handling process were all described by abstract symbols without adhering to any specific application domain. Therefore, it can be concluded that the SLEDI solution supports good application domain independency.

8.7.4 Limitations

Apart from the results discussed above, some issues were also raised during the case study which may arguably signify that the SLEDI solution may not be perfectly suitable under certain circumstances. The issues are discussed as follows:

- As introduced in chapter 6 and 7, although all the organizational evolutions ever occurring in the system were stored in the central registry to keep track of transitions over multiple versions of the organizational structure, only the latest version of the organizational structure was maintained. Consequently, because all the user queries were answered with respect to the latest version of the organizational structure, the user queries are not supported at this stage if they intend query multiple versions of the organizational structure. This issue requires further research.
- As mentioned in chapter 1, data inconsistency has not been addressed in the SLEDI solution through assuming there is no conflict among the data in different participating databases if they represent the same instance in the application domain. This assumption may not always be applicable in practice. For example, it is possible that a patient was recorded by different

names in different databases. This may indicate the occurrence of data inconsistence error in the system. This issue may be solved through further research

- The schematic evolutions were identified and handled immediately after they occurred in the participating databases. This requires the database administrator to identify the evolution in atomic type, represent each atomic evolution in the data model provided and send the data to EHS to trigger the schematic evolution handling. However, in practice, this involves a considerable amount of manual work and a set of schematic evolutions may have happened in short time which brings further work to the administrators of the participating databases. This issue may also require further research.
- As mentioned previously, the types of databases other than relational database, may not be integrated into the system directly at this stage, because different types of database may cause the heterogeneity and evolution problems differently. As other types of databases may exist in practice, this issue also requires further research.
- As the SLEDI solution follows the virtual data integration and service-based approaches, the efficiency of query answering may not be as good as traditional solutions such as data warehousing which follow materialized data integration and hard code program approaches. Since the query processing in SLEDI not only involves the process of rewriting user queries with respect to the available global definitions, but also the communications with the data services, the SLEDI may not be a suitable solution for the data integration applications which focus on query answering efficiency, rather than on evolution maintenance.

8.8 Conclusion

Through conducting the case study, all the research questions defined were answered by verifying the propositions and the propositions were examined through obtaining the results of the measures. The test data of the case study covered all the heterogeneities defined in chapter 1 and all the evolutions defined in the thesis. Moreover, the schematic evolution applied during the case study covered all possible evolution handling processes. Consequently, the case study conducted can be considered as a representative case.

Apart from the capabilities of the SLEDI solution of solving the heterogeneity and evolution problems, other characteristics such as scalability, expandability, programming language independency and application domain independency were also discussed. Despite some limitations being found during the case study, the evaluation criteria were successfully met based on the results.

Therefore, the conclusion can be drawn that the SLEDI solution constructed in this research has the ability of solving most of the heterogeneity and evolution problems defined in the thesis. The main design purpose of the SLEDI solution, to reduce the maintenance effort of the data integration system, has been achieved.

8.9 Summary

This chapter has presented an extensive evaluation of the SLEDI solution through conducting a case study. The characteristics of the SLEDI such as its ability to solve heterogeneity and evolution problems; its scalability, expandability, programming language and application domain independency were discussed in detail. The SLEDI solution was deemed as success.

Chapter 9 concludes the thesis by summarizing the heterogeneity and evolution problems in the data integration and the solution presented in this research. The success of the research is discussed and ideas for further research are suggested.

Chapter 9 Conclusions

9.1 Introduction

Chapter 8 presented an extensive and detailed evaluation of the SLEDI solution. The research questions were answered. The propositions were supported by the results of the measures obtained from the case study. General characteristics were examined followed by the discussion of the limitations found during the evaluation.

This chapter reviews the research presented in this thesis. The work accomplished is considered with reference to the research aims and criteria for success as defined in Chapter 1. Some general issues and suggestions for further work are also discussed.

9.2 Review of Research

9.2.1 Research aims and issues

This thesis investigated the data integration from autonomous data sources where the participating databases are distributed, heterogeneous and frequently evolving due to the autonomy. Through following the virtual data integration and service based approaches, the Service Late binding Enabled Data Integration (SLEDI) solution is designed to reduce the maintenance effort of the data integration system caused by the evolutions occurring in the participating databases. The two objectives the SLEDI solution intends to achieve are:

1. Construction of a unified vision from the data supplied by the data sources, in order to fulfill the information requirements from end users.
2. Providing automatic assistance for handling the evolution occurring at the data source level, in order to decrease the maintenance costs arising from the evolution.

Data source describing, query processing and evolution handling were identified as three major research issues. Data source describing integrates the data provided by the participating databases into a virtual integrated database and solves the database heterogeneities. Query processing answers user queries through dynamically identifying the relevant participating databases and obtaining the actual results data from the databases. Evolution handling employs automatic assistance to examine the evolutions occurring in the participating databases and modifies the affected components of the system accordingly in order to reduce the maintenance effort.

9.2.2 Service based approach

Chapter 2 explored the virtual and materialized data integration approaches and looked at some existing solutions following the two different approaches, including federated database systems and data warehousing. Which applications the solutions were best applied to were discussed through comparisons between the various aspects of the solutions. The weaknesses of the solutions on dealing with database evolutions were analyzed. Then the concepts and technologies of service based approach were introduced such as Data as a Service (DaaS), Service-Oriented Architecture (SOA) and Web Service technology. Then, the potential of the data integration solution to reduce the maintenance costs of handling the database evolutions through following the service based approach was investigated. Finally, the case study research method employed by this research and the application domain used for conducting the case study, were briefly introduced.

