Chapter 3 Research Method

In this chapter we further describe the integration problems and present our research method and research structure. We propose the approach to tackle the research problems addressed in Chapter One. We focus on the resolution of the heterogeneity problems among information integration over the Internet. This research approach hopes to provide systematic and methodological information integration.

3.1. Research Method

Liang (1997) summarized the MIS research methods. He stated that MIS scholars held a series of conferences on research methods in 1989, and identified the five primary research methods including (1) case study, (2) survey, (3) experiment, (4) model driven, and (5) prototyping. Taking the five methods into consideration, prototyping is suitable and fit to be applied in this research.

3.2. Research Structure

According to the literature reviewed in Chapter Two, there have been many works focusing on heterogeneous information integration. Typical information integration systems have adopted mediator/wrapper architecture (Wiederhold, 1993). Under such architecture, the mediator provides an integrated and global view of different heterogeneous information sources. With this view, queries can be formulated by the clients. Besides, wrappers provide local views of information sources in a uniform data model. The local views can be queried in a limited way according to wrapper capabilities.

TSIMMIS, DISCO, Garlic, Information Manifold and so on as described in Chapter Two were the methods which have adopted mediator/wrapper architecture. They focused on providing an integrated data model that is an object model. However, beginning in the 21st century, XML has taken the place of object model as the pivot model. XML has become an emerging standard of data exchange and has many advantages to become the best candidate to be the common data model when performing heterogeneous information integration.

However, the information integration studies which adopt mediator/wrapper architecture and use XML as the common data model to capture heterogeneous sources have met with semantic problems, but only syntactical and structure ones. Ontology from the field of artificial intelligence describes the knowledge representation that provides definitions of vocabulary in certain domain. The use of ontology to explicate and explore the implicit and hidden knowledge seems to be a promising approach to tackle the problems of semantic heterogeneity. Therefore, we add ontology and develop an information integration model and method that is based on mediator/wrapper architecture to solve the heterogeneity problems over the heterogeneous information sources. Users can thus access heterogeneous information sources via one uniform and seamless platform.

The research structure is illustrated in Figure 3-1.

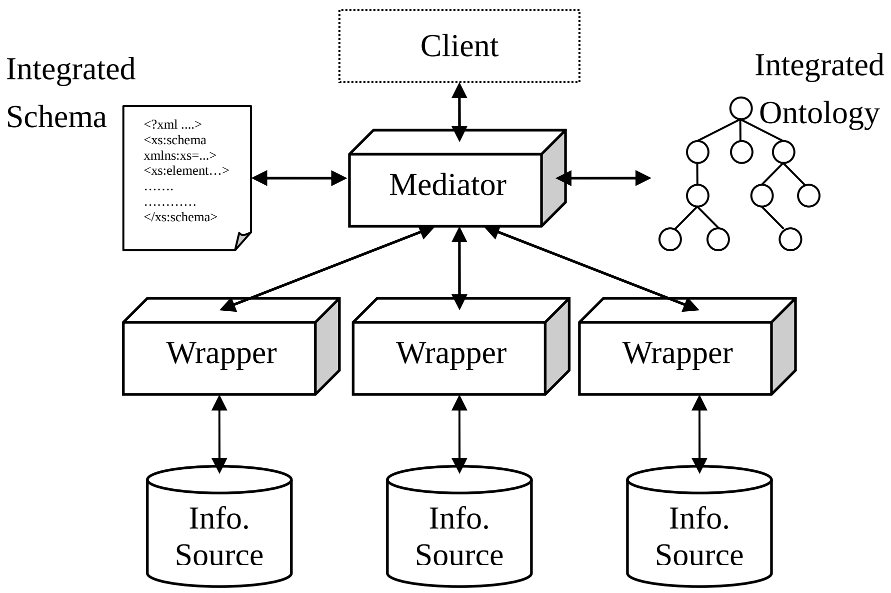


Figure 3-1: Research Structure

At the bottom of Figure 3-1 there are a number of information sources which contain diverse information that needs integration. Different information sources present their own data in a different data model so the client has to use different access interfaces to get the data, and at the same time, take the following details into consideration, such as the location of data, effectiveness and efficiency of accessing different information sources, data quality, and consistency if an update is performed. To overcome the above difficulties, we construct corresponding wrappers for different types of information sources. The wrapper is used to translate data access and manipulation requests between mediator and information sources. Above each wrapper in the figure is a mediator, in charge of query processing in the research structure. In addition, the mediator provides the client with the integrated view of the underlying heterogeneous information sources and processes clients' queries against the information sources. We will describe the data model integration issues and process in Section 3.3, query resolution process in Section 3.4.

In the following, we describe the research structure in detail. The components in the research structure are depicted in Figure 3.2.

