Code generator for integrating warehouse XML data sources.

Chunsheng. Liu
University of Windsor

Code generator for Integrating Warehouse XML Data Sources

BY

Chunsheng Liu

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in
Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2001

0-612-67632-3

952190

Abstract

XML – the extensible Markup Language, has been recognized as the standard for data representation and exchange on the world wide web. Vast amounts of XML data are available on the web. Since the information on the web is stored on separate web pages, it is very hard to combine pieces of information for decision support purposes. Data warehouse data integration provides a solution for integrating the different XML source data into a unique format with meaningful information for decision support systems.

A data warehouse is a large integrated database organized around major subjects of an enterprise for the purpose of decision support querying. Many enterprises are creating their own data warehouse systems from scratch in different varying formats, making the issue of building a more efficient, more reliable, cost-effective and easy-to-use data warehouse system important. Building a code generator for creating a program that automatically integrates XML data sources into a target data warehouse is one solution.

There is little research showing the use of the newest XML techniques in code generator for data warehouse XML data integration. This thesis proposes a Warehouse Integrator code generator for XML (WIG4X), which integrates XML data sources into a target data warehouse by first generating Java programs for data extracting, cleaning and loading XML data into the data warehouse. WIG4X system also generates the programs for creating XML views from the data warehouse. XML schema mapping strategy is employed for structural integration of each XML data source to data warehouse using a first order logic-like-language similar to that used in INFOMASTER. The content integration is handled through XML data extraction, conversion constraints, data cleaning and data loading.

Keywords: XML, data warehouse, code generator, XML Schema, SAX, DOM, Java, extraction, integration

To my parents.

To my wife,

To my teachers,

To my friends

...

Acknowledgments

I take this opportunity to thank my advisor Dr. Christie Ezeife. This work could not have been accomplished without her solid help in theory and practice. Thanks for her precious consulting time during this work, her continuous encouragement and intensive support throughout my graduate studies. Thanks to Dr. Froduald Kabaza and Dr. Myron Hlynka for comments, questions, and criticisms on this thesis. Thanks to Dr. Li for chairing my thesis defense.

I would like to thank my friends for their support, advice and help during my entire M.Sc. study.

Table of Content

List of Figures

Chapter 1: Introduction

Vast amounts of information are available on the Internet and XML is fast emerging as a dominant standard for representing data on the world wide web. Since information on the web is stored on separate web pages, it is very hard to combine pieces of information for decision support purposes. Data warehouse data integration provides a solution for integrating the different source data into a unique format with meaningful information for decision support system. Companies or organizations would benefit from a data warehouse, which stores huge information for easy access and quick business decisions [Ez01].

Building a data warehouse system from the XML data sources has some challenges including (1) XML data are distributed across a number of web pages and they can not be obtained from a single information source, (2) XML data are semi-structured and a mapping is needed to map semi-structured data to relational format before data extraction, (3) there might be semantic mismatches between different web sources whereby the same concept might be represented by different words, or the same word might refer to different concepts, (4) XML data change frequently, because new information sources appear while others disappear and existing information sources may change the format of their data or their content.

The conventional approach for building a data warehouse system often needs designing, coding, debugging and adapting many programs from scratch. Adding a new data source or incorporating new functions in the system needs re-designing and re-coding, which is tedious, costly, time and labour consuming. The code generator approach can generate standard data warehouse integrator programs automatically to reduce huge coding and transformation work. Generally, these kinds of automatic code generator are based on well known approaches [BS97], and ensure consistency between user design and implementation stages [ELS01]. So, building a warehouse integrator code generator is a good solution for integrating heterogeneous XML data into data warehouse systems to benefit from automatic correct and consistent code.

A data warehouse is a single, integrated store of data that provides stable, point-in-time data for decision support applications. In most cases, for a large dataset, executing queries with aggregations against the detailed transaction records is prohibitively expensive, simply because of the volume of records that are being accessed. As a result, data warehouses provide some form of pre-aggregation in order to support complex data-intensive queries.

A ***view*** is a derived relation defined in terms of base relations. It defines a function from a set of base tables to a derived table. Traditionally, views in database are virtual views. Queries over virtual views are answered by accessing the base data every time the views are referenced. In order to provide faster accesses to data warehouse in response to queries, a view can be materialized by storing its content in the data warehouse [XE00]. A materialized view is a stored view derived from base relations. In relational databases materialized derived relations (views) have long been proposed to speed up query processing. In the data warehouse, these views store redundant, aggregated information and are commonly referred to as summary tables. As a result, querying the view instead of the base relations offers several orders of magnitude faster query speeds. In our approach, some kinds of views are made in XML format for easier transfer and display of query results.

This thesis proposes WIG4X, a warehouse integrator code generator system for XML data sources. This system (1) accepts as its input XML data source descriptions, data warehouse schema description and some constraints for the mapping between them. Then, (2) the system generates data warehouse integrator programs in a target language Java. (3) The generated programs can extract, clean and load the data from the XML documents into a relational data warehouse system, and the generated programs can also create some useful views in XML format from the created data warehouse system.

1.1 Data Warehouse

In recent years, the data warehouse has become an active research area in the database community. A data warehouse is a single, complete, and consistent store of data obtained from a variety of data sources and made available to the end users in a way that they can understand and use in a business context [De97].

Generally, a data warehouse system consists of a fact table and dimension tables. The fact table is the table that stores the integrated data with some aggregate measures of interest. Attributes of the fact table are foreign keys representing subjects of interest, the integration attribute(s), the attribute representing historical (usually time) and the measure aggregate attributes. Dimension tables store detailed information related to foreign key attributes of the fact table. Thus, each foreign key attribute of the fact table refers to a primary key attribute in a dimension table [Ez01, XE00]. For example, consider a shopping system which has two branches: Canadian branch and US branch. Each branch has its own database system which stores the customers' order information over years. The source data table schemata are shown in Figure 1.1:

Canadian branch:

CAOrder (orderID, ItemID, CustID, OrderDate, Quantity, CAPrice)

CAItem (ItemID, name, brand, type)

CACust (CustID, name, street, city, province, postcode, phone, gender)

US branch:

USOrder (orderNum, ItemNum, CustKey, OrderDate, Quantity, USPrice)

USItem (ItemNum, name, brand, type)

USCust (CustKey, fname, lname, address, phone, gender)

Figure 1.1: Source table schemata of a shopping system

In Figure 1.1, the customers' shopping order information of two branches of one shopping company are stored in the source tables of the two branches. Integrating data from these source tables, we can get a warehouse system which stores the order information of all branches of this shopping system over a period of several years. Figure 1.2 is a sample data warehouse in star schema for this shopping system:

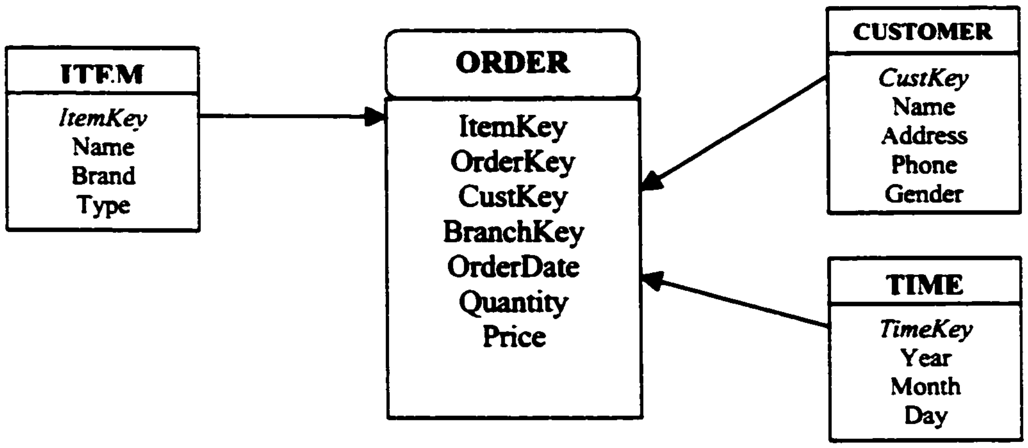


Figure 1.2 Warehouse tables of a shopping system

In Figure 1.2, the central fact table is the ORDER table, and its attribute "BranchKey" is the integration attribute which represents the different branches, its attribute "OrderDate" represents historical information, the measure aggregate attributes of interest are "Quantity" and "Price" for representing the quantity and price of store items ordered by customers respectively. Dimensions of the data are captured through the ITEM, CUSTOMER and the TIME tables. The fact table is associated with dimension tables through foreign-key reference to the three dimension tables.

A materialized view with this example is a query to find the total quantity of all items ordered by each customer at each branch every month. This view contains the attributes (CustKey, Name, ItemKey, OrderDate, MonthlyQuantity).

A Data Warehouse is a subject oriented, integrated, nonvolatile, time collection of data in support of management's decision [BS97].

Subject oriented: A data warehouse is oriented around the major subject areas of the enterprise [EZ01], e.g., item and customer might be two subjects of interest in a shopping system.

Integrated: The data warehouse data are integrated from different data sources [XE00]. For example, in the fact table of the data warehouse system, some attributes are measure aggregates, e.g. Quantity in the ORDER table in Figure 1.2, some attributes are integration attributes, e.g., BranchKey in the ORDER table in Figure 1.2.