9.2.3 Data integration framework

A descriptive framework was constructed in chapter 3 to characterize the data integration activity through examining various aspects of the activity. The framework is specialized in integrating data from autonomous and evolving data sources. It captures the essential processes for solving the heterogeneity problems of the distributed participating databases to produce a unified integrated database which end users can communicate with directly to fulfil their information needs. More importantly, the framework explicitly describes the process of solving the problems in the data integration caused by the evolutions occurring in the participating databases. The framework characterizes the data integration activity from an abstract view angle where both the virtual and materialized data integration approaches can be fitted in, and provides a context for the SLEDI solution constructed in this research.

9.2.4 SLEDI solution

The SLEDI solution presented in this thesis addresses the three major research issues reviewed in section 9.2.1 through three processes. The detailed delineation of each process includes the algorithms, data models and sub-processes which the process may involve were described throughout chapter 4 to chapter 6. Chapter 3 presented the overview of the general architecture, components and services of the SLEDI for realizing the three processes.

The first process of data source describing was presented in chapter 4. Based on the Information Provision Unit (IPU) mechanism and the Information Provision Unit Describing (IPUD) algorithm, each data source describes the data it is willing to share as a set of IPUs. As each IPU is equipped with a global definition and a local definition, the mappings between the local schemas of the participating databases and the application domain ontology are established. Thus the databases heterogeneity problems are addressed by the IPUs. Through representing the global and local definitions of the IPUs as Direct Acyclic Graph (DAG) structured data and accommodating the IPUs into Information Provision Services (IPSs), the IPUs are encapsulated as metadata of the service descriptions of the IPSs. In addition, the IPSs are organized into an organizational structure which is also represented as DAG structured data and stored in the central registry. Consequently, further processes can be realized through exploiting those constructed data. The formal model of the global and local definitions, organizational structure and service description metadata were also presented.

Then the process of query processing was presented in chapter 5. The process takes the responsibility of answering user queries and involves three sub-processes: data source filtering, query rewriting and result generating. Data source filtering identifies the relevant participating databases for answering user queries; query rewriting transforms user queries in terms of application domain ontology into resulting queries in terms of global definitions of the IPUs through adopting the PICSEL query rewriting algorithm; and result generating obtains the results data of resulting queries through accessing the IPSs and producing the final results data for answering user queries.

The final process is evolution handling which was presented in chapter 6. It involves handling the organizational and schematic evolutions. The organizational evolution is handled through inserting a process as the first step of the data source filtering of the query processing. The schematic evolutions is tackled through identifying the global and local definitions of the IPUs which are affected by the evolution, and then automatically modifying the affected global and local definitions through the 10 processes defined with respect to various types of schematic evolution.

Some characteristics of the SLEDI solution such as the complexity, flexibility and scalability of each process were discussed with respect to the related works and the data integration framework in chapter 3. Furthermore, various types of evolution were identified, described and discussed in chapter 6.

9.2.5 Case study and experimental implementation

The main focus of this research is on reducing the maintenance costs of the data integration system caused by the evolutions occurring in the participating databases. It is impractical to implement a complete data integration system based on the SLEDI solution in a single research by a single researcher, since the frequency of different types of evolution may be vary largely in practice. A single case study based on the experimental implementation system: Service Late

Binding Enabled Data Integration System (SLEDIS) was conducted in the mental health application domain in order to evaluate the SLEDI solution.

Various issues involved in the design and implementation of the SLEDIS were discussed in chapter 7. As the SLEDI solution follows the service based approach, the design of the SLEDIS employed both the service design and Object-Oriented design methods. The component services of the SLEDIS which realizes the algorithms and processes of the SLEDI solution and the cooperation among the services was presented. The SLEDIS was developed in the .Net environment and the services were implemented by using the C# programming language and .Net web service toolkits. The implementation of the metadata was also described. Furthermore, in order to thoroughly evaluate the solution, the research questions, the propositions for answering the research questions and the measures for examining the propositions were also presented.

Chapter 8 presented an extensive and detailed evaluation of the SLEDI solution based on the case study. A set of local schemas and the application domain ontology were designed, and the capability of solving heterogeneity problems was investigated based on the global and local definitions constructed from the local schemas. Most types of heterogeneity problems defined in chapter 1 were successfully solved while the missing item discrepancy and relation isomorphism discrepancy may result in incomplete results for answering user queries due to the PICSEL query rewriting algorithm adopted in the solution. The capability of solving the evolution problems was examined through applying the evolution designed covering all the types of evolution defined in chapter 6 and analyzing the results data of user queries. Through examining the results of the measures obtained in the case study, all the propositions were verified as generally supported. In addition, the scalability of the SLEDI solution was investigated, the results showed the number of global and local definitions affected by the various types of schematic evolution and the time costs for handling the evolution generally remained unchanged, while the number of global and local definitions had a slow linear growth when the number of the participating databases increased. The SLEDI solution has good programming language independency and application domain independency. Finally, some limitations of the solution were also discussed.

9.3 Evaluation of Research

This research is evaluated with respect to the research aims and criteria for success as defined in chapter 1. The discussion of the evaluation is presented as follows:

1. The definition of a framework for integrating data from the autonomous and evolving data sources is constructed. The framework should capture the essential processes involved in the data integration activity.

Chapter 2 defined a descriptive framework for data integration. The framework specializes in solving the heterogeneity and evolution problems in the data integration due to the autonomy of the participating databases. It explicitly describes the processes involved for achieving the required data integration.