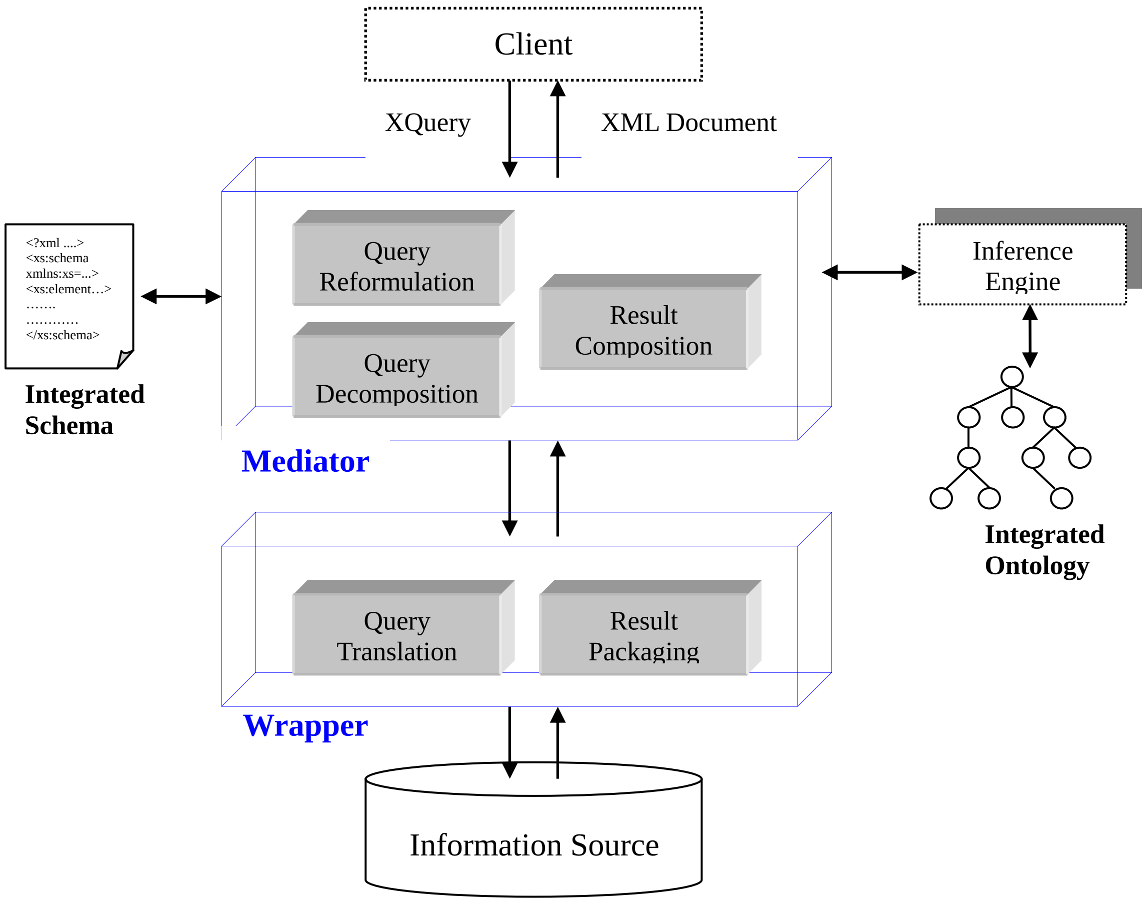


Figure 3-2: Components in Research Structure

As the Figure 3-2 shows, there are two major parts in our research structure: (1) Mediator (2) Wrapper. We describe the functions of the individual component of each part as follows.

First, the components in the mediator:

(a) Query reformulation is used to receive the query from system query interface and then send out a reasoning request according to the query from the interface to the inference engine in order to find out the implicit knowledge and relationships in that query. The inference engine then gets the reasoning result and passes it back to the mediator, in which query reformulation component receives this result. The component, according to the result, reformulates the query to represent the facts in an explicit form. Afterwards, it passes the reformulated queries to the query decomposition component for further process.

(b) Query decomposition receives the reformulated query from query reformulation component. After receiving the query, it decomposes the query into several sub-queries according to the integrated schema and the specified mapping between global schema and local schemas. Then it passes those sub-queries to the corresponding wrappers.

(c) Result composition receives the packaged results from wrappers and recombines the results into an XML document according to the user request. It may also require the assistance of the integrated schema while composing the results.

Second, the components in the wrapper:

(a) Query translation is used to receive the sub-query of the target source and then translate it into native query of that information source. After that, it sends the native query into the underlying information source for finding out the data demanded.

(b) Result packaging gathers the native results and packages them in a form that is known by the mediator. Then, it sends the packaged result to the mediator for further process.

In the following Section 3.3, we illustrate the information integration method in research structure. That is the backbone of our research structure.

3.3. Information Integration Method in Research Structure

In this section, we detail our methods of information integration in our research structure. Our goal is to provide a convenient and effective way for users to access a number of heterogeneous information sources simultaneously and get an integrated result just like accessing only one information source. Users who interact with the information integration structure do not have to consider the details of the information sources they face. To achieve this goal, we must integrate the underlying sources and provide users with a unified view of the structure and content of these sources. Providing the unified view depends on the integration of different data models of the underlying information sources. Hence, integrating different data models of the underlying sources is significant and helpful.

But before performing the data model integration, we must identify problems that we will meet in the information integration. Problems coming from heterogeneity of the data are already well known within the distributed database systems community: (Cui, Jones, & O'Brien, 2001; Wache, Vögele, Visser, Stuckenschmidt, Schuster, Neumann, & Hübner, 2001).

1. The system level of heterogeneity includes incompatible hardware and software systems, which results in a variety of different access mechanisms and protocols.
2. The syntactic level of heterogeneity refers to different languages and data representations;
3. The structural level includes different data models;
4. The semantic level considers the contents of an information item and its intended meaning.

XML is widely predicted to improve the degree of interoperation on the Internet. Yet XML does not address ontology and provides only a syntactic and structure representation of knowledge. For this reason, we use XML as the uniform data model for performing HII with ontology assisted for the dimension of the semantics. We would like to present the details of our methods of HII as follows. And we use an example which is about the domain of university to explain our method of information integration.

3.3.1. The Creation of Global Schema

When performing heterogeneous information integration, we first encounter the representation problem for the structure of different data models. Parent et al. 1998 formalized the database integration process in order to develop an integrated schema (see Figure 3-3). To establish the integrated schema as a unified view of existing information sources, the heterogeneous schemas of the corresponding underlying information sources are usually transformed to make them as homogeneous as possible. Researchers in database integration generally assume that input schemas are all expressed using the same data model, the so-called “common” data model.

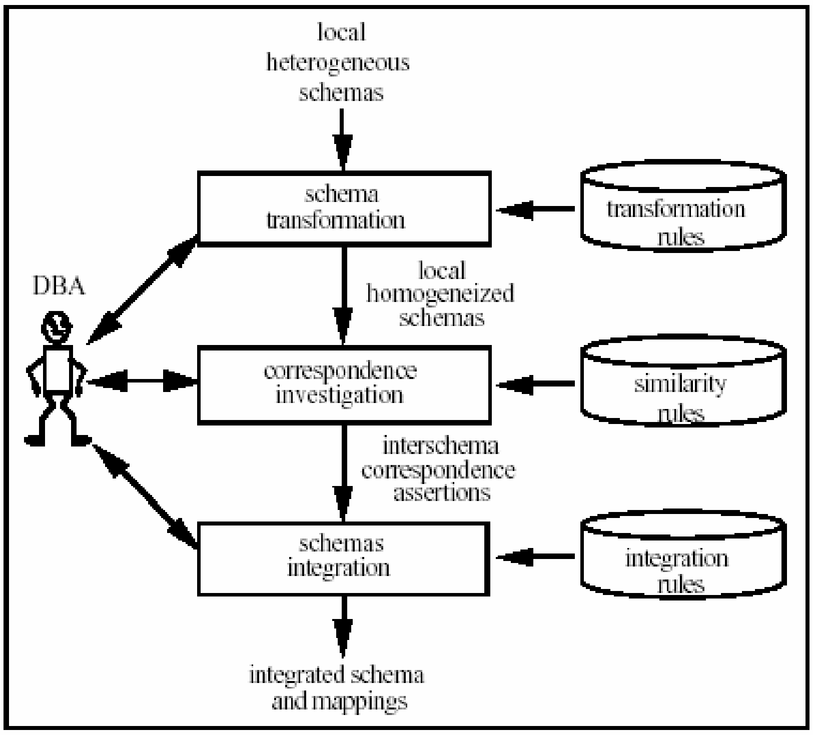

local heterogeneous schemas
schema transformation
transformation rules
DBA
local homogeneized schemas
correspondence investigation
similarity rules
interschema correspondence assertions
schemas integration
integration rules
integrated schema and mappings

Figure 3-3: The Global Integration Process

(Data Source: Parent, & Spaccapietra, 1998)

In this subsection, we extend the published database integration process to our information integration method to create the global view of the underlying information sources. In contrast to the traditional HII, they use an expert-dependent method to create their global schema. However, in this research, we try to provide a more general method to handle this issue. We use XML as the common data model to enable HII and propose two steps for the creation of global schema in our method, which are: (1) Generic Construct Oriented Schema Rewriting, and (2) Schema Integration. Performing the schema transformation by using the method of generic construct oriented schema rewriting is a more general and convienent way to apply to most kinds of information sources in contrast to the taditional method. That is our emphatic point.