Nonvolatile: A data warehouse stores historical data. When new data are loaded into data warehouse, the old data do not have to be updated. e.g., the shopping data warehouse may store information for 18 years and the new information will be moved into the data warehouse to refresh it without the need to change data already in the warehouse.

Time collection: A data warehouse is designed to store historical data over long periods of time [BZ96]. The structure of the data warehouse always contains some elements of time to answer questions like "How much is the total sale record of Canadian Tires in Ontario in the last 10 years?".

A relational data warehouse, unlike a traditional relational database system, is not well normalized and it is large. It is used to store business information for many years, and the data warehouse data need to be aggregated for business decision making.

1.1.1 Architecture of a Data Warehouse Integration

A data warehouse system consists of source data, data integration and transformation tools, data warehouse and metadata as well as front end (OLAP, DSS etc) applications for querying the data warehouse [Ez01]. A typical data warehouse architecture is shown in Figure 1.3:

Data Sources: are the data that are integrated into the data warehouse. These data can be in different formats, structures, names and measurements, e.g., data sources can be in relational database tables, in html file, in XML documents or even in text files. In Figure 1.3, data source refers to data in level 1.

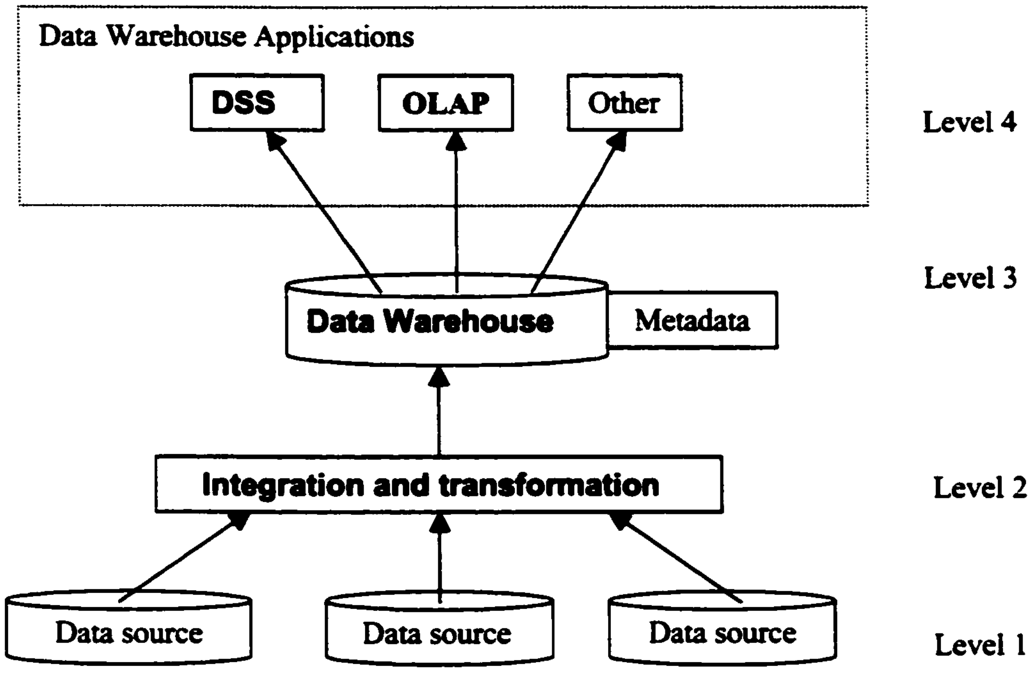


Figure 1.3: Architectures of Data Warehouse

Data integration and transformation: represent a set of programs that integrates the different formats of source data into the unique format of warehouse data, get aggregated values, clean redundant data and load these data into the data warehouse. The data warehouse is the data store for the data integrated from different source data, e.g., in the two branches of a shop, the customer gender may be represented in different format: one branch represents gender as "M" and "F" and another branch represents it as "0" and "1". During integration of data from these two branches, the customer gender information in the data warehouse is represented with a uniform format like "Male" and "Female". The data integration tool is shown in level 2 of Figure 1.3.

Metadata: is "data about data" [BZ96]. Metadata describes the data and the environment for both source data and the data warehouse, as well as the relationships between the source database and the data warehouse including the transformational rules. That is shown as part of level 3 in Figure 1.3.

Applications: in Figure 1.3 are sets of programs that are for data warehouse data mining and querying suitable for business decision making. That part refers to level 4 in Figure 1.3.

1.1.2 Warehouse Data

From the ***role of data*** in the data warehouse system, there are three classes of data - internal data, external data and metadata.

Internal data are generated by the operational transaction system in the DBMS. For example, a shopping system accepts the transaction information from an automatic sale machine of this shop (information from inside of the shopping system).

External data are captured outside of the enterprise and are, most often, made available at a cost by a specialist information provider [Ke98], e.g., the stock index of a finance system is captured from the stock exchange center (information from outside of the finance system).

Metadata are "data about data" [BZ96]. They describe the data and the environment in which the data are managed, the contents, the location of data in a data warehouse as well as the relationship between the source database and the data warehouse.

From the ***type of data***, data sources can be divided into three types:

- ***Structured data sources***: such as a relational database or an object-oriented database. Data are stored in a predefined strict format.
- ***Semi-structured data sources***: for semi-structured data, the information that is normally associated with a schema is contained within the data, and is sometimes called "Self-describing" [Bu97]. Semi-structured data include HTML and XML.
- ***Unstructured data***: Unstructured data are the data with no apparent organization, such as data in free text file.

1.1.3 Warehouse Data Integration and Transformation

Data transformation: converts the input data into the desired format, the input data in WIG4X are XML data. Over the years, different database designers have made numerous individual decisions about how a source database should be built. Data transformation should deal with differences such as in encoding, different measurements of attributes, different attribute structures, different data type characteristics, different naming conventions, different data redundancy. Whatever format source data come from, a data transformation process should keep them in a uniform state. For example, in two branches of a shop, the customer gender represented in different formats, as "M" and "F" and as "0" and "1", are transformed into customer gender information of data warehouse with the uniform format "Male" and "Female" as follows:

Source1	Source2	Transformation	Warehouse
"M"	"0"	–>>	"Male"
"F"	"1"	–>>	"Female"

Data extraction: Source data are distributed accross different kinds of sources and having different formats. They have to be moved to the data warehouse, which implies that a data extraction system has to be employed.
Data cleaning: Since data are extracted from different sources, there is a possibility that some data are stored in multiple sources. There is also a possibility that different data belonging to the same key might be present at different source databases under different identification. Hence, to keep data correct, there is need to clean the data to retain a single warehouse identity for this entity. Data cleaning can be done before data extraction, during data conversion process, after data extraction process.
Data Loading: After data extraction and cleaning, data from the source database are now loaded into the data warehouse [GW98] by (1) *Loading changed data*: moving just the changed item in order to reduce the amount of data that has to be moved. This includes both newly obtained transaction information and the changes from the source system, (2) *Creating aggregate records*: Aggregate records are summaries that can be obtained by sorting on one or more attributes, (3) *Indexing newly loaded records* for speeding query response time [GW98].

Data integration approaches include the structured data integration approach, semantic data integration approach and the virtual data integration approach [DG97].
The Structured data integration approach is characterized by a self-describing model whereby each data item has an associated descriptive label e.g., <label value>. With this approach, a client can discover the structure of the information as queries are posed. Generality and conciseness of a self-describing model makes the "structured approach" a

good candidate for the integration of widely heterogeneous and semi-structured information sources. *The semantic data integration approach* has the requirement that for each source, metadata and conceptual schema must be available. A common data model as the basis for describing sharable information must be available. Semantic information encoded into a metadata schema permits one to efficiently extract information. *The virtual Data Integration approach* is based on transformational rules that allow one-to-one mapping of source to target relations and attributes.

1.2 Code Generator for Data Warehouse Data Integration

A ***compiler*** is a software unit that translates a program from an input program written in a language to machine code, so that it can be executed directly on the computer. A compiler is used for translating high level programming languages such as Java to low level computer languages such as executable codes. The language that the compiler translates is called the source language while the executable language is the target language [Liu01]. The ***code generator for data warehouse*** creates code from data sources input rather than program input, and the generated code is always a high-level language program that can access the database.

Warehouse Code generators create a data transformation program based on source and target data definitions, and on data transformation and enhancement rules defined by the developer. Code generation products employ data manipulation language statements to capture a subset of the data from the source system. In most products, user definition can be used for performing additional processing, transformation and enhancement. Code generation products are used for data conversion projects, and for building an enterprise-wide data warehouse, when there is a significant amount of data transformation to be done involving a variety of flat files, and nonrelational, and relational data sources.

1.2.1 Why Use a Code Generator?

The traditional approach to data warehouse integration always designs a unique system for each application system, and the designing process is expensive and time consuming. In fact, many tasks are done repeatedly in a similar manner. When the data warehouse of the application system changes, the re-designing process needs a lot of re-coding work, and is always very prone to errors.

A code generator will generate a program that integrates data from the source system to a target system based on each group of related data sources. This approach reduces the need for an organization to write its own data capture, transformation and load programs [Liu01]. Automatic code generation is a way to ensure consistency from design to the implementation stages in the software development process.