2. The creation of the Service Late-binding Enabled Data Integration (SLEDI) solution which defines the processes and data structures involved in the solution precisely with respect to the framework discussed in criterion 1.

Chapter 3 presented the overview of the SLEDI solution through briefly introducing its realization of the three processes defined in the descriptive framework. The detailed description of the algorithms, data models and sub-processes involved in each process was presented through chapter 4 to chapter 6.

3. The schematic heterogeneities of the participating databases defined in section 1.2.3 can be solved by the Information Provision Unit Describing (IPUD) algorithm created for the data source describing in the solution.

Chapter 4 described the IPU mechanism and IPUD algorithm. The formal representation of the global and local definitions, view definitions and validation rules of the views involved in the IPUD were defined. The results in chapter 8 demonstrated that the heterogeneity problems can be solved.

4. The organizational and schematic evolutions introduced in section 1.3.1 can be handled by the evolution handling process constructed in the solution allowing the data integration system to still functions properly despite the evolutions which occurred.

Chapter 6 delineated the evolution handling processes covering various types of evolution defined in section 1.3.1. The details of handling each different types of evolution were presented as processes and the results in chapter 8 showed that the evolution can be successfully handled by the SLEDI solution.

5. The DAG structured data source descriptions can support the evolution handling process hence reducing the maintenance effort of the data integration system caused by the evolutions.

Chapter 4 described how the global and local definitions were represented as DAG structure data. Using the structured data instead of hard coded programs, the automatic modification can be conducted on the global and local definitions for handling the evolutions occurring in the participating databases. The results in chapter 8 showed that the DAG structure data and the evolution handling processes can reduce the maintenance effort caused by the evolutions

6. The DaaS approach and Service Late Binding Technique can help to mitigate the maintenance costs of the data integration system caused by the evolution introduced in

section 1.3.1

Various characteristics of the DaaS approach and Service Late Binding Technique were covered in chapter 2 and the component services based on the web service were described in chapter 7. Through realizing the global and local definitions as metadata of the IPS and central registry, the evolutions can be handled through automatically modifying the metadata and the relevant data services are bound at run time instead of at design time. Hence the maintenance costs can be reduced and this was discussed in chapter 8.

It has been demonstrated that the work presented in this thesis meets the research aims and criteria for success, as defined in chapter 1. The accomplishments and ideas for continuing work which may be carried out based on this research are discussed in the following sections.

9.4 Discussion

A reflective discussion of the accomplishments achieved in this research is presented as follows. In general, the SLEDI solution is a success as it meets the requirements defined in chapter 1.

Applying the DaaS approach and Late Binding Technique has been considered to be successful. The IPU mechanism is a good initiative as it allows participating databases to flexibly describe what data they are willing to share. The global and local definition helps to precisely represent the mappings between the local database schemas and application domain ontology. The use of the conjunctive query has been a good idea as it provides a foundation for constructing the local definitions and represents the definitions as structured data. The adoption of the PICSEL query rewriting algorithm is a good choice as only the global definitions need to be accessed for finding the relevant databases in query answering and the global definitions can be dynamically changed without affecting the query answering process. Although the query rewriting may result in high costs and produce incomplete results under some circumstances, the global and local definitions and the query answering process provide a base for reducing the maintenance costs.

One of the major successes of this research has been encapsulating the local and global definitions into the metadata of the data services and central registry. The structured metadata has been demonstrated to be effective as all the data constructed in the data source describing process can be realized into the metadata. It has been the key mechanism in avoiding hard coded programs. Hence automatically modifying the metadata can be implemented to handle the evolutions occurring in the participating databases which reduce human effort involved in the maintenance work.

The data source filtering in query processing handles the eight types of organizational evolution through accessing the metadata in the central registry which records all the organizational evolutions occurring, thus the existing user queries do not require modification to adapt to the evolutions. The query rewriting has been proved to be effective in the original work and the result generating produces results data through service invocations. The query processing has been proved to be correct and effective.

The evolution handling process has been demonstrated to be effective. The 10 types of schematic evolution were solved by 10 processes to modify the metadata of the data services based on verifying the validation rules of the view definitions involved in the local definitions and global definitions. The processes have been proved to be correct and effective. Although the handling of the addition of attributes, relations and databases still requires human intervention, the automatic evolution handling based on the processes largely reduced the human effort involved in the maintenance work. One issue concerning the current schematic evolution handling process is that each evolution is supposed to be handled sequentially and immediately. However, this may be improved in further work.

The Service Late Binding based on the SOA and web service has been proved to be helpful in reducing the maintenance costs. Participating databases can join in or drop out easily through publishing data services. The system level evolution was solved through modifying the descriptions of the data services and the schematic evolution handling processes were carried out through automatically modifying the metadata of the data services. The characteristics of service autonomy, discoverability, abstraction and loose coupling facilitate the modification of the data services hence the maintenance cost is reduced and the relevant data services always reflect the latest state of the participating databases and can only be bound at the time the user queries being answered.

Comparing the SLEDI solution with the related work such as the federated database system and data ware housing has proved an interesting exercise. SLEDI is similar to the federated database system as they both following the virtual data integration approach. However, SLEDI provides better flexibility for handling evolution as the mappings were represented as structured metadata instead of hard coded programs. Overall, the SLEDI solution has been proved to be a successful data integration solution specializing in handling various types of evolution occurring in the participating databases.