3.3.1.1. Generic Construct Oriented Schema Rewriting

In order to homogenize the representations of the data models using in heterogeneous information sources, we have to create rules for rewriting between XML and the native data models. Since our information integration model is regarded as a generic model, it is expected to tackle any kinds of information sources. The heterogeneous information sources that we most often encounter can be roughly classified into three categories, which are: structured information sources, semi-structured information sources, and unstructured information sources. Structured information sources include Relational Database Management System (RDBMS) and Enterprise Information System (EIS, such as ERP, SCM, and CRM) files, among others. One example of semi-structured information sources may be Object Database Management System (ODBMS) or XML data files. Unstructured information sources may include HTML pages, multimedia files, office flies, and legacy files, and so on.

We deign to apply the generic construct oriented schema rewriting process to the structured and semi-structured information sources. The unstructured information source here is hard pressed to receive this type of HII pre-processes because it is lack of the structure definition, schema. As such, in our research structure we treat the unstructured information sources as special cases and they need an additional process described in the later sections.

To transform the data models of the structured and semi-structured information sources into XML, we have to specify one-to-one rewriting rules for every native data model. Before specifying the rewriting rules, we have to identify the correspondences between the constructs of XML and other native data models. Here we provide the correspondences between XML and two representative data models of structured and semi-structured information sources, which are a relational model and an object model as explained. Table 3-1 shows the correspondences of relational schema constructs and XML Schema constructs. According to the specified correspondences, the relational schema can be rewritten into a W3C XML Schema just as the example shown in Figure 3-5 & 3-6 describes.

Table 3-1: Correspondences between Relational Schema Constructs and W3C XML Schema Constructs

Relational Schema Constructs	**W3C XML Schema Constructs**
Relation	element (with xs:complexType)
Attribute	element
date type	date type (primitive type / xs:simpleType)
Cardinality	multiplicity (minOccurs / maxOccurs)
primary key (PK)	key (xs:key)
foreign key (FK)	keyref (xs:keyref)

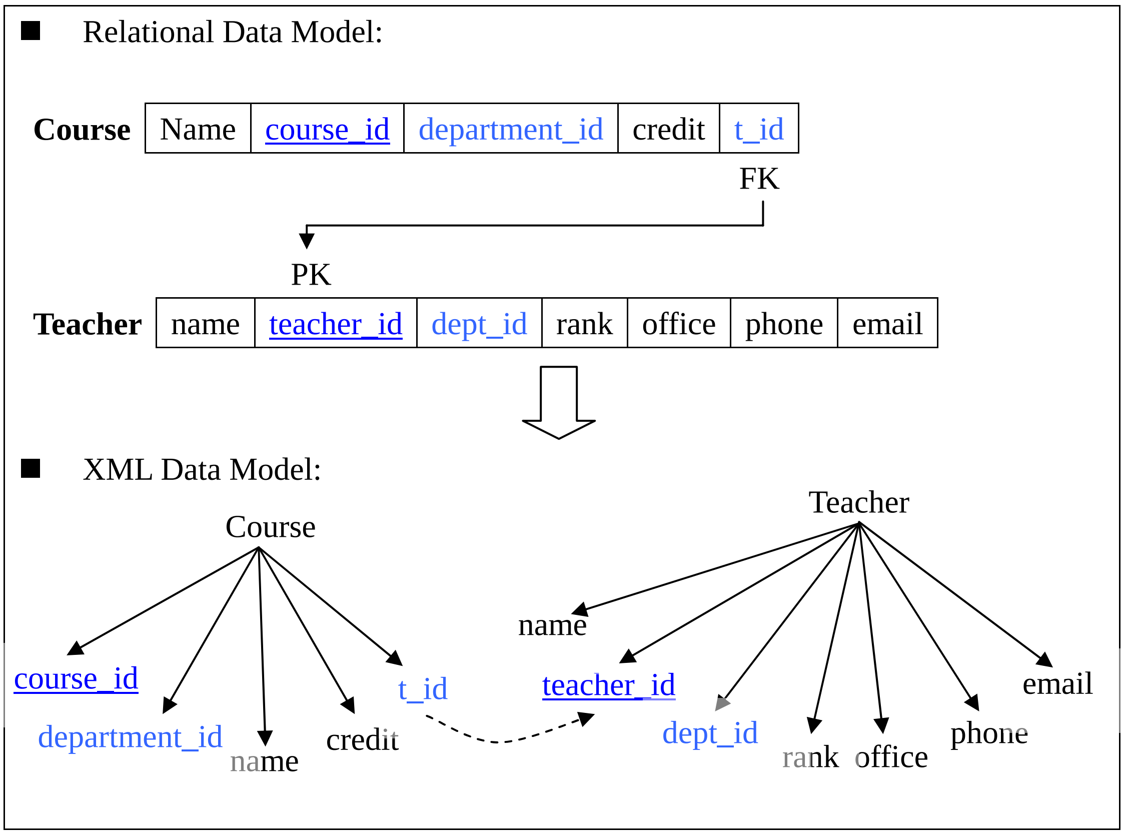


Figure 3-4: Transform Relational Data Model into XML Data Model

■ **Relational schema:**

Course (course_name(string), <u>course_id</u>(string), department_id(string), credit(int), t_id(string))

Teacher (name(string), <u>teacher_id</u>(string), dept_id(string), rank(string), office(string), phone(string), email(string))

$\Downarrow$

■ **W3C XML Schema (S1):**

```
<?xml version="1.0" encoding="Big5" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
- <xs:element name="Course">
  - <xs:complexType>
    - <xs:sequence>
        <xs:element name="course_name" type="xs:string" />
        <xs:element name="course_id" type="xs:string" />
        <xs:element name="department_id" type="xs:string" />
        <xs:element name="credit" type="xs:integer" />
        <xs:element name="t_id" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  - <xs:key name="Course_PrimaryKey">
      <xs:selector xpath="." />
      <xs:field xpath="course_id" />
    </xs:key>
  - <xs:keyref name="Course_To_Teacher" refer="Teacher_PrimaryKey">
      <xs:selector xpath="." />
      <xs:field xpath="t_id" />
    </xs:keyref>
  - <xs:keyref name="Course_To_Department"
      refer="Department_PrimaryKey">
      <xs:selector xpath="." />
```

```
        <xs:field xpath="department_id" />
      </xs:keyref>
    </xs:element>
- <xs:element name="Teacher">
  - <xs:complexType>
    - <xs:sequence>
          <xs:element name="name" type="xs:string" />
          <xs:element name="teacher_id" type="xs:string" />
          <xs:element name="dept_id" type="xs:string" />
          <xs:element name="rank" type=" xs:string" />
          <xs:element name="office" type="xs:string" />
          <xs:element name="phone" type="xs:string" />
          <xs:element name="email" type="xs:string" />
        </xs:sequence>
      </xs:complexType>
  - <xs:key name="Teacher_PrimaryKey">
        <xs:selector xpath="." />
        <xs:field xpath="teacher_id" />
      </xs:key>
  - <xs:keyref name="Teacher_To_Department"
        refer="Department_PrimaryKey">
        <xs:selector xpath="." />
        <xs:field xpath="dept_id" />
      </xs:keyref>
    </xs:element>
</xs:schema>
```