1.2.2 Data Warehouse Code Generator Architecture

Data warehouse code generator is the code generator that takes source data, source metadata, data integration rules (assume based on the virtual data integration approach) and target program templates (assume based on the template driven code generator approach) as input. The output of the project is a set of target programs (in this case in Java) for data warehouse data integration, which includes [ELLS02]:

(1) Warehouse creating program that creates target warehouse and ODS tables

(2) Data extracting program that extracts data from different sources.

(3) Data cleaning program that cleans non-useful data

(4) Data loading program that loads data into a data warehouse

(5) XML view creating program that creates XML views from a data warehouse

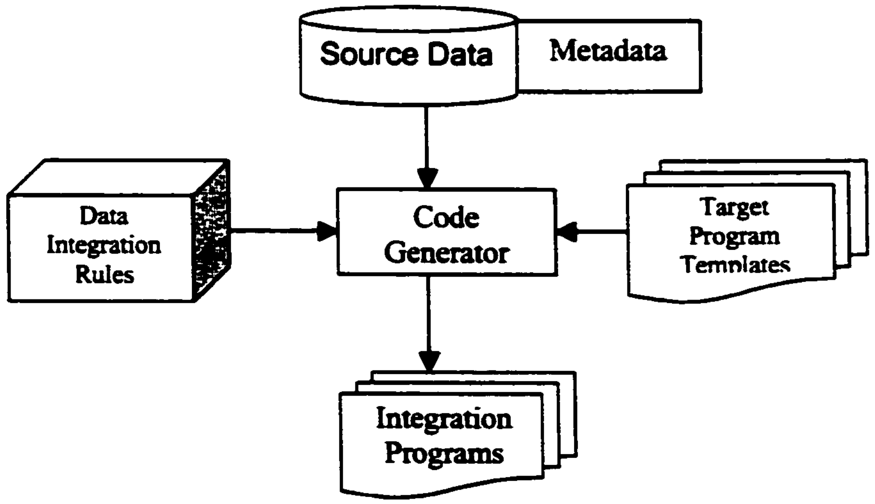


Figure 1.4: Data Warehouse Code Generator Architecture

1.2.3 Code Generation Approaches

There are many kinds of code generators used in different areas. Generally, there are two important approaches: Rule based code generator and Stack based code generator.

A rule based code generator is a code generator mainly used for transferring an older programming language to a new generation programming language. The main key features of rule based code generator are generating complete code, easy and complete integration of source systems, adaptable to any architecture or source code and generation of multiple languages from a single specification. A rule based code generator enables organizations to incorporate their knowledge and best practices in the generation rules, allowing automatic implementation of the models within the desired architectures. Any specification can be implemented in different target environments without changes, and it is architecture independent.

A stack based code generator generates code from source language to target language using stack variable memory. The basic algorithm for stack based code generator includes raw input process, code clean up, stack scheduling, Peephole optimization, Code generation [Ko92].

1.3 XML

XML, extensible Markup Language, is a new markup language which was developed by the W3C (World Wide Web Consortium). The W3C is the organization in charge of the development and maintenance of most web standards. XML is fast emerging as the dominant standard for representing data in the world wide web today [FLM98].

userID	Name	Address			Telephone
		street	province	Postcode	
1001	Jeffrey Liu	564 Randolph Ave.	ON.	N9B2T6	519-9775641
1002	Linda Jones	479 Rankin Ave.	ON.	N9B2F6	519-5603738

Figure 1.5: An address book information

An example of XML for representing the address book information in Figure 1.5 is given in Figure 1.6:

```
<?xml version="1.0"?>
<address-book>
   <entry userID=1001>
         <name>Jeffrey Liu</name>
         <address><street> 564 Randolph Avenue</street>
                  <province>ON</province>
                  <postcode>N9B2T6</postcode>
          </address>
          <tel>519-9775641</tel>
    </entry>
    <entry userID=1002>
         <name>Linda Jones</name>
         <address><street> 479 Rankin Ave.</street>
                  <province>ON</province>
                  <postcode>N9B2F6</postcode>
          </address>
          <tel>519-5603738</tel>
    </entry>
 </address-book>
```

Figure 1.6: An XML example for representing an address book information

In Figure 1.6, the content of the address book is stored within corresponding XML tags, each XML tag is closed. For example, the name "Jeffrey Liu" is stored in tag pair <name> </name> which is also called "element", and the <name> tag has the corresponding close tag </name>. For certain tags, there are attributes within them, for example, "userID" is the attribute within tag <entry>.

All tags and attributes are defined by users themselves. Tags can be given as any names and thus makes XML is extensible and flexible.

1.3.1 Advantages of XML?

Both XML and HTML belong to markup languages. Markup are the codes, embedded within the document text, which store the information required for electronic processing [DFS99]. HTML is the most popular markup language today, but HTML has some shortcomings. HTML has grown into a complex language, with more than 100 tags. Generally, these severely limit HTML in several important respects, among which are [Bo97]:

Extensibility: HTML does not allow users to specify their own tags or attributes in order to parameterize or otherwise semantically qualify their data.

Structure: HTML does not support the specification of deep structures needed to represent database schemas.

Validation: HTML does not support the kind of language specification that allows consuming applications to check data for structural validity on importation.

XML was developed to address these shortcomings. It incorporates many successful features of HTML, and it breaks new grounds where it is appropriate. XML differs from HTML in three major respects [Bo97]:

1. Information providers can define new tag and attribute names at will.
2. Document structures can be nested to any level of complexity.
3. Any XML document can contain an optional description of its grammar for use by applications that need to perform structural validation.

Some of the areas where XML would work are [B099]:

1. Applications that require the web client to mediate between two or more heterogeneous databases. For example, between the DB2 and Oracle database, data can be transferred in XML format, which can be recognized by both sides.
2. Applications that attempt to distribute a significant proportion of the processing load from the web server to the web client. For example, if user inputs some data from the browser, then the data can be sent to web server in XML format.
3. Applications that require the web client to present different views of the same data to different users. For example, with data stored in one XML file, a web server can use a different XSLT (eXtensible Stylesheet Language Transforming) file to show the XML file in different views to different users.
4. Applications in which intelligent web agents attempt to tailor information discovery to the needs of individual users. For example, information stored in XML file can be extracted by the web agents as required.

1.3.2 XML Techniques

XML is a standard markup language that many vendors are willing to support, for example, Internet Explorer 5.0 and Netscape 6.0. In particular, the W3C has developed a number of standards that complement XML. These standards are often referred to as "XML companion standards." Many other organizations also develop some techniques to support XML.

XML Namespace:

XML Namespace often associates an owner with elements, this enables extensibility because it means an organization can add to existing elements and clearly label who is responsible for the extension. This prevents name conflicts and is the only way to enable reuse of standard structures.

DTD:

DTD stands for Document Type Definition. It is a mechanism for describing the XML document structure tree, which comprises a set of declarations defining the elements used where they may be applied in relation to each other. DTD is the metadata for XML data source. The DTD for the XML example in Figure 1.6 is shown in Figure 1.7:

```
<!ELEMENT address-book (entry+)>
<!ELEMENT entry (name, address, tel)>
<!ATTLIST userID CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (street, province, postcode)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT province (#PCDATA)>
<!ELEMENT postcode (#PCDATA)>
<!ELEMENT tel (#PCDATA)>
```

Figure 1.7 An example DTD for XML document

In Figure 1.7, the DTD describes the structure of XML file in Figure 1.6 as: the content of <address-book> tag is composed by at least one <entry>, the content of <entry> tag is composed by <name>, <address>, and <tel>, the content of <address> tag is composed by <street>, <province> and <postcode>.

More generally, declarations allow a document to communicate meta-information to the parser about its content. Meta-information includes the allowed sequence and nesting of tags, attribute values and their types and defaults, the names of external files that may be referenced and whether or not they contain XML, the formats of some external (non-XML) data that may be included, and entities that may be encountered.

There are four important declarations in DTD file:

1. <!ELEMENT>: Declares the elements' name and what they may contain.
2. <!ATTLIST>: Declares the attributes of the given element, and the attributes' constraints.
3. <!ENTITY>: Declares the entities which define the replacement text for use in XML documents.
4. <!Notations>: Declares the notations which associate external programs with certain types of content.

There are two main data types in DTD file: CDATA and PCDATA. CDATA stands for character data, PCDATA means parsed character data. The difference between CDATA and PCDATA is that PCDATA can not contain markup characters (markup character means character containing a markup such as < >).

For our approach, since we deal with the structure of the XML documents, so we only focus on the declarations <!ELEMENT> and <!ATTLIST>.

XML Schema:

An XML Schema is a formal expression of the structure of an XML document and of constraints on the text contained therein [Th00]. An XML schema is a separate file which comes from the original XML document. It defines and describes a class of XML documents by defining the markup constructs that appear in them, such as elements and their contents, attributes and their values, datatypes as well as notations. It affects the information content of the document (e.g., through default value). It describes the document's meaning and usage of markup language. Figure 1.8 shows the XML schema file for the XML document in Figure 1.5.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="address-book" type="address-bookType"/>

<xsd:complexType name="address-bookType">
<xsd:element name="entry" type="entryType"/>
</xsd:complexType>

<xsd:complexType name="entryType">
<xsd:sequence>
<xsd:element name="name"  type="xsd:string"/>
<xsd:element name="address"  type="addressType"/>
<xsd:element name="tel"  type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="userID"  type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="addressType">
<xsd:sequence>
<xsd:element name="street"  type="xsd:string"/>
<xsd:element name="province"  type="xsd:string"/>
<xsd:element name="postcode"  type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

Figure 1.8: An example XML schema file for XML document

In Figure 1.8, the format of the XML schema file conforms to XML syntax, each <xsd:element> has the data type. From the content of <xsd:complexType> tag, we can extract similar relations which are shown in Figure 1.5.