9.5 Further Work

The work presented in this thesis could be extended in many ways and some ideas suggesting possible further works are discussed as follows:

9.5.1 User queries involving multiple versions of organizational structure

As mentioned in previous chapters, a data integration system based on the SLEDI solution can only answers the user queries based on the latest version of the organizational structure. This is because, although all the organizational evolutions occurring are recorded, only the latest version of the organizational structure is maintained in the central registry. Thus a user query cannot be directly answered if it requires results data over a period of time and organizational evolutions occurring during this time. For example, a user query asks for all the patient information from the databases belonging to virtual group A in 2009 and a database DB_1 was in group A in 2009 but dropped out before the user query was answered. Under these circumstances, the DB_1 would not be considered, although it should be.

A possible solution is to add an extra property in the organizational structure to identify its version number and maintain all the versions of the organizational structure in the central registry. Thus the user queries can indicate the versions of the organizational structure they intend to cover and the relevant databases can be properly filtered out through accessing the right versions of the organizational structure. Under the circumstances of the above example, all the versions of organizational structure during 2009 would be accessed. However, further work is required on this issue.

9.5.2 Data Inconsistency

As introduced in chapter 1, it is assumed that the data in the different participating databases are all consistent. However, the assumption may not always be true in practice. For example, the information about the same patient obtained from participating databases have identical patient ID numbers but different patient names. Under these circumstances, it is required to verify which information should be considered as reliable. A possible solution is using simple rules to validate the inconsistent information. In this example, a device can be designed to compare all the patient names obtained and select the majority answer or simple pick one randomly. This issue would be subject for further research.

9.5.3 Other models for representing application domain ontology

As introduced in chapter 4, the application domain ontology in SLEDI is represented in the model based on description logic. Employing other models to represent the application domain ontology may facilitate the effectiveness of constructing the data integration systems based on SLEDI. For example, the local schemas of the participating databases may have a large amount of relations in common, then the application domain ontology may be represented as relational schema and the common relations can be directly used in the application domain ontology. It is important to realize that this change may cause the modifications to the global definition representation and the query rewriting algorithm hence any potential benefits should be weighed against this change. The issue may require further research.

9.5.4 Other types of participating databases

As introduced in chapter 1, all the participating databases involved in this research are relational databases. In practical situations, other types of databases such as Object-Oriented databases, xml databases and flat files may exist. Integrating other types of databases could be a fruitful line of research. As discussed in chapter 8, since the local definitions are constructed as virtual views over local databases schemas and the views are defined in a conjunctive query which is an abstract form, the SLEDI solution has the potential to integrate other types of participating database. The local definitions of other types of database may still be constructed if the databases can describe their data into a set of (virtual) relations through some mechanisms (e.g. a wrapper). However, the heterogeneity problems may appear differently in other types of databases and hard coded queries may not be completely avoidable if the mechanisms have to be realized through programs instead of structured data. This may raise the complexity of evolution handling. The complexity may still be mitigated by using the experience gained from this research: using structured data as much as possible to replace the hard coded programs to reducing the maintenance costs. This issue would be subject for further research.

9.5.5 Extension of schematic evolution handling

Currently, schematic evolution is handled sequentially and immediately. The rationale of schematic evolution handling is to modify the local definitions to make them valid again with respect to the latest version of the local databases schemas and the evolution handling process which is triggered manually by the administrators of the local databases. A possible research direction is to change the current rationale to follow the similar fashion of organizational evolution handling. This may be achieved through representing the transitions between every two adjacent versions of local schemas. The relationship between the original version and the latest version of local schemas may be formally built if the transitions are represented in a well defined way. Hence the original version of local schemas when the local definitions were constructed may be established based on the latest version of local schemas even though only the latest version is accessible and more than one schematic evolution has already occurred. Under these circumstances, instead of modifying the affected local definitions every time the local schema evolves, only the transitions are recorded. The local queries produced based on local definitions which are defined over the original version of the local schema may be transformable into equivalent queries with respect to the latest version of local schema. Since the modifications required for handling the schematic evolution are to the transitions instead of to the local definitions, the amount of modification work may be reduced. Furthermore, automatic assistance may be employed to construct the transitions without human intervention hence the maintenance costs may be further reduced. However, the query processing may have to be changed through adding the query transformation process and this issue may require further research.

9.5.6 Large scale evaluation

The evaluation in chapter 8 provides a large amount of useful information about various characteristics of the SLEDI solution, especially on using automatic tools for conducting evolution handling. A large scale evaluation can be undertaken such as the effect of considerably increasing the items in the application domain ontology, the number of attributes and relations in the local schemas, the number of participating databases and the number of evolutions applied. This can further investigate evolution handling to determine the effectiveness of the tool when used in a practical situation and may guide the development of further research on the method and tools.

9.6 Final Summary

A review of the work accomplished in this research has been presented in this chapter. The overall success of the research has been discussed with respect to the research aims and criteria for success defined in Chapter 1. Several directions for further work have been suggested.

This thesis has examined the context, motivation, and definition of integrating data from autonomous and evolving data sources, leading to the development of a framework to describe the data integration activity. A new solution following a service based approach and employing a service late binding technique called Service Late binding Enabled solution has been presented. The mechanisms and algorithms for accomplishing the processes of data source describing, query processing and evolution handling have been described and compared to other similar systems. An extensive evaluation based on a case study has demonstrated various characteristics of the solution through obtaining the results of measures to verify the propositions. Issues found in the case study were discussed and ideas for further work were also suggested.

The SLEDI solution includes the IPU mechanism and IPUD algorithm for data source describing. The procedures addressing query processing and evolution handling offer novel and successful solutions for integrating data from autonomous and evolving data sources.