Figure 3-5: Rewrite Relational Schema into W3C XML Schema According to the Generic Constructs Correspondence

Table 3-2 shows the correspondences between object database schema constructs and XML Schema constructs. Similar to rewriting relational schema into W3C XML Schema, we provide a simple example in Figure 3-7 & 3-8 to illustrate the transformation between object database schema and XML Schema according to the specified correspondences in Table 3-2.

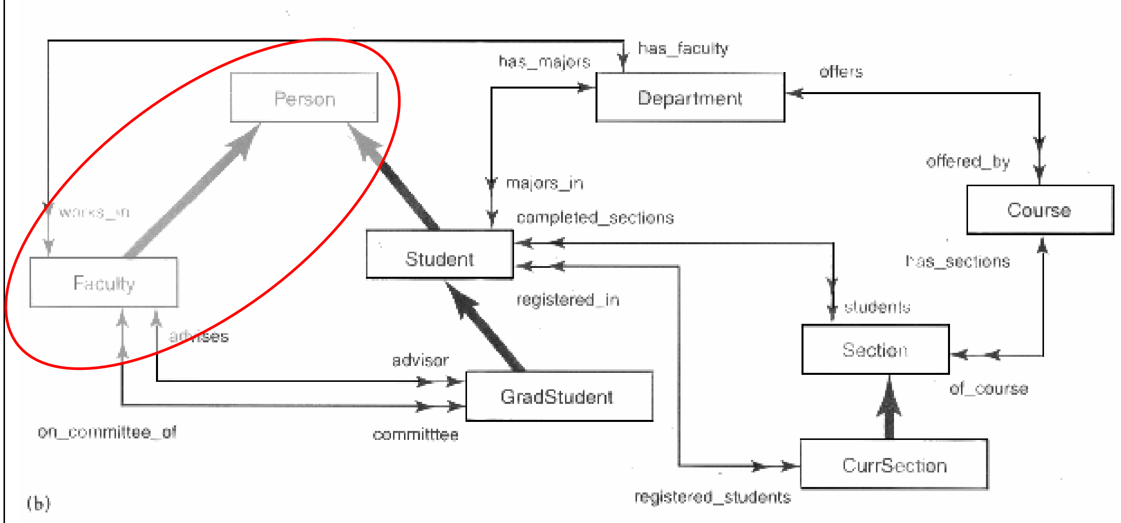
Table 3-2: Correspondences between Object Database Schema Constructs and W3C

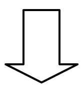
XML Schema Constructs

Object Database Schema Constructs	W3C XML Schema Constructs
class	element (with xs:complexType)
attribute (simple)	element
primitive type	data type (primitive type)
struct (user-defined type)	data type (xs:simpleType / xs:complexType)
key	key (xs:key)
extend (inheritance)	only single inheritance supported (xs:extension / xs:restriction)
relationship/inverse	Not Supported
extent	
method	

We use an object data model that is illustrated in (Elmasri, & Navathe, 2000) to be an example. We use just two classes, “Person” and “Faculty”, of the entire data model in order to show how to rewrite an ODL schema for object database into W3C XML Schema.

■ **Object Data Model:**
(Data Source: Elmasri, & Navathe, 2004)





■ **XML Data Model:**

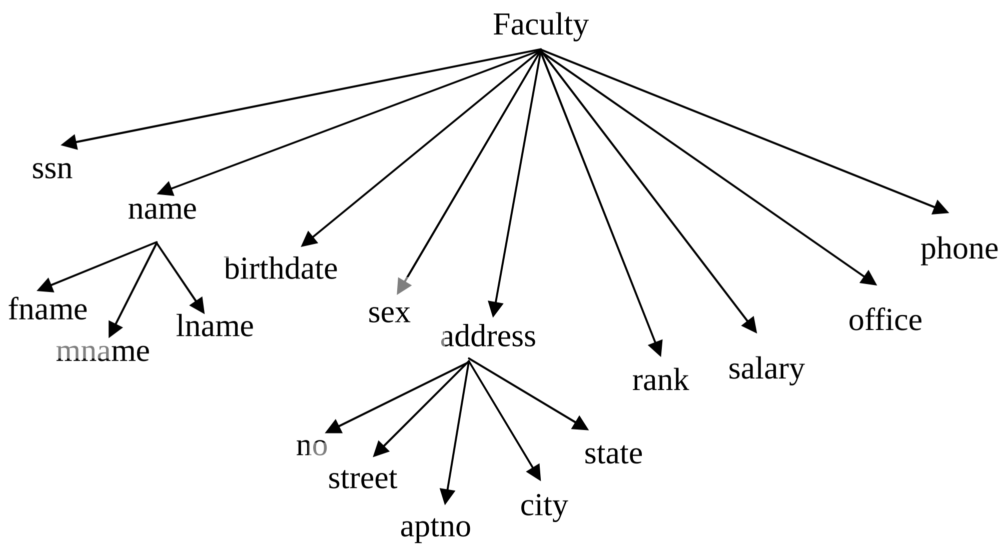


Figure 3-6: An Example of Transforming Object Data Model to XML Data Model

■ **ODL schema for object database:**

(Data Source: Elmasri, & Navathe, 2004)

```
class Person
(   extent persons
    key   ssn   )
{
    attribute struct Pname {string fname, string mname, string lname}
                                          name;
    attribute string                      ssn;
    attribute date                        birthdate;
    attribute enum Gender{M, F} sex;
    attribute struct Address
                {short no, string street, short aptno, string city, string state,
                 short zip}
                                          address;
    short age();
}
```

```
class Faculty extends Person
(   extent faculty   )
{
    attribute     string              rank;
    attribute     float               salary;
    attribute     string              office;
    attribute     string              phone;
    relationship Department      works_in inverse Department::has_faculty;
    relationship set<GradStudent> advises inverse GradStudent::advisor;
    relationship set<GradStudent> on_committee_of
                                     inverse GradStudent::committee;
    void give_raise(in float raise);
    void promote(in string new_rank);
}
```