DTD is widely used in defining the structure of XML documents, but it has many limitations. It uses non-XML syntax, has no datatypes, is complex to extend, does not play well with namespaces, has no inheritance, it does not have enough checking and too much work is left to the application.

The XML Schema is similar to DTD, but it is a tremendous advancement over DTD. It has enhanced datatypes, written in the same syntax as XML instance documents, less syntax to remember, can extend or restrict a type (by xsd:restriction), can specify element content as being keys (by xsd:key), can define multiple elements with the same name but different content, can define substitutable elements.

Generally, there are some basic declarations in XML schema file:
<xsd:schema>: declares the namespace for XML schema file
<xsd:element>: declares the elements of the XML document, including their name and data type.
<xsd:complexType>: declares the elements with complex data type, which contains other sub-elements.
<xsd:sequence>: declares the sub-elements within a complex data type.
<xsd:restriction>: declares the restriction for given elements
<xsd:simpleType>: declares one element without sub-elements.
<xsd:attribute>: declares attributes for a given element.
<xsd:annotation>: declares the annotation of the XML schema file.

Since the XML schema is now in the developing stage, there are still a lot of new declarations, format and even some new namespaces (such as xs: or xsi:). For our approach, we focus on the basic format of an XML schema file.

The XML-Schema is an ongoing effort of W3C to aid and eventually replace DTD in the XML world. The XML-Schema aims to be more expressive than DTD and more usable by a wider variety of applications [LC00].

Parser:

An XML parser is a software component that sits between the software application and the XML files. Its goal is to parse the XML document shielding the developer from the intricacies of the XML syntax. A parser breaks data into smaller elements, according to a set of rules that describe its structure. Most data can be decomposed to some degree. A parser can use the grammar to parse the data source; that is, to break data elements into smaller ones such as <order ID= "1234567"> can be parsed to "<", "order", "ID", "=", "1234567" and ">", the data in the smaller elements can be processed by applications through the standard interface. The output of the parser is a parse tree. The parse tree expresses the hierarchical structure of the input data. A parser is the most basic yet most important XML tool. Every XML application is based on a parser. The XML parsers work as shown in figure 1.9:

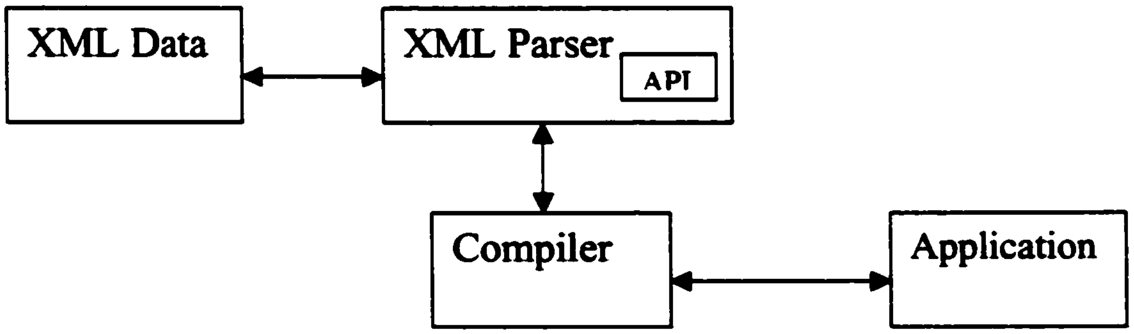


Figure 1.9: XML parser's working process

In Figure 1.9, XML document is parsed by parser and then the parsed content is processed by the standard interface. The standard interface refers to the Application Programming Interface (API), which contain many reusable classes, that programmers can include in their programs to access XML data through the corresponding parser. The APIs are developed by software vendors. The application refers to the programs which deal with XML data through the standard interface.

There are dozens of parsers freely available on the Internet. Among many XML parsers available today, the most popular ones are:

IBM: XML4J [XML4J]

SUN: JAXP [JAXP]

Microsoft: MSXML [MSXML]

For a Java program (application) to access XML data, an XML parser from where the XML API standard interfaces are obtained, should be stored on the machine with a Java compiler. For our approach, we use Java JDK1.3.1 and JAXP 1.1.

DOM and SAX Standards for XML API:

There are two major types of XML APIs (Application Programming Interface): the tree-based APIs like DOM [DOM00], and event-based APIs, like SAX [SAX00]. For programming languages such as Java or C++, there are no commands within their basic compilers to process the XML data. Thus, in order to access XML data, they use the XML APIs, such as DOM and SAX.

SAX (Simple API for XML) and DOM (Document Object Model) were both designed to allow programmers to access their information without having to write a parser in their programming language of choice. However, both of them take very different approaches to giving access to needed information.

DOM gives access to the information stored in your XML document as a hierarchical object model. DOM creates a tree of nodes (based on the structure and information in your XML document) and you can access your information by interacting with this tree of nodes.The textual information in your XML document gets turned into a bunch of tree nodes.

SAX chooses to give you access to the information in your XML document, not as a tree of nodes, but as a sequence of events! The SAX-based code uses much less memory because it buffers only one row of data at a time, while the DOM-based code buffers the entire document. The SAX-based code is faster because it does not have to spend time building a DOM tree. However, with SAX, it is hard to make random access and lateral movement, because the SAX only focuses on the element events and has no tree structure. For example, it is hard to find a parent node of a child node in SAX-based code.

So, when an application deals with a large XML document, or retrieves specific values from a document, SAX is the better choice. In other cases, such as modifying the document or randomly accessing the XML document, DOM provides an easy-to-use, clean interface to XML processing. Some XML parsers are built based on SAX standard while others are based on DOM standard. IBM XML4J and SUN JAXP support both SAX and DOM.

DOM is the World Wide Web consortium (W3C) recommendation, and its API can be found in the website www.w3c.org/dom [DOM00]. SAX is not a W3C recommendation. It is a public domain software, created by members of the XML-DEV mailing list. Its API standards can be found at www.megginson.com/sax [SAX00].

1.4 Thesis Problem and Contributions

XML is widely used across the world wide web and more data and applications are based on XML format. In order to manage such huge information for decision support, integrating XML data into a relational data warehouse is a very good approach. How to extract data from the XML documents efficiently, how to integrate the XML data and develop a data warehouse system for them successfully and effectively, are new and important topics. A code generator is a very useful tool for XML data integration, data warehouse creation, and maintenance because of its power in creating the program code for integrating different source data into the data warehouse. It can save a great deal of labour and development cost as well as increase the system reliability and flexibility.

This thesis aims at:

1. Finding efficient techniques to extract and integrate data from all kinds of XML documents, map XML document schema to a relational data warehouse schema, so that the WIG4X can employ these techniques in the code it generates.

2. Building the segment of the WIG4X (Warehouse Integrator code Generator for XML) system which accepts XML data sources and generates a program in Java for integrating them into relational data warehouse.
3. Extending WIG4X to create useful views in XML format from the data warehouse.

1.5 Outline of the Thesis

The organization of the rest of this document is as follows: Chapter 2 reviews previous work related to warehouse and XML data integration approaches as well as code generator approaches. Chapter 3 presents the proposed work WIG4X, introducing its architecture and the XML data integration techniques in WIG4X. In Chapter 5, the conclusion and future work are presented.

Chapter 2: Previous/Related Work

This chapter reviews earlier work on data warehouse integration, XML data integration and code generator systems. Section 2.1 briefly reviews previous projects on data warehouse and XML data integration. In section 2.2, we focus on reviewing the code generator approaches and related work on previous WIG (Warehouse Integrator code generator) system.

2.1 Data warehouse and XML data integration

Since XML provides a standard syntax for representing data, it is perceived to be a key enabling technology for exchange of information on the world wide web and within corporations. As a consequence, integration of XML data from multiple external sources is becoming a critical task.

There are three major approaches for data integration: structured data integration approach, semantic data integration approach and virtual data integration approach [CHS+94][AKH96][CGL+98][Ca95]. *The structured data integration* approach is based on a structured model and a set of queries written by a query language for integrating data. This approach is flexible and good for already known business data integration, but it is difficult to extend to all business in the world. *The semantic data integration* approach is based on a knowledge model to maintain the relations of data among data sources. The relations in knowledge model link each pair of data that have similar meanings, making it more extendable to integrate data sources which are not in relational database. *The Virtual approach* is a vertical data integration approach. This approach associates a relation in each data source with one table in the target DW. It is very extendable and flexible and thus good for the data warehouse data integration (DWDI) code generator.

Regarding the kind of database system (DBS) used for the integration of XML documents, there exists four different approaches.

(1) special-purpose DBS are particularly tailored to store, retrieve, and update XML documents. Examples are research prototypes such as Lore [GMW99], Strudel [FLM98] and Natix [KM00] as well as commercial systems such as eXcelon [OD99] and Tamino [SW00].

(2) Because of the rich data modeling capabilities of *object-oriented DBS,* they are well-suited for storing hyper-text documents. Object-oriented DBS and special-purpose DBS, however, are neither in wide-spread use nor mature enough to handle large scale data in an efficient way.