Reference:

[01] Mario Antonioletti , Malcolm Atkinson , Rob Baxter , Andrew Borley , Neil P. Chue Hong , Brian Collins , Neil Hardman , Alastair C. Hume , Alan Knox , Mike Jackson , Amy Krause , Simon Laws , James Magowan , Norman W. Paton , Dave Pearson , Tom Sugden , Paul Watson , Martin Westhead, 'The design and implementation of Grid database services in OGSA-DAI', Concurrency and Computation: Practice and Experience, Volume 17, Issue 2-4, Year of Publication: 2005, Pages 357 – 376.

[02] Yigal Arens, Craig A. Knoblock, Wei-Min Shen, Query reformulation for dynamic information integration Source, Journal of Intelligent Information Systems, ISSN:0925-9902, Year of Publication: 1996, Volume 6, Issue 2-3, Pages: 99 – 130.

[03] Yigal Arens, Chun-Nan Hsu, Craig A. Knoblock, Query processing in the SIMS information mediator Source, Readings in agents, ISBN:1-55860-495-2, Year of Publication: 1997, Pages: 82 – 90.

[04] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider, The description logic handbook: theory, implementation, and applications, ISBN:0-521-78176-0, Pages: 5-8, Year of Publication: 2003, Cambridge University Press.

[05] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider, The description logic handbook: theory, implementation, and applications, ISBN:0-521-78176-0, Pages: 51-55, Year of Publication: 2003, Cambridge University Press.

[06] V R Basili, R W Selby, D H Hutchens, Experimentation in software engineering, IEEE Transactions on Software Engineering, ISSN:0098-5589, Volume 12, Issue 7 (July 1986), Pages: 733 – 743.

[07] C. Batini, M. Lenzerini, S. B. Navathe, A comparative analysis of methodologies for database schema integration, ACM Computing Surveys (CSUR), ISSN:0360-0300, Volume 18, Issue 4, Year of Publication: 1986, Pages: 323 – 364.

[08] R. J. Bayardo, Jr., W. Bohre, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, D. Woelk, InfoSleuth: agent-based semantic integration of information in open and dynamic environments Full text, Proceedings of the 1997 ACM SIGMOD international

conference on Management of data, ISBN:0-89791-911-4, Year of Publication: 1997, Pages: 195 – 206.

[09] C Beeri, A Y Levy, M-C Rousset, Rewriting queries using views in description logics, Proceeding PODS '97 Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, ISBN:0-89791-910-6, Year of Publication: 1997, Pages: 99 – 108.

[10] Sonia Bergamaschi, Silvana Castano, Maurizio Vincini, Domenico Beneventano, Semantic integration of heterogeneous information sources, Data & Knowledge Engineering, Volume 36 Issue 3, March 2001.

[11] K.H.Bennett, S.Bradley, G.Glover, D.Barnes, "Software Evolution in an Interdisciplinary Environment", Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice, ISBN:0-7695-2218-1, Year of Publication: 2003, Pages: 199 – 203.

[12] Keith H. Bennett, Václav T. Rajlich, Software maintenance and evolution: a roadmap, Proceedings of the Conference on The Future of Software Engineering, Proceedings of the Conference on The Future of Software Engineering, Year of Publication: 2000, Pages: 73 – 87.

[13] K.H.Bennett, J.Xu, "Software Services and Software Maintenance", Proceedings of the Seventh European Conference on Software Maintenance and Reengineering, 2003, ISBN:0-7695-1902-4 pp.3.

[14] Paul V. Biron, Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004.

[15] Ronald J. Brachman, Hector J. Levesque, Knowledge Representation and Reasoning, ISBN:1558609326, Year of Publication: 2004.

[16] S.P. Bradley, K.H. Bennett, "Mental Health Minimum Data Set (MHMDS)", September 2002

[17] Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, Günter Kniesel, Towards a taxonomy of software change: Research Articles, Journal of Software Maintenance and Evolution: Research and Practice, ISSN:1532-060X, Volume 17, Issue 5 Year of Publication: 2005, Pages: 309 – 332.

[18] Diego Calvanese, Giuseppe De Giacomo, M. Lenzerini, Maurizio Lenzerini, On the Decidability of Query Containment under Constraints, Proceeding PODS '98 Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, ISBN:0-89791-996-3, Year of Publication: 1998, Pages: 149 – 158.

[19] Diego Calvanese, Maurizio Lenzerini, Daniele Nardi, Riccardo Rosati, Description logic framework for information integration, Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98), ISBN: 1-55860-554-1, Year of Publication: 1998, Pages: 2 – 13.

[20] S. Chaudhuri, U. Dayal. An Overview of Data Warehousing and OLAP Technology, SIGMOD Record, ISSN: 0163-5808, Year of Publication: 1997, Volume 26, Issue 1.

[21] Peter Pin-Shan Chen, The entity-relationship model: toward a unified view of data, ACM Transactions on Database Systems, ISSN:0163-5840, Volume 10, Issue 3, Pages: 9 – 9, Year of Publication: 1975.

[22] E.F.Codd, The relational model for database management: version 2, ISBN:0-201-14192-2, Year of Publication: 1990, Addison-Wesley Longman Publishing Co., Inc.

[23] C.J.Date, An Introduction to Database System, 8th Edition, ISBN: 978-0321197849, Pages: 26 – 29, Year of Publication: 2003.

[24] C.J.Date, An Introduction to Database System, 8th Edition, ISBN: 978-0321197849, Pages: 173 – 250, Year of Publication: 2003.