⇩

■ **W3C XML Schema (S2):**

```
  <?xml version="1.0" encoding="Big5" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
  - <xs:element name="Person">
    - <xs:complexType name="PersonType">
      - <xs:sequence>
            <xs:element name="name" type="Pname" />
            <xs:element name="ssn" type="xs:string" />
            <xs:element name="birthdate" type="xs:date" />
            <xs:element name="sex" type="Gender" />
            <xs:element name="address" type="Address" />
        </xs:sequence>
      </xs:complexType>
    - <xs:key name="Person_Key">
        <xs:selector xpath="." />
        <xs:field xpath="ssn" />
      </xs:key>
    </xs:element>
```

```
- <xs:complexType name="Pname">
  - <xs:sequence>
      <xs:element name="fname" type="xs:string" />
      <xs:element name="mname" type="xs:string" />
      <xs:element name="lname" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
- <xs:simpleType name="Gender">
  - <xs:restriction base="xs:string">
      <xs:enumeration value="M" />
      <xs:enumeration value="F" />
    </xs:restriction>
  </xs:simpleType>
- <xs:complexType name="Address">
  - <xs:sequence>
      <xs:element name="no" type="xs:short" />
      <xs:element name="street" type="xs:string" />
      <xs:element name="aptno" type="xs:short" />
      <xs:element name="city" type="xs:string" />
      <xs:element name="state" type="xs:string" />
      <xs:element name="zip" type="xs:short" />
    </xs:sequence>
  </xs:complexType>
- <xs:element name="Faculty">
  - <xs:complexType>
    - <xs:complexContent>
      - <xs:extension base="PersonType">
        - <xs:sequence>
            <xs:element name="rank" type="xs:string" />
            <xs:element name="salary" type="xs:float" />
            <xs:element name="office" type="xs:string" />
            <xs:element name="phone" type="xs:string" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 3-7: An Example of Transforming Object Database Schema to XML Schema

In this thesis, we show just the generic construct correspondences between XML and two structured information sources, RDBMS and ODBMS, as explained. According to the correspondences, we can rewrite the native data model using in local information source into XML. Different information sources use different data model to describe their own data. To enable heterogeneous information integration, the one-to-one generic construct correspondences to rewrite the local data model into the common data model, XML, by using our integration structure is necessary.

3.3.1.2. Schema Integration

Before we integrate the schemas, we have to identify the commonalities between different schemas and characterize the inter-schema relationships. Schema integration uses the correspondences to find similar structures in heterogeneous schemas, which are then used as integration points.

However, in order to find out the correspondences between a set of independently developed schemas, we must recognize the causes for the structural heterogeneity between them in advance. We must gain the interoperability among the underlying sources by solving the heterogeneity problems between them so that we will achieve information integration. But the causes of the heterogeneity must be clarified first, and then we can deal with the heterogeneity problems by pointing out the correspondences between different schemas. Kashyap et al. (1996) and Visser et al. (2003) have categorized the causes for structural heterogeneity. We summarize the reasons for them in Table 3-3.

Table 3-3: Causes for Structural Heterogeneity

Causes	Explanations
Naming Conflict	These are of two types. Synonyms are the one which means that two attributes (or entities) that are semantically alike might have different names. Homonyms are the other one which means that two attributes (or entities) that are semantically unrelated might have the same names.
Domain Conflict	Two attributes that are semantically similar might have different domains or data types.
Default Value Conflict	This one depends on the definition of the domain of the concerned attributes. For example, the default value for age of an adult might be defined as 18 in one data source and as 21 in another.
Identifier Conflict	The primary keys of two entities in two sources are incompatible, because they use identifier records that semantically different. For example, the key of student entity might be defined as ID# in one source and as NAME in another.
Integrity Constraint Conflict	Two semantically similar attributes might be restricted by constraints which might not be consistent with each other. For example, the age of adult is defined to over 18 in one source and as to over 21 in another.
Missing Data Item Conflict	This conflict arises when, of the entity descriptors modeling semantically similar entities, one has a missing attribute.
Aggregation Conflict	These conflicts arise when an aggregation is used in one source to identify a set of entities (or attributes) in another source.
Attribute – Entity Conflict	This one arises when the same thing is being modeled as an attribute in one source and an entity in another source.
Data Value – Entity Conflict	It arises when the value of an attribute in one source corresponds to an entity in another source.
Data Value – Attribute Conflict	This conflict arises when the value of an attribute in one source correspond to an attribute in another source.

Those mentioned above are possible structural heterogeneity problems likely encountered while performing the schema integration to construct a global, unified schema to be the foundation of the information integration. It deserves consideration to construct a global schema with correspondences or mappings between the different source schemas to solve the structural conflicts between them and then gain the interoperability.

Therefore, we can analyze the structural heterogeneity problems between the schemas that are rewritten in XML we want to integrate according to the listed causes of recognition. In the following, we continue to use the example addressed in the previous subsection to explain the process of correspondences identification.

1. Naming Conflict: element “Teacher” using in schema S1 and element “Faculty” using in schema S3 are semantically the same but have different names.

   Correspondence: S1.Teacher = S2.Faculty

2. Aggregation Conflict: the aggregation of element “fname”, “mname”, and “lname” using in schema S2 is semantically the same.

   Correspondence:

   S1.Teacher.name = S2.Person.name (S2.Person.name.fname +

   S2.Person.name.mname + S2.Person.name.lname)

3. Identifier Conflict: the primary key of entity “Teacher” in source schema S1 is “teacher_id”, but the primary key of class “Faculty” in source schema S2 is not specified explicitly, that is “ssn” which is inherited from its parent class “Person”.

Afterward, we can specify the integration rules according to the identified correspondences to integrate the independent schemas into the global schema. Continuing the previous example, we can specify the following integration rules:

1. Correspondence: S1.Teacher = S2.Faculty

   Integration rule: G.Faculty

2. Correspondence: S1.Teacher.name = S2.Person.name (S2.Person.name.fname + S2.Person.name.mname + S2.Person.name.lname)

   Integration rule:

   G.Faculty.name (G.Faculty.name.fname + G.Faculty.name.mname + G.Faculty.name.lname)

3. Identifier Conflicts:

   Integration rule:

   Because the semantics of these two identifiers is a little different, we keep them separately in the integrated schema and use "teacher_id" to be the identifier of the integrated element "Faculty".

However, identifying the structural conflicts and correspondences between the independent schemas and specifying the integration rules for our method still needs the intervention of human experts. There are still some research efforts for automatic schema matching (Rahm, & Bernstein, 2001) for producing correspondences between different schemas. Once they are identified, matching elements can be unified under a coherent, integrated schema or viewed by using techniques like schema merge.

Finally, Figure 3-9 is a continuing example to illustrate an integrated schema built according to the integration rules we specify.