(3) Object-relational DBS would also be appropriate for mapping to and from XML documents since the nested structure of the object-relational model blends well with XML's nested document model [Bo99].

(4) The more promising alternative for storing XML documents are *relational database systems (RDBS).* Such an integration would provide several advantages: First, RDBMS products are mature and scale very well. Second, traditional data and semi-structured data can co-exist in a relational database, making it possible to develop applications that use both kinds of data with virtually no extra effort. Third, data processing in RDBMS is very fast.

Concerning the kind of XML data integration with RDBMS, there exists four basic alternatives.

(1) The most straightforward approach would be to store XML documents as a whole within a single database attribute. But actually existing systems do not handle it this way.

(2) Another possibility would be to interpret XML documents as graph structures and provide a relational schema allowing us to store arbitrary graph structures, as done by XcoP [SRL00] and Lore [GMW99].

(3) The third approach is Meta Data Modeling, which uses a framework to model not only the data models, but also model the mapping between the data models and the application schemas with a set of map metaconstructs employed. Sangam system [CR01] belongs to this approach.

(4) The fourth approach is the Schema integration, in which the structure of XML document (DTD or XML Schema) is mapped to a corresponding relational schema wherein XML documents are stored according to the mapping, such as X-Ray [KKR00], SilkRoute [FTS00]. Some of these approaches will be introduced in detail in the following sections.

2.1.1 Schema mapping integration approach: X-Ray

X-Ray [KKR00], is a generic approach for integrating XML with RDBS. The key idea is that mappings can be dynamically defined between DTDs and relational schemata thus coping with data model heterogeneity and schema heterogeneity. The integration fully preserves the autonomy of both the DTD and the relational schema, which in turn, ensures the continuity of already existing applications working with the XML documents or the RDBS.

This is made possible by introducing meta schema storing information about the DTD, the relational schema and the mapping knowledge itself. The meta schema is the key mechanism for X-Ray to map DTDs and relational schemata. It mediates between heterogeneous concepts and provides the basis for X-Ray to automatically compose XML documents out of the relational database when requested and decompose them when they have to be stored. The mapping knowledge is not hard-coded within an application but rather centrally stored within the meta schema, thus enhancing maintainability and changeability. The architecture of the Meta Schema [KKR00] is given as Figure 2.1.

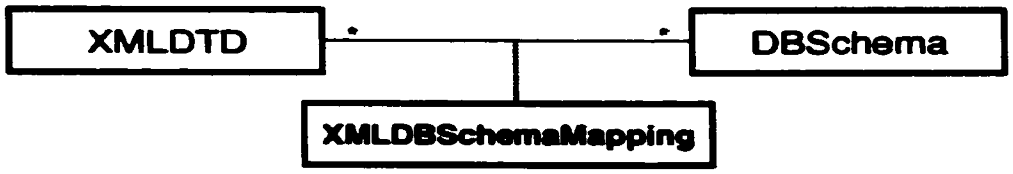


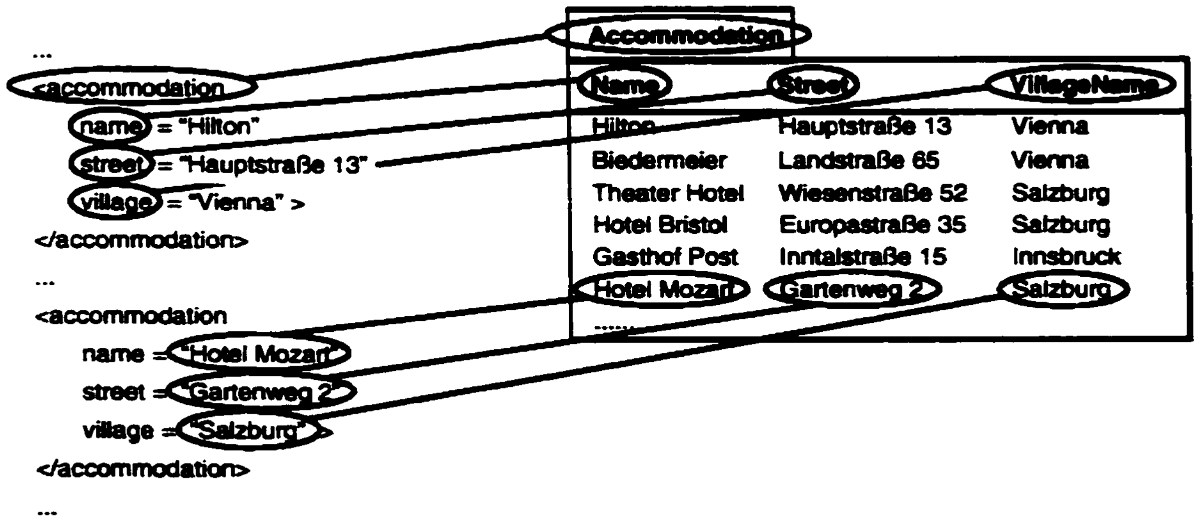
Figure 2.1: Architecture of the X-Ray Meta Schema

The meta schema consists of three components describing the meta knowledge:
The DBSchema component is responsible for storing information about relational schemata that shall be mapped to DTDs to make their data available to XML documents

or that shall be used to store XML documents. *XMLDTD* component stores schema information about XML documents as specified by means of DTDs. *XMLDBSchemaMapping* component stores the mapping knowledge between DBSchema and XMLDTD.

The goal of XMLDBSchemaMapping is to bridge both data model heterogeneity and schema heterogeneity in order to support a lossless mapping. This means that if an XML document is stored within the database, it should be possible to reconstruct it by retrieving the corresponding data out of the database and vice versa. It has to be emphasized that although the meta schema is designed on the basis of concepts provided by DTDs, X-Ray does not require the existence of an explicit DTD. However, there must be at least a common implicit structure of the XML documents, which can be used by an administrator as input for XMLDTD and XMLDBSchemaMapping.

The mapping rules between XML documents and Relational DBMS form the most important part of this approach. There are many different kinds of mapping rules, the most straightforward way is to map each XML element (such as "Accommodation" in Figure 2.2) type to a relation and each XML attribute (such as "name", "street" in Figure 2.2) to an attribute of the respective relation. Figure 2.2 is an example of a straightforward mapping:



```
...
<accommodation
   name = "Hilton"
   street = "Hauptstraße 13"
   village = "Vienna" >
</accommodation>
...
<accommodation
   name = "Hotel Mozart"
   street = "Gartenweg 2"
   village = "Salzburg" >
</accommodation>
...
```

Accommodation

Name	Street	VillageName
Hilton	Hauptstraße 13	Vienna
Biedermeier	Landstraße 65	Vienna
Theater Hotel	Wiesenstraße 52	Salzburg
Hotel Bristol	Europastraße 35	Salzburg
Gasthof Post	Inntalstraße 15	Innsbruck
Hotel Mozart	Gartenweg 2	Salzburg
......		

Figure 2.2: Straightforward Mapping of XML Concepts to Relational database

In the X-Ray approach, there are three basic kinds of mapping rules between XML DTD and the Relational database at the data model level as shown in Figure 2.3:

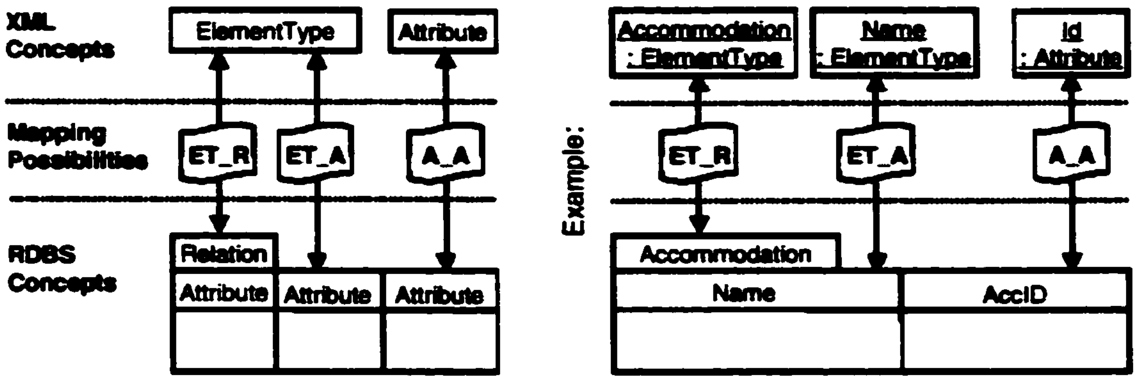


Figure 2.3: Three kinds of Mapping Rules

In Figure 2.3, three mapping rules [KKR00] are presented:

(1) ET_R: An element type (ET) is mapped to a relation (R), called *base relation*. Note that several element types can be mapped to one base relation. An example for an ET_R mapping is the element type accommodation in Figure 2.3.

(2) ET_A: An element type is mapped to a relational attribute (A), whereby the relation of the attribute represents the base relation of the element type. Note that several element types can be mapped to the attributes of one base relation.

(3) A_A: An XML attribute is mapped to a relational attribute whose relation represents the base relation of the XML attribute. Again, several XML attributes can be mapped to the attributes of one base relation. The XML attributes name, street, and village in Figure 2.3 give an example.

For the mapping rules, both element types and attributes can be mapped to a single base relation and a single attribute, only. Another point is that ET_A and A_A mappings determine also the instance level, in that database values are mapped to XML values. Thus, it makes sense that ET_R mappings occur together with ET_A and A_A mappings.