[25] N. J. Davies, D. Fensel, M. Richardson, The Future of Web Services Full text, BT Technology Journal, ISSN:1358-394, Year of Publication: 2004, Volume 22, Issue 1, Pages: 118 – 130.

[26] Thomas Erl, Service-Oriented Architecture: Concepts, Technology, and Design, ISBN:0131858580, Year of Publication: 2005.

[27] Daniela Florescu, Alon Levy, Alberto Mendelzon, Database Techniques for the World-Wide Web: A Survey, ACM SIGMOD Record, Volume 27 Issue 3, Sept. 1, 1998, Pages: 59 – 74.

[28] Bent Flyvbjerg, Five Misunderstandings about Case-Study Research, Qualitative Inquiry,

Volume 12, Number 2, April 2006, Page 219-245.

[29] Ian Foster, Carl Kesselman, 'The grid: blueprint for a new computing infrastructure', ISBN:1-55860-475-8, Year of Publication: 1998.

[30] Ian Foster, Carl Kesselman, Steven Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". International Journal of High Performance Computing Applications, ISSN:1094-3420, Year of Publication: 2001, Volume 15, Issue 3, Pages: 200 – 222.

[31] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey Ullman, Vasilis Vassalo, Jennifer Widom, The TSIMMIS Approach to Mediation: Data Models and Languages, Journal of Intelligent Information Systems, ISSN:0925-9902, Volume 8, Issue 2, Year of Publication: 1997, Pages: 117 – 132.

[32] Globus Toolkit 4.0, http://www.globus.org/toolkit/docs/4.0/key/index.html, last accessed in August 2005.

[33] Gyles Glover, Adult mental health care in England, European Archives of Psychiatry and Clinical Neuroscience, ISSN: 0940-1334, 257:71–82, (2007).

[34] François Goasdoué, Véronique Lattès and Marie-Christine Rousset, The Use of CARIN Language and Algorithms for Information Integration: The PICSEL System, International Journal of Cooperative Information Systems (IJCIS), ISSN: 0218-8430, Volume 9, Issue 4 (December 2000), Pages: 383-401.

[35] Alistair Grant, Mario Antonioletti, Alastair C. Hume, Amy Krause, Bartosz Dobrzelecki, Michael J. Jackson, Mark Parsons, Malcolm P. Atkinson, Elias Theocharopoulos, OGSA-DAI: Middleware for Data Integration: Selected Applications, Proceedings of the 2008 Fourth IEEE International Conference on eScience, ISBN:978-0-7695-3535-7, Year of Publication: 2008, Page 343.

[36] Thomas R. Gruber, A translation approach to portable ontology specifications, Knowledge Acquisition, Volume 5, Issue 2, June 1993, Pages: 199-220, ISSN:1042-8143.

[37] Alon Y Halevy, Answering queries using views: A survey. The International Journal on Very Large Data Bases J., 10(4):270–294, 2001.

[38] Alon Y Halevy, Anand Rajaraman, Joann Janet Ordille, Data integration: the teenage years,

Proceedings of the 32nd international conference on Very large data bases, Pages: 9 - 16 , 2006.

[39] Ian Horrocks, Peter F. Patel-Schneider, Frank Van Harmelen, From SHIQ and RDF to OWL: The Making of a Web Ontology Language, Journal of Web Semantics, Volume 1, Number 1, Pages: 7-26, Year of Publication: 2003.

[40] David K. Hsiao, Federated databases and systems: part I --- a tutorial on their data sharing, The International Journal on Very Large Data Bases, ISSN:1066-8888, Volume 1, Issue 1, Year of Publication: 1992, Pages: 127 – 180.

[41] Richard Hull, Gang Zhou,A framework for supporting data integration using the materialized and virtual approaches, Proceedings of the 1996 ACM SIGMOD international conference on Management of data, ISBN:0-89791-794-4, Year of Publication: 1996, Pages: 481 – 492.

[42] R. Hull, Managing semantic heterogeneity in databases: A theoretical perspective, In Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97), 1997.

[43] IEEE standard for software maintenance, 1219-1998, Software Engineering Standards Committee of the IEEE Computer Society, ISBN: 0-7381-0336-5, Publication Date: 21 Oct 1998.

[44] M. Jarke, M. Lenzerini, and P. Vassiliadis Y. Vassiliou, Fundamentals of Data Warehouses, Springer Verlag, 2000. ISBN 3540420894, pp19-21

[45] Jena – A Semantic Web Framework for Java, http://jena.sourceforge.net/, last accessed in December 2007.

[46] Vipul Kashyap, Amit Sheth, Semantic and schematic similarities between database objects: a context-based approach, The VLDB Journal — The International Journal on Very Large Data Bases, ISSN:1066-8888, Volume 5, Issue 4 Year of Publication: 1996, Pages: 276 – 304.

[47] Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam, Jarrett Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Transactions on Software Engineering, ISSN:0098-5589, Volume 28, Issue 8 (August 2002), Pages: 721 – 734.

[48] Barbara Kitchenham, Lesley Pickard, Shari Lawrence Pfleeger, Case Studies for Method

and Tool Evaluation, IEEE Software, ISSN:0740-7459, Volume 12, Issue 4 (July 1995), Pages: 52 – 62.

[49] Dirk Krafzig, Karl Banke, Dirk Slama, "Enterprise SOA", Prentice Hall Professional Technical Reference, ISBN: 0-13-146575-9 p. 56-57, 2004.