```
  <?xml version="1.0" encoding="Big5" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
  - <xs:element name="GlobalSchemaRoot">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="Person" minOccurs="0" maxOccurs="unbounded"
            />
          <xs:element ref="Faculty" minOccurs="0"
            maxOccurs="unbounded" />
          <xs:element ref="Course" minOccurs="0" maxOccurs="unbounded"
            />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Person">
    - <xs:complexType name="PersonType">
      - <xs:sequence>
          <xs:element name="ssn" type="xs:string" />
          <xs:element name="name" type="Pname" />
          <xs:element name="birthdate" type="xs:date" />
          <xs:element name="sex" type="Gender" />
          <xs:element name="address" type="Address" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:complexType name="Pname">
    - <xs:sequence>
        <xs:element name="fname" type="xs:string" />
        <xs:element name="mname" type="xs:string" />
        <xs:element name="lname" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  - <xs:simpleType name="Gender">
    - <xs:restriction base="xs:string">
        <xs:enumeration value="M" />
        <xs:enumeration value="F" />
      </xs:restriction>
    </xs:simpleType>
- <xs:complexType name="Address">
  - <xs:sequence>
        <xs:element name="no" type="xs:short" />
        <xs:element name="street" type="xs:string" />
        <xs:element name="aptno" type="xs:short" />
        <xs:element name="city" type="xs:string" />
        <xs:element name="state" type="xs:string" />
        <xs:element name="zip" type="xs:short" />
      </xs:sequence>
    </xs:complexType>
- <xs:element name="Faculty">
  - <xs:complexType>
    - <xs:complexContent>
      - <xs:extension base="PersonType">
        - <xs:sequence>
              <xs:element name="teacher_id" type="xs:string" />
              <xs:element name="dept_id" type="xs:string" />
              <xs:element name="rank" type="xs:string" />
              <xs:element name="salary" type="xs:float" />
              <xs:element name="office" type="xs:string" />
              <xs:element name="phone" type="xs:string" />
              <xs:element name="email" type="xs:string" />
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
  - <xs:key name="Faculty_PrimaryKey">
        <xs:selector xpath="." />
        <xs:field xpath="teacher_id" />
      </xs:key>
  - <xs:keyref name="Faculty_To_Department"
        refer="Department_PrimaryKey">
        <xs:selector xpath="." />
        <xs:field xpath="dept_id" />
      </xs:keyref>
    </xs:element>
  - <xs:element name="Course">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element name="course_name" type="xs:string" />
          <xs:element name="course_id" type="xs:string" />
          <xs:element name="department_id" type="xs:string" />
          <xs:element name="credit" type="xs:int" />
          <xs:element name="t_id" type="xs:string" />
        </xs:sequence>
      </xs:complexType>
    - <xs:key name="Course_PrimaryKey">
        <xs:selector xpath="." />
        <xs:field xpath="course_id" />
      </xs:key>
    - <xs:keyref name="Course_To_Faculty" refer="Faculty_PrimaryKey">
        <xs:selector xpath="." />
        <xs:field xpath="t_id" />
      </xs:keyref>
    - <xs:keyref name="Course_To_Department"
        refer="Department_PrimaryKey">
        <xs:selector xpath="." />
        <xs:field xpath="department_id" />
      </xs:keyref>
    </xs:element>
  </xs:schema>
```

Figure 3-8: An Integrated Schema in W3C XML Schema for the Example

3.3.1.3. Special Process for the Unstructured Information Sources

Unstructured information sources such as static web pages, multimedia files, etc. do not have "schema", so we must treat such information sources as a special case and provide special process for them. We create indexes of those sources and do not perform transformation on them. We simply wait until the global schema is created and specify the mapping between them, which will be described in a later section.

3.3.2. The Creation of Ontology

XML is a representation language for specifying the structure of the underlying information sources and thus their structure dimension. The structural representation can represent some semantic properties but it is not clear how this can be deployed outside of a special purpose application. To allow for a real semantic interpretation for HII, the common data model, XML, must be complemented by a conceptual model that adequately describes the domain we want to perform the information integration. This role cannot be filled by just XML data model (Erdmann, & Decker, 2000).

Using an ontology containing facts and relationships about the application domain of interest as the conceptual model to capture real world knowledge may be a promising approach. However, most ontology creation is carried out on a manual basis. There are a number of publications about ontological development that have been published. Uschold & Grüniger 1996 proposed four main phases when developing ontologies, which are: (1) identifying a purpose and scope, (2) building the ontology: this includes three sub-phases, which are: (a) ontology capturing, (b) ontology coding, (c) integrating existing ontologies. The later two phases are (3) evaluation and (4) guidelines for each phase. Furthermore, Sugumaran & Storey 2002 provided a heuristics-based ontology creation methodology to create a domain ontology.

For our research, we follow the proposed principles to create the needed ontology on a manual basis. We create ontology in order to allow for real semantic interpretation for HII to complement the shortcoming of just using XML in the task of information integration. Besides defining the terms and relationships for the domain in which we perform HII on the ontology, the semantic heterogeneity should also be considered when creating the needed ontology. We recognize the reasons for the semantic heterogeneity problems that the ontology in our research structure wants to handle. Visser et al. (2003) have also categorized the reasons for semantic heterogeneity. We list and explain the reasons for semantic heterogeneities in Table 3-4.

Table 3-4: Causes for Semantic Heterogeneity

Causes	Explanations
Conflicts with Scale and Currency	Two attributes that semantically similar might be represented using different units and measures.
Representation Conflicts	Two attributes are semantically similar, but they might be represented in different formats, for example, school grade: {1, 2, 3, 4, 5} vs. {A, B, C, D, E}
Subjective Mapping Conflicts	The subjective of two attributes is the same, but they are represented in their own styles. For example, German grades: {15, 14, …, 0} vs. American grades: {A, B, C, D, E}
Subsumption Conflicts	The content of an attribute is subsumed by the other one. For example, “hotels” includes “congress-hotels”, but the latter, with smaller scope of concept, is only part of the former.
Overlapping Conflicts	Parts of the content of two attributes are the same, but they are not equal to each other. For example, hostels and hotels vs. hotels and camp-sites.
Incompatibilities	The concepts of two attributes are the same, but actual meanings of them are still a little different. For example, hostels and hotels all mean the places for accommodation when traveling, but hostels are cheaper, some are only for youth. In contrast, hotels are more expensive.
Aggregation Conflicts	The concept of two attribute are different, but the

	concept of one of them is the aggregative concept of the other one. For example, hotel company vs. hotel. Hotel company means a company that operates hotels, but hotel means the place for accommodation when traveling.

We take the above conflicts into consideration when performing the conceptual modeling for the underlying information sources for the creation of the needed ontology. Afterward, we must create a connection between the global schema and the created ontology for the use in our research structure in the following.

Because the ontology defines terms and relationships with axioms of a domain, we view the elements in the global schema as the instance of the resources defined in the ontology. In other words, we use the ontology to define the relationships between the elements in the global schema. For instance,

```
<owl:Class rdf:ID=”Faculty”>
   <rdfs:subClassOf>
      <owl:Class rdf:about=”#Employee”>
   </rdfs:subClassOf>
</owl:Class>
```

The fragment of the above ontology defines a resource “Faculty” and represents the relationship between resource “Faculty” and resource “Employee” which means faculty must be an employee. We use such definition to define the element “Faculty” in the global schema as follow:

```
<Faculty rdf:ID=”schema_Faculty”>
   …
</Faculty>
```

Under such kind of connection between the two different data model, semantics defined in the ontology itself can be appended to the elements in the global schema which is represented in XML format. We provide another example that the semantics only can be caught by ontology to address the importance of the combination:

```
<owl:Class rdf:ID=”TeachingAssistant”>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty>
            <owl:ObjectProperty rdf:ID=”mustbe”/>
         </owl:onProperty>
         <owl:allValuesFrom rdf:resource=”#GraduateStudent”/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf rdf:resource="#Assistant" />
</owl:Class>
```

We can hardly describe the relationship that a teaching assistant “must be” a graduate student in XML data model. However, such kind of relationship is common in the real world. To complement such a shortcoming of XML, we catch the relationship and define it in the ontology. And then we connect the two data model by defining the element about teaching assistant in the global schema as the instance of the resource in the ontology like:

```
<TeachingAssistant rdf:ID="schema_TeachingAssistant">
  …
 </TeachingAssistant>
```

Afterwards, we can obtain the knowledge that a teahing assistant must be a graduate student by inferencing against the ontology. Obtaining more knowledge about the real world can help enhance the accuracy and precision of the interoperation between the underlying heterogeneous information sources.

Although an XML data model could represent certain kinds of semantics, for instance inheritance, there are still some relationships that cannot be represented by the XML data model, for instance intersection, union and so on. We use ontology to define the complete relationships of the domain and then use it to define the relationships between the elements in the global schema by viewing the global schema as an instance of the ontology.

Only while creating the connection between global schema and ontology, we can enable reasoning over the ontology to assist the query against the global schema in the research structure and reach the interoperability of structure and semantics.

3.3.3. Mapping Global Schema to Local Data Sources

After we create the integrated schema, we still have to consider the mapping between global schema and the local data source schema. Since we view the integration structure as an independent system from the local data source, we must build some bridges between the schemas of the integration system and those local data sources. The mapping is the bridge of the global schema and the local data source schema. However, there have been several proposed approaches to specify the mapping between global schema and local schema, which are global-as-view (GAV), local-as-view (LAV) (Levy, 2000; Manolescu, Florescu, & Kossmann, 2001). The first approach is to define the global schema as a view over the local schemas. In contrast, the second approach is to define the local sources as views over the global schema. The fundamental comparison between these three approaches is presented in Table 3-5.

Table 3-5: Comparison between GAV and LAV

	GAV	**LAV**
Query Reformulation	Translating the query on the global schema into queries on the local schemas is a simple process of view unfolding.	The query on the global schema needs to be reformulated in the terms of the local data sources' schemas; this process is traditionally known as "rewriting queries using views" and is a known hard problem.
Data Modification	To handle modifications in the local data sources set or in their schemas, the new global schema needs to be redesigned considering the whole modified set of sources.	A local change to a data source can be handled locally, by adding, removing or updating only the view definitions concerning this source.
Data Format	If the local data sources do not have the same data format (e.g. some are relational while others are XML), it would be difficult to define the global schema as a view over sources in different formats.	Each source can be described in isolation, by a view definition mechanism appropriate to its format.

We adopt the GAV approach to specify the mapping between global schema and local data source schema because its query reformulation process is easier than the LAV approaches. Although the evolution process of GAV approach is harder than LAV approach, the query reformulation process is our prior consideration of adopting which approach for specifying the mapping. We also apply this approach to the unstructured information sources to specify the mapping between the global schema and the index created in the previous section. In Figure 3-9, we show a fragment of the mapping by continuing the previous example.

```
G.Person :- S2.Person
G.Person.name :- S2.Person.name
G.Person.name.fname :- S2.Person.name.fname
G.Person.name.mname :- S2.Person.name.mname
G.Person.name.lname :- S2.Person.name.lname
G.Person.ssn :- S2.Person.ssn
G.Person.birthdate :- S2.Person.birthdate
G.Person.sex :- S2.Person.sex
G.Person.address :- S2.Person.address
G.Person.address.no :- S2.Person.address.no
G.Person.address.street :- S2.Person.address.street
G.Person.address.aptno :- S2.Person.address.aptno
G.Person.address.city :- S2.Person.address.city
G.Person.address.state :- S2.Person.address.state
G.Person.address.zip :- S2.Person.address.zip
G.Faculty :- S1.Teacher, S2.Faculty
G.Faculty.name :- S1.Teacher(name), S2.Faculty.name
G.Faculty.name.fname :- S2.Faculty.name.fname
G.Faculty.name.mname :- S2.Faculty.name.mname
G.Faculty.name.lname :- S2.Faculty.name.lname
G.Faculty.ssn :- S2.Faculty.ssn
G.Faculty.birthdate :- S2.Faculty.birthdate
G.Faculty.sex :- S2.Faculty.sex
G.Faculty.address :- S2.Faculty.address
G.Faculty.address.no :- S2.Faculty.address.no
G.Faculty.address.street :- S2.Faculty.address.street
G.Faculty.address.aptno :- S2.Faculty.address.aptno
G.Faculty.address.city :- S2.Faculty.address.city
G.Faculty.address.state :- S2.Faculty.address.state
G.Faculty.address.zip :- S2.Faculty.address.zip
G.Faculty.teacher_id :- S1.Teacher(teacher_id)
G.Faculty.dept_id :- S1.Teacher(dept_id)
G.Faculty.rank :- S1.Teacher(rank), S2.Faculty.rank
G.Faculty.salary :- S2.Faculty.salary
G.Faculty.office :- S1.Teacher(office), S2.Faculty.office
G.Faculty.phone :- S1.Teacher(phone), S2.Faculty.phone
```