X-Ray is an approach for mapping between XML documents and relational database schemata, on the basis of the meta schema. XML documents may be automatically composed out of data stored within a relation DBMS and on the other side, decomposed into relational data without any loss of information, then it can be used for integrating XML documents to data warehouse, or vice versa.

2.1.2 Meta data modeling approach: SANGAM

Sangam [CR01], is a modeling transformation tool for data integration. "Sangam" is a Hindi word meaning Junction. It allows users to explicitly model mappings between different data models as well as re-structuring within one data model. It maps meta model based on a set of re-usable mapping constructs that can, in principle be applied on any data model described in a domain framework.

In Sangam, the operators for XML and relational model mappings were tested. Using the description of maps at the model level, mappings between specific application schemas and transformations of associated application data can be automated by their framework. Their framework guarantees the correctness of the map, of the generated transformation code, of the output data model, and of the generated application schemas, based on the correctness criteria for the map meta model. With Sangam, they show not only the feasibility of their approach but also demonstrate the re-usability and the ease of end-to-end development of modeling strategies.

Sangam's work focuses on modeling mappings between data models and application schemas, declaratively describing the latter is a necessity as these are the inputs and outputs of its maps. Following the four layer UML Meta model architecture [Boo94], the data models, application schemas and data are represented in a four-tier architecture depicted in Figure 2.4.

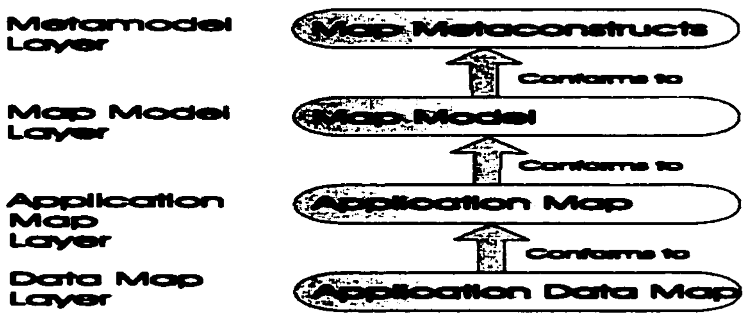


Figure 2.4: The Architecture of Sangam

In Figure 2.4, the *map metaconstructs* given in the top layer defines the building blocks required for modeling maps. A map model composed of these map metaconstructs describes the translation of a given input data model to an output data model in the map model layer. For example, we can define a map model for translating the constructs of the XML (DTD) model (e.g., element) to constructs in the relational data model (e.g., relation or attribute).

In the *application map layer*, application maps conforming to the map model structure translate an input application schema to an output application schema. For example, an application map conforming to a given map model may map the element Accommodation to a relation ACCOMMODATION, or it may map the element Name to an attribute name in the relation ACCOMMODATION. The relation can be seen in Figure 2.2.

In the bottom layer, *data maps* describe the translation of the input application data to the output application data. This data transformation is guided by the information stored in the application map. Thus, for example, the values for element Accommodation for all XML documents conforming to the Hotel application DTD are translated to rows in the ACCOMMODATION relation; and the values for element Name are mapped to a value in a column of the relation ACCOMMODATION (Figure 2.2).

Sangam uses a two-pass algorithm. In the first pass, called COLLECT, it collects a set of tables that need to be created, and gathers the set of nodes that belong to a given data model by following the containment edges from a given root. In the second pass, called MSSG (map storage structure generator), it generates the application schema storage

structure and table definitions as represented by the map model. Each map node is represented by a table with the table name the same as the node's label. Different kinds of edges are treated differently.
For the nodes, it uses the intuition that most nodes can have multiple instantiations. Thus, given a data model DM, for every row in the NODE table representing a node $n \in DM$, it generates a table in the application layer with the table name equal to the node's label. All edges between the nodes are represented as attributes of the table.

Being a transformational tool for data integrating, Sangam has some advantages:
Automatic Generation - Mapping between application schemas and the subsequent mapping between the application data can be almost completely automated, thereby increasing user productivity. But it is not a code generator integrator.
Correctness - Based on its theoretical framework, it can guarantee the correctness of the mapping strategies.
Plug-n-Play - An integration engineer can edit and customize a map.
Extensibility - mapping strategies are discovered between existing data models, they can be added in Sangam with relatively little effort when compared to writing a transformation program from scratch.

2.1.3 Interpreting XML as graph structure with relational schema: Lore

Lore standing for Lightweight Object Repository [GMW99][MAG97], is a database management system designed for semi-structured information.

Lore's original data model OEM (Object Exchange Model), is a simple, self-describing, nested object model that can intuitively be thought of as a labelled, directed graph [Ca94]. Lore's query language is Lorel, which has a familiar select-from-where syntax with certain modifications and extensions that can efficiently query semi-structured data.

Lore uses DataGuides [GW97] as the schema. A DataGuide is a concise and accurate structural summary of the underlying database. It is in fact a graph (Figure 2.6) in which every labeled path in the database appears exactly once in the DataGuide while every labelled path in the DataGuide exists in the original database. An External Data Manager is set up and this component enables Lore to get data from external sources dynamically as needed during query execution, without the user being aware of the distinction between local and external data. Figure 2.5 is the architecture of Lore system [MAG97].

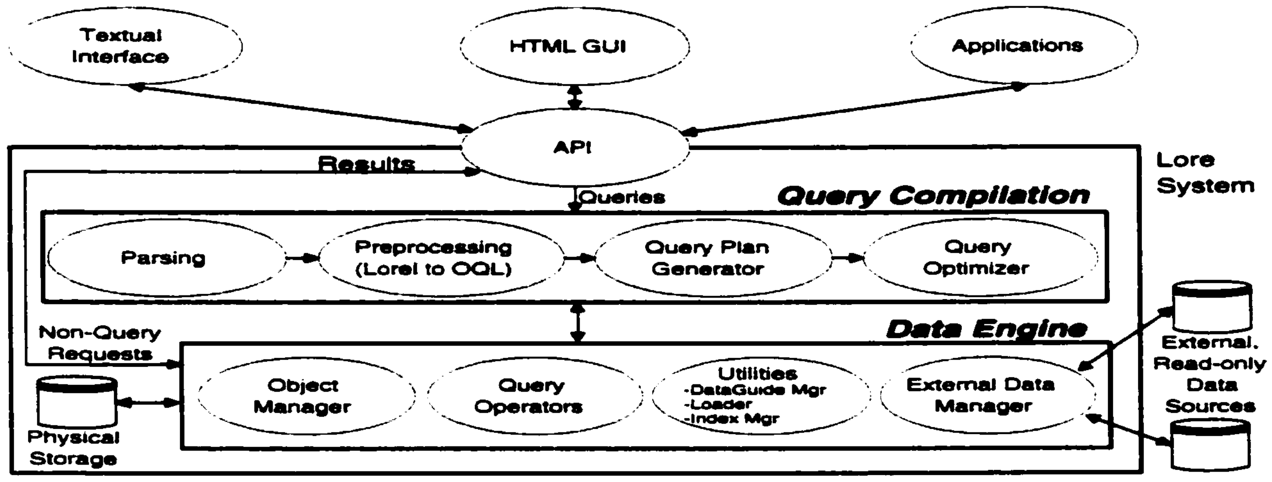


Figure 2.5: Lore architecture

In Lore's XML-based data model [GMW99], an XML element is a pair <eid, value>, where eid is a unique element identifier, and value is either an atomic text or a complex value. An XML document is mapped easily into Lore's XML data model. The model ignores comments and whitespace between tagged elements. As a base case, text between tags is translated into an atomic text element, a document element then is translated into a complex data element. Multiple XML documents can be loaded into a single database, and any system of cross-document links (e.g., XLink or XPointer) can be used provided information that uniquely identifies elements is not lost. In the XML data model, data can be visualized as a labelled, ordered graph, such as Figure 2.6 [GW97].

```
<DBGroup>
  <Member Name="Smith" Advisor="m1" >
    <Age>28</Age>
  </Member>
  <Member ID="m1" Project="p1">
    <Name>Jones</Name>
    <Advisor>Ullman</Advisor>
  </Member>
  <Project ID="p1" Member="m1">
    <Title>Lore</Title>
  </Project>
</DBGroup>
```

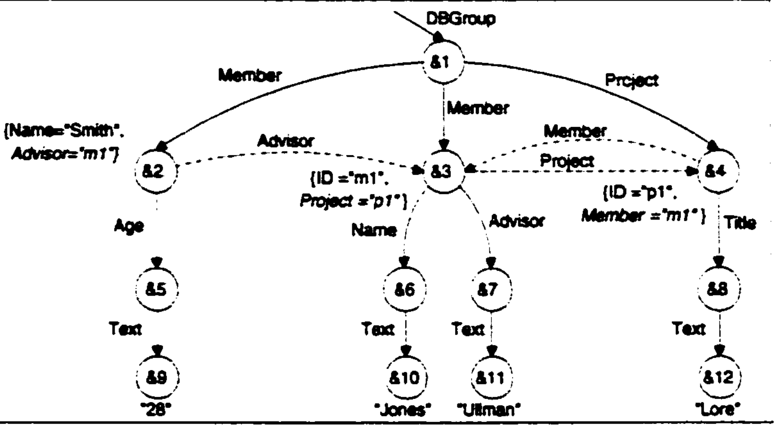


Figure 2.6: An XML document and its graph

A path expression in Lorel is essentially a sequence of labels, such as dbgroup.project.title. In the query evaluation process, path expressions are equal to paths in the database graph. In the XML model, it includes both attributes and sub-elements. Lore uses Path Expression Qualifiers to distinguish the attributes and sub-elements.