[50] Wilburt J. Labio, Yue Zhuge, Janet L. Wiener, Himanshu Gupta, Héctor García-Molina, Jennifer Widom, The WHIPS prototype for data warehouse creation and maintenance, Proceedings of the 1997 ACM SIGMOD international conference on Management of data, ISBN:0-89791-911-4, Year of Publication: 1997, Pages: 557 – 559.

[51] Stéphane Lafortune, Eugene Wong, 'A State Transition Model for Distributed Query Processing' August 1986, ACM Transactions on Database Systems (TODS), Volume 11 Issue 3.

[52] P.J.Layzell, K.H.Bennett, D.Budgen, P.Brereton, L.A.Macaulay, M.Munro, 'Service-Based Software: The Future for Flexible Software' Asia-Pacific Software Engineering Conference, 5-8 December 2000, ISBN: 0-7695-0915-0 pp. 214-222.

[53] Fritz Lehmann, Semantic Networks in Artificial Intelligence, ISBN:0080420125, Year of Publication: 1992, Pages: 768.

[54] Meir.M.Lehman, Juan F. Ramil, "Software Evolution and Software Evolution Process", Annals of Software Engineering, ISSN:1022-7091, Volume 14 , Issue 1-4 (December 2002).

[55] Maurizio Lenzerini, Data integration: A theoretical perspective. Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2002), pages 233-246, 2002.

[56] Alon Y. Levy, Anand Rajaraman , Joann J. Ordille, Query-Answering Algorithms for Information Agents, AAAI'96 Proceedings of the thirteenth national conference on Artificial intelligence, ISBN: 0-262-51091-X, Year of Publication: 1996, Pages: 40 – 47.

[57] Alon Y. Levy, Anand Rajaraman, Joann J.Ordille, Querying Heterogeneous Information Sources Using Source Descriptions Source, Proceedings of the 22th International Conference on Very Large Data Bases, ISBN:1-55860-382-4, Year of Publication: 1996, Pages: 251 – 262.

[58] Alon Y Levy, Marie-Christine Rousset, The limits on combining recursive horn rules and description logics, Proceeding AAAI'96 Proceedings of the thirteenth national conference on

Artificial intelligence, ISBN:0-262-51091-X, Volume 14 , (1996), pages 577-584.

[59] Alon Y. Levy, Marie-Christine Rousset, Combining Horn rules and description logics in CARIN, Artificial Intelligence, ISSN:0004-3702, Volume 104, Issue 1-2 (September 1998), Pages: 165 – 209.

[60] Alon Y. Levy, Logic-based techniques in data integration, Logic-based artificial intelligence, ISBN: 0-7923-7224-7, Year of Publication: 2000, Pages: 575 – 595.

[61] B.P.Lientz, E.B.Swanson, Software Maintenance Management, Addison-Wesley Publishing Company, 1980, ISBN 0201042053.

[62] David Martin, Mark Burstein, Drew Mcdermott, Sheila Mcilraith, Massimo Paolucci, Katia Sycara, Deborah L. Mcguinness, Evren Sirin, Naveen Srinivasan, Bringing Semantics to Web Services with OWL-S, World Wide Web, ISSN:1386-145X, volume 10 Issue 3, September 2007.

[63] Marvin Minsky, A framework for representing knowledge, Computation & intelligence: collected readings, ISBN:0-262-62101-0, Year of Publication: 1995, Pages: 163 – 189.

[64] OMII uk, http://www.omii.ac.uk/wiki/SoftwareOverview, last accessed in April 2007.

[65] P. Oreizy, R. Taylor, On the Role of Software Architectures in Runtime System Reconfiguration, Proceedings of the International Conference on Configurable Distributed Systems, ISBN: 0-8186-8451-8, Page: 61, Year of Publication: 1998.

[66] OWL Web Ontology Language Overview, W3C Recommendation 10 Feburary 2004. http://www.w3.org/TR/owl-features/, last accessed in November 2007.

[67] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, Katia Sycara, Semantic Matching of Web Services Capabilities, ISWC 2002 In The Semantic Web, Pages: 333 – 347.

[68] Michael P. Papazoglou, Willem-Jan Van Den Heuvel, Service-Oriented design and development methodology, International Journal of Web Engineering and Technology archive, ISSN:1476-1289, Volume 2, Issue 4, Pages: 412-442, Year of Publication: 2006.

[69] Mike P. Papazoglou, Willem-Jan Heuvel, Service-Oriented architectures: approaches, technologies and research issues, The VLDB Journal — The International Journal on Very Large Data Bases, ISSN:1066-8888, Year of Publication: 2007, Volume 16 , Issue 3 (July 2007),

Pages: 389 – 415.

[70] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann, "Service-Oriented Computing: State of the Art and Research Challenges", Computer, ISSN:0018-9162, Year of Publication: 2007, Volume 40, Issue 11, Pages 38-45.

[71] Pellet: OWL 2 Reasoner for Java, http://clarkparsia.com/pellet/, last accessed in March 2008.

[72] Vijayshanker Raman, Inderpal Narang, Chris Crone, Laura Haas, Susan Malaika, Tina Mukai, Dan Wolfson and Chaitan Baru, 'Data Access and Management Services on Grid', Database Access and Integration Services Working Group, Global Grid Forum (GGF) 5, 2002.

[73] RDF Primer, W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-primer/, last accessed in November 2009.

[74] RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-schema/, last accessed in November 2009.

[75] Semantic Annotations for WSDL and XML Schema, W3C Recommendation 28 August 2007, http://www.w3.org/TR/2007/REC-sawsdl-20070828/, last accessed in March 2008.

[76] A. Sheth, J. Larson, 'Federated database systems for managing distributed, heterogeneous and autonomous databases' ACM Computing Surveys, 1990, 22, 3, pp. 183-236.