```
G.Faculty.email :- S1.Teacher(email)
…
```

Figure 3-9 A Fragment of the Example of the Mapping between Global Schema and Source Schema

Creating the mapping between global schema and local source schema gains the interoperability of different systems. That is also the goal of this research. In the following, we describe how to apply the integrated model in our research structure.

3.4. Query Resolution in Research Structure

In this section, we describe the course of query processing we designed for our research structure.

In the beginning, the query interface was designed according to XQuery because we used XML for the common data model of our research structure. As a result, users have to formulate their query request in an XQuery form against the integrated schema. Figure 3-10 shows the steps of query processing in the research structure. We explain every step in more details as follow:

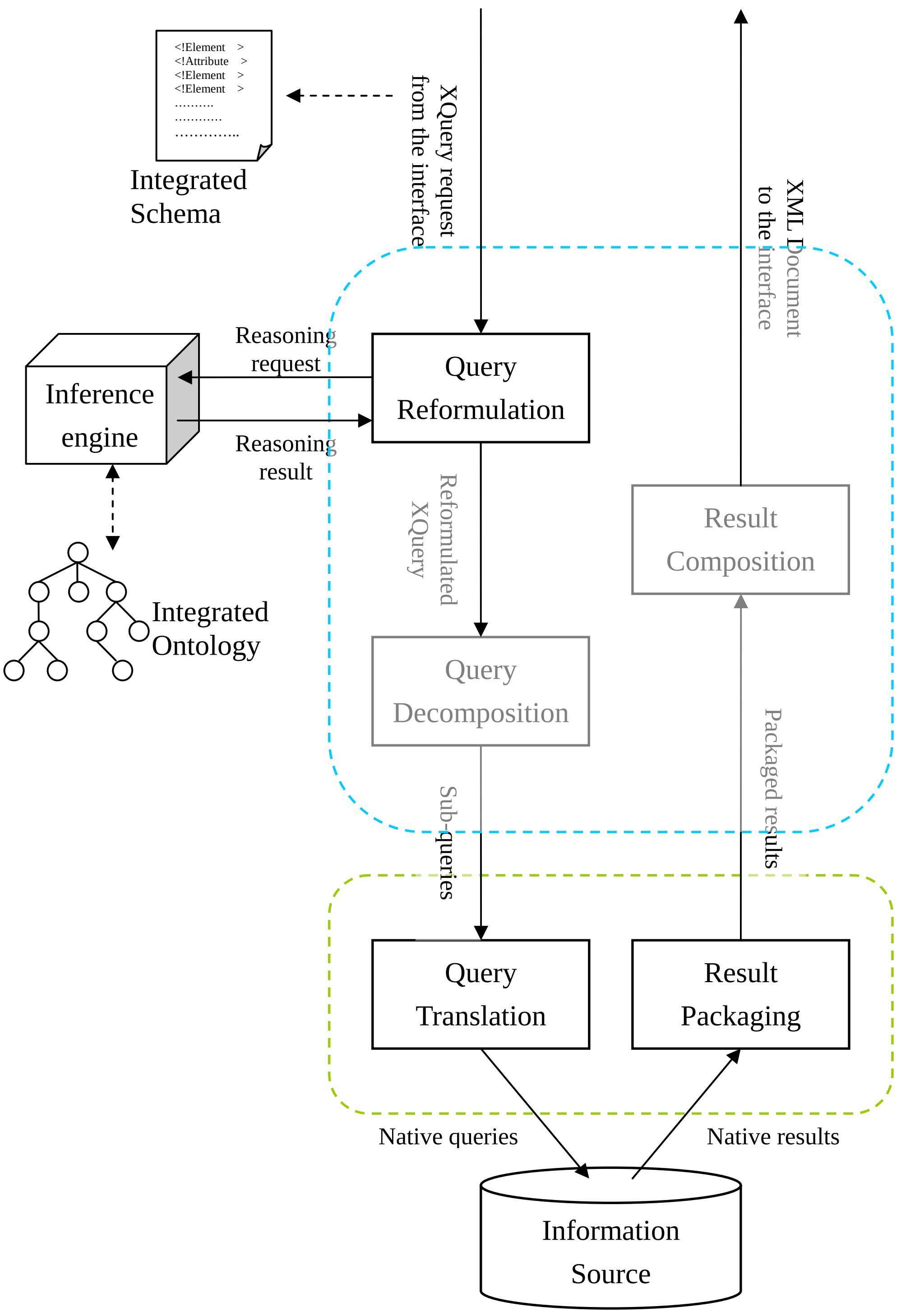


Figure 3-10: Query Processing in Research Structure

1. Query Reformulation:

After a user issues an XQuery request from the query interface against the unified view we provide, this query request would be sent into the mediator, where the query reformulation component in the mediator would receive it. The query reformulation component would pass a reasoning request according to the original XQuery request issued by the user to the inference engine and then inference engine would access the ontology in order to find out the relationships that are implicit in the user's original XQuery request.

Since we want to find out the implicit relationships in the user original XQuery request, we have to take the user query apart. We identify and extract entities in the user query for issuing the reasoning request to the ontology. According to the reasoning request, inference engine can have the reasoning results.

The reasoning results are sent back to the query reformulation component. Hence, the query reformulation component could reformulate the original XQuery request on the basis of the reasoning results. For example, the user might not discover the implicit relationships between the entities specified in the global schema because the global schema just gives the user the sketch of the structure of the underlying information sources. Therefore, the reasoning results can be used to complement the path expression that formulate by the user original that can clear the relationships specified in the user query. Besides, the reasoning results may help to find out much more and related answers of the query by adding new query expression.

Afterward, the query reformulation component would send the reformulated query to the query decomposition component.

2. Query Decomposition:

After the query decomposition component received the reformulated query, the reformulated query would be decomposed with the assistance of the mapping between integrated schema and local source schema. So query decomposition component could use such information to decompose the query into several sub-queries that are respectively applicable to their target information sources. Afterwards, query decomposition component would send these sub-queries to the corresponding wrappers.

3. Query Translation:

The wrapper would then receive the corresponding sub-queries. However, due to the limited capability of the underlying information source, the wrapper would have to translate the generic sub-queries (used in this framework) into native queries (e.g. SQL) according to the capability. Afterwards, such native queries would be issued to the corresponding source in order to find out the data really needed.

To enable query translation, we have to identify the correspondences between XQuery expression used in our research structure and the local source query expression. In this research, we also use the representative heterogeneous information sources that are RDBMS and ODBMS as the explanation. We identify the correspondences between the different query languages and list them in Table 3-5 and Table 3-6.

Table 3-6: The Correspondences between XQuery Expression and SQL Expression

XQuery Expression	SQL Expression
For	No Corresponding function
Let	
Where	Where
Order by	Order by
Return	Select

Aggregation function	Aggregation function
Comparison	Comparison
Path expression	No Corresponding function
included in XPath expression	From
No Corresponding function	Group by
	Having

Table 3-7: The Correspondences between XQuery Expression and OQL Expression

XQuery Expression	**OQL Expression**
For	No Corresponding function
Let	
Where	Where
Order by	Order by
Return	Select
Aggregation function	Aggregation function
Comparison	Comparison
Path expression	Path expression
included in XPath expression	From
No Corresponding function	Group by
	Having

According to the list of correspondences between the two different query languages, we can find out some kinds of the query expression cannot be completely mapped to another one. However, we just treat the exception of the mapping as a special case and markup it for the further process that might be the requirement for writing additional rules at the execution time.

4. Result Packaging:

After the information source processes those native queries, it would send the native results (e.g. record set) back to the wrapper. Because it is one-on-one between a wrapper and a type of information source, this makes sure that the results would be sent back to the corresponding wrapper. After receiving the results, the wrapper would package the results into a normal form and send it back to the mediator.

5. Result Composition:

It is the query composition component in the mediator that would collect several packaged results sent back from several wrappers according to the previous decomposed result. And with the aids of the integrated schema, the individual results could be composed in a complete XML document. Finally, it would send the XML document to the interface for the user.

To construct an integration system, we must consider execution rate, completeness of the query result, and consistency of all the underlying data. And the optimization issue must be taken into consideration while performing the query process. However, because the optimization issue is out of our scope, we won’t discuss it here in this research.