Many comparisons are used when querying XML data in the Lore system. In Lorel, the default semantics transformation rules are as follows [GW97]:

1. For an atomic (Text) element, the default value is the text itself.
2. For elements that have no attributes and only one or more Text elements as children, the default value is the concatenation of the children's text values (a restricted case of the Concatenate function).
3. For all other elements, the default value is the element's eid represented as a string (the Eid function).

By the path expression qualifiers and the comparison functions, the Lore system can query the data in the XML documents. The following is an example of a query in the Lore system (for Figure 2.6):

Select DBGroup.Member.Name

Where DBGroup.Member.Advisor like "%Ulman"

The result of this query is: "Jones"

The Lore system also extended Lorel language: it can use Range qualifiers, the range is a list of single numbers or ranges that allows users to query the XML data for a certain range. After obtaining the query from the XML database, the Lore system can give users an ordered list. For example: there is such a clause: *"order-by document-order"*. Document order is frequently exactly what users want, but it can produce unintuitive results for graph-structured data. In the Lore system, the XML data can be restructured by queries. Lorel has the ability to create both attributes, elements, and order-relevant updates (e.g., *"Inserting name after the sixth subelement"*).

Lore is a famous system in the web database area and the Lore-XML system is the extension of the original Lore system. It has many useful features, but it is still not good enough, because it needs large space, and the indexing system is not perfect. In fact, this system is still under improvement.

2.1.4 Infomaster

Infomaster is an information integration system that provides integrated access to multiple distributed heterogeneous information sources on the Internet, thus giving the illusion of a centralized, homogeneous information system [GGK95]. It is based on the virtual data integration approach. In the virtual data integration approach, there is no predefined data model and any query is based on a set of query rules. The core of Infomaster is a facilitator that dynamically determines an efficient way to answer the user's query using as few sources as necessary and harmonizes the heterogeneities among these sources. Its architecture is shown in Figure 2.7.

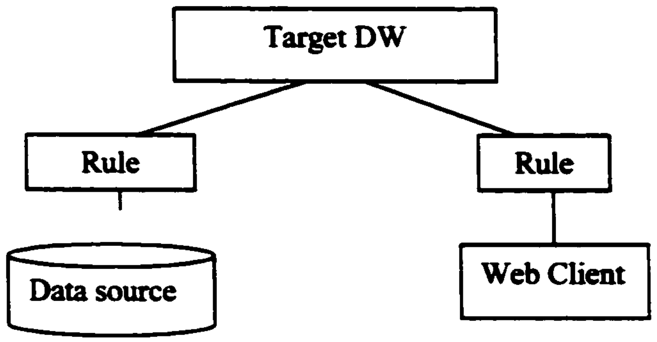


Figure 2.7: Architecture of Infomaster

Infomaster uses rules and constraints to describe information sources and translation into common form (target DW). Every translation is based on these rules and constraints.

An example from [GKD97], supposes there is a source with the transferring rules, Mercedes car table (Figure 2.8) and the target *common table* (Figure 2.9). The purpose of this demonstration is to integrate different data from different tables to the common (target) table, e.g., rule 3 says Honda sedan car is a Honda car with 4 doors. So in the target table, the Honda sedan car will have 4 doors and 4 seats (from rule 6) after the data are extracted from source table.

```
1. ((door ?x  2) <= (type ?x sport))
2. ((door ?x  2) <= (type ?x coupe))
3. ((door ?x  4) <= (type ?x sedan))
4. ((seat ?x  2) <= (type ?x sport))
5. ((seat ?x  4) <= (type ?x coupe))
6. ((seat ?x  4) <= (type ?x sedan))
7. ((range ?x ?y) <= (x*y?x?y)(fuel ?x ?y)(mpg ?x ?y)
```

Japanese Model	Type	Fuel	Miles per gallon (MPG)	Price
Civic	Sedan	2.0	30	25000
Honda	Sport	1.7	28	32000

Figure 2.8: Honda cars table and transferring constraints

In the Honda car source table (Figure 2.8), the attributes appear different from the target table (Figure 2.9). Also the name and price, the type, fuel and Mpg are not in target table. Searching the target table, the name can be obtained from the table's name—Honda. The range can be obtained from fuel * Mpg. There is no information about doors and seats from source table. In this situation, additional information in the rule is required, e.g., from the first rule (<= (door ?x 2) (type ?x sport)) and the fourth rule (<= (seat ?x 2) (type ?x sport)) in Figure 2.8, because the car type is *sport*, there are two doors and two seats corresponding to *sport* car. Again, from the third rule (<= (door ?x 4) (type ?x sedan)) and the sixth rule (<= (seat ?x 4) (type ?x sedan)) in Figure 2.8, because the car type is *sedan*, there are four doors and four seats corresponding to *sedan* car. Filling the information offered by rules into the target table, the integrated common table is shown in Figure 2.9:

Model	Makes	Doors	Seats	Range	Price
Civic	Honda	4	4	600	25000
Accord	Honda	2	2	476	32000

Figure 2.9: The target common table

The algorithm of Infomaster consists of *query planning* and *query execution*. The *query planning algorithm* takes a query Q, a collection Δ of rules and integrity constraints (like definitions) as inputs. The output of the algorithm is a query process plan suitable for input to the plan execution algorithm. The *plan execution algorithm* takes a query plan as input. It retrieves data from the available databases and merges this as described in the plan. The output is a table of answers to the original query. In the query planning algorithm, the step includes reduction, abduction, conjunctive and disjunctive minimization and grouping [DG97]:

(1) *Reduction*: In this step, the system rewrites each atom in the query using the definition of the associated predicate. It then repeats the process until it obtains an expression in terms of base relations (which, by definition, have no definitions of their own.) Termination is assured due to the non-recursive nature of the definitions.

(2) *Abduction*: Given an expression φ in terms of base relations and a set of definitions, the abduction process produces the set R of all consistent and minimal conjunctions of retrievable atoms that can be shown from the definitions to be contained in φ.

(3) *Conjunctive Minimization*: In this step, eliminating any redundant conjunct, i.e. one that can be shown to contain the remaining conjuncts.

(4) *Disjunctive Minimization* In this step, Infomaster drops any disjunction that can be shown to be contained in the union of the remaining disjunction.

(5) *Grouping*. Finally, the conjuncts within each conjunction are grouped so that the atoms in each group all share a common provider.

2.2 Code Generator

Code generator is a very important tool for building a data warehouse. There are many kinds of code generator, such as a stack based, rule based code generator, table-driven code generator and a template driven code generator. The basic idea of a stack based code generator is to store the variable and operator information into a stack during the programming language translation, e.g., GENTLE [Ge97]. In fact, it is also the major approach in compiler field but it is not convenient for a data warehouse code generator. The rule based code generator needs a set of rules (specification language syntax) with which it produces target output code. This approach asks users to learn a new rule definition language before generating code, e.g., OBLOG [Ko92]. The table-driven approach can be used in the data warehouse integration, such as SAS to specify mapping rules. The template-driven code generating technique can be used to implement target code language syntax and to integrate mapping formulas in the program code syntax [Liu01]. The template driven code generator is better because it is extensible to any language. In fact, many visual tools, such as Jbuilder, Visual Basic, Template Manager [Re92] use the template code generator approach. The WIG [Liu01] is implemented by the template-driven code generator approach.

2.2.1 SAS

SAS/Warehouse Administrator [SWA99] is a customizable solution that offers a single point of control, making it easier to respond to the ever-changing needs of the business community. It integrates extraction, transformation and loading tools for building and managing data warehouses. It also provides a framework for effective warehouse management through metadata. With graphical user interface, SAS can simplify the visualization, navigation and maintenance of the data warehouse.

SAS integrates data sources using a set of processes, e.g., *Address Standardization Add-In* routine demonstrates how to use an add-in tool to standardize an address in a data warehouse environment. The algorithm (1) Breaks down address strings to tokens. (2)

Standardizes tokens. (3) Categorizes and rearranges tokens. (4) Computes scores for evaluating the extent of standardization. (5) Outputs tokens classified as unknown by the routine to a separate data set for further investigation. (6) Constructs standardized addresses. As a result, this algorithm converts different formats of the same address to a single standardized form. For example, the two addresses: Apart 9, 564 Northyork Main Street and 564 NY Main Str, Apartment 9 would both be standardized to 546 NY MAIN ST APT 9. Another example, called *Data Step Mapping Add-In program*, demonstrates a user-written one-to-one data mapping process by extracting information by metadata.

A limitation of the *SAS Warehouse Administrator* is only setting up and managing SAS data warehouse with SAS data and no evidence shows new data integration approaches will be used. It is not flexible for generating any other language program for data warehouse data integration

2.2.2 WIG – A Warehouse Integrator Code Generator

WIG [Liu01, ELLS02] is a code generator for data warehouse data integration that can deal with many kinds of source data. One part which has been implemented is to solve a problem when the sources include only relational databases (structured data sources), and the target data warehouse is based on the general architecture of data warehouse (Figure 1.2).