[77] Amit. P. Sheth, "changing focus on interoperability in information systems: from system, syntax, structure to semantics", Michael F. Goodchild, Max J. Egenhofer, Robin Fegeas, Cliff Kottman "Interoperating geographic information systems", ISBN 0792384369, Published by Springer, 1999, pages 5-30.

[78] Alan Simon, Steven Shaffer, Data warehousing and business intelligence for e-commerce, ISBN: 978-1-55860-713-2, Year of Publication: 2002.

[79] Munindar P. Singh, Michael N.Huhns. "Service-Oriented Computing" John Wiley & Sons, Ltd, ISBN 0-470-09148-7, 2005.

[80] SOAP Version 1.2 Part 0: Primer (Second Edition), W3C Recommendation 27 April 2007, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/, last accessed in December 2007.

[81] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation 27 April 2007, http://www.w3.org/TR/2007/REC-soap12-part1-20070427/, last accessed in December 2007.

[82] SPARQL Query Language for RDF, W3C Recommendation 15 January 2008. http://www.w3.org/TR/rdf-sparql-query/, last accessed in November 2009.

[83] The Protege; Ontology Editor and Knowledge Acquisition System, http://protege.stanford.edu/, last accessed in December 2007.

[84] Duane P. Truex, Richard Baskerville, Heinz Klein, Growing systems in emergent organizations, Communications of the ACM, ISSN:0001-0782, Volume 42, Issue 8, Year of Publication: 1999, Pages: 117 – 123.

[85] M. Turner, Zhu, F., Kotsiopoulos, I., Russell, M., Budgen, D., Bennett. K., Brereton, P., Keane, J., Layzell, P., and Rigby, M., 'Using Web Services Technologies to create an Information Broker: An Experience Report', Proceedings of the 26th International Conference on Software Engineering, ISBN ~ ISSN:0270-5257 , 0-7695-2163-0, Year of Publication: 2004, Pages: 552 – 561.

[86] UDDI v3.0, http://www.uddi.org/pubs/uddi_v3.htm, last accessed in May 2006.

[87] J.D. Ullman, Database and Knowledge-Base Systems, Volumes I and II. Computer Science Press 1989

[88] J. D. Ullman, Information integration using logical views. In Proc. of the 6th Int. Conf. on Database Theory (ICDT'97), volume 1186 of Lecture Notes in Computer Science, pages 19–40. Springer, 1997.

[89] Web Services Addressing 1.0 – Core, W3C Recommendation 9 May 2006, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/, last accessed in October 2006.

[90] W3 Working Group, Web Services Architecture W3C Working Group Note 11, February 2004, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, last accessed in December 2005.

[91] Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation 9 November 2005, http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/, last

accessed in November 2006.

[92] Web Services Description Language (WSDL) Version 2.0 Part 0: Primer, W3C Recommendation 26 June 2007, http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626, last accessed in January 2008.

[93] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Recommendation 26 June 2007, http://www.w3.org/TR/wsdl20/, last accessed in January 2008.

[94] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, Donald F. Ferguson, Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More, Prentice Hall PTR, ISBN:0131488740, Year of Publication: 2005.

[95] Jennifer Widom, Research Problems in Data Warehousing, Conference on Information and Knowledge Management Proceedings of the fourth international conference on Information and knowledge management, ISBN:0-89791-812-6, Year of Publication: 1995, Pages: 25 – 30.

[96] XML Schema Part 0: Primer Second Edition, W3C Recommendation 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/, last accessed in February 2006.

[97] XML Schema Part 1: Structures Second Edition, W3C Recommendation 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/, last accessed in February 2006.

[98] XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004. http://www.w3.org/TR/xmlschema-2/, last accessed in March 2006.

[99] XQuery 1.0 and XPath 2.0 Data Model (XDM) W3C, Recommendation 23 January 2007, http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/, last accessed in July 2007.

[100] XQuery 1.0 and XPath 2.0 Formal Semantics W3C, Recommendation 23 January 2007, http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/, last accessed in July 2007.

[101] S.S. Yau, J.S. Collofello, T. MacGregor, Ripple effect analysis of software maintenance, Computer Software and Applications Conference, 1978. COMPSAC '78. pp. 60-65, 1978.

[102] Robert K. Yin, Applications of case study research, second edition, Publisher: SAGE Publications 2003, Series: Applied Social Research Methods, Volume 34, ISBN:

9780761925507.

[103] Robert K. Yin, Case Study Research Design and Methods, Fourth Edition, Pages: 1-11, Publisher: SAGE Publications (28 December 2008 - Thousand Oaks, United States), ISBN: 1412960991.

[104] Robert K. Yin, Case Study Research Design and Methods, Fourth Edition, Pages: 12-17, Publisher: SAGE Publications (28 December 2008 - Thousand Oaks, United States), ISBN: 1412960991.

[105] Gang Zhou, Richard Hull, Roger King, Generating data integration mediators that use materialization, Journal of Intelligent Information Systems, ISSN:0925-9902, Year of Publication: 1996, Volume 6 Issue 2-3, Pages: 199 – 221.

[106] F. Zhu, M. Turner, I. Kotsiopoulos, K. Bennett, M. Russell, D. Budgen, P. Brereton, J. Keane, P. Layzell, M. Rigby, and J. Xu, 'Dynamic Data Integration Using Web Services', Proceedings of the IEEE International Conference on Web Services, ISBN:0-7695-2167-3, Year of Publication: 2004, Page: 262.