The sets of WIG result programs include [ELLS02]:

- *A table creating program* which creates a set of data warehouse tables as the target tables
- *A data extracting program* which translates different format data into a uniform format data based on virtual data integration approach.
- *A data cleaning program* which eliminates non-useful data and redundant data
- *A data loading program* which inserts cleaned data into the data warehouse.

If the code generator is a black box, the input to this box will be (1) Source data descriptions, (2) Data warehouse schema description, (3) Constraints, and (4) Target language syntax. The WIG system in the first step generates data warehouse data integration programs for extracting, cleaning and loading data from source data sources to the data warehouse. These programs are then run to perform the desired functions [ELLS02].

The working process of the WIG code generator is as follows: The first step is the creation of the empty template (looks like a blank Visual Basic template). In WIG, the templates for pro.C database language would be 'creating.pc' (for Java will be creating.java etc.), for creating a data warehouse table, 'extracting.pc' for integrating source data and extracting to the target table, 'cleaning.pc' for removing the data redundancy and 'loading.pc' for loading data to the data warehouse. The next task is the population of the components of each template (looks like putting control button or text area component into the Visual Basic template). The step involves setting properties for each component, and this is done through the parsing of templates and using integration mappings (looks like set property for control button and text area components to issue what the component will do) [ELLS02].

The data integration subsystem accepts the data integration source-to-warehouse mappings from data integration rule generator and integrates these attributes based on this set of data integration mappings. Any integration only relating metadata and table definition will be integrated here (because the code generator for integrating the warehouse is only for creating target programs based on the mapping information from source metadata, not for processing data directly). After running data integration subsystem, a set of statements will be output to code generator to create target programs. By generating target programs, which are written in Pro.C, users can integrate source data in the relational databases to the data warehouse.

The previous WIG system only deals with relational data sources.

Chapter 3: Warehouse Integrator Code Generator for XML

As discussed in previous chapters, relational data warehouses can integrate data from XML source documents, use this information for decision support, and then the data warehouse system can provide users with some XML format views from the relational data warehouse. DWDI (Data Warehouse Data Integration) is very important for data warehouse system development. This chapter will discuss techniques of Warehouse Integrator code generator for XML (WIG4X).

3.1 System Architecture

Generally, WIG4X goes through the following steps:

Step 1: WIG4X accepts XML Schema description and data warehouse definitions to generate the schema mapping between them.

Step 2: WIG4X accepts integration constraints and target language syntax in form of a template, generates the data warehouse data integration programs in Java for extracting, cleaning and loading XML data from sources to the data warehouse.

Step 3: Running the generated programs to accomplish XML data integration.

Step 4: WIG4X accepts user queries and generates programs for creating the XML views from the data warehouse, and then runs the programs to create the views in XML format.

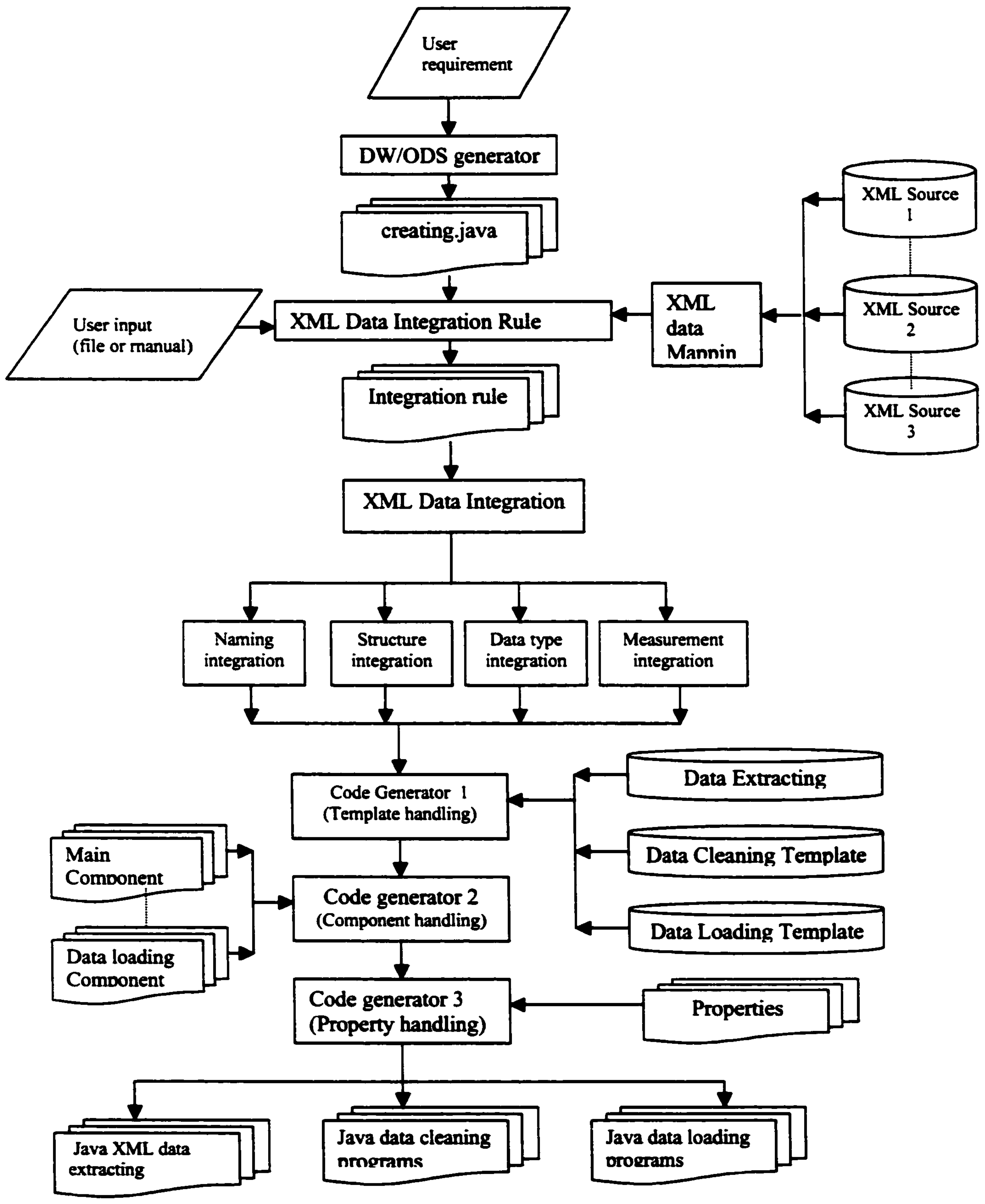


Figure 3.1: WIG4X Architecture (part 1: XML data integration)

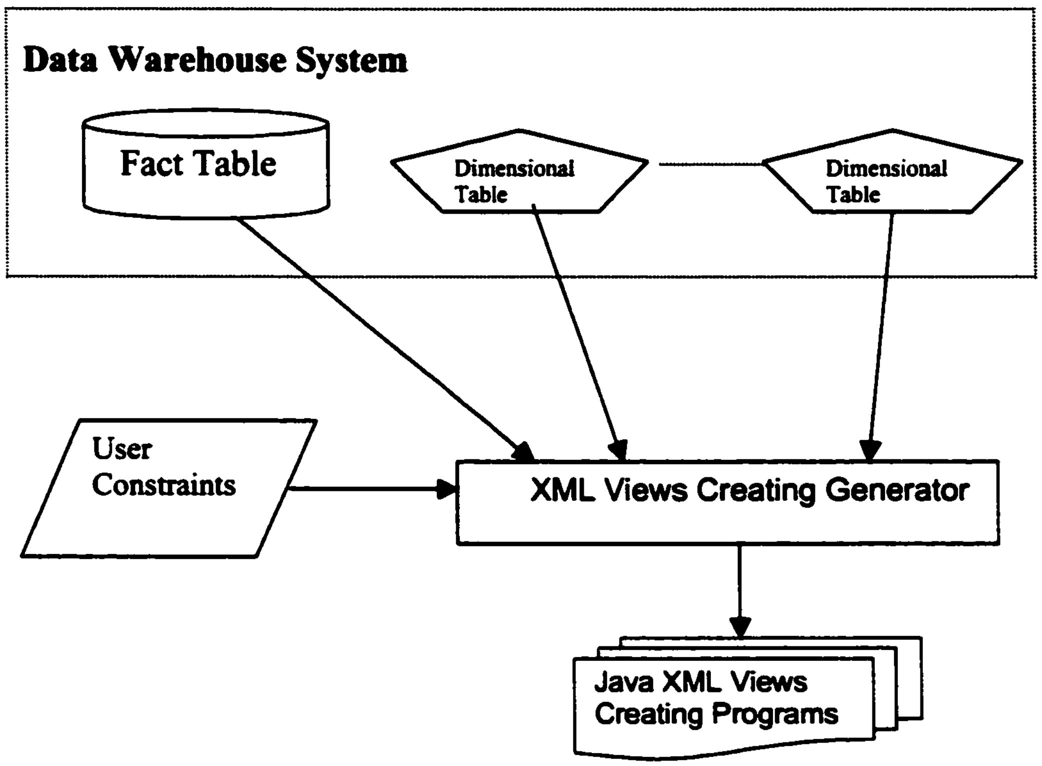


Figure 3.2: WIG4X Architecture (part 2: Creating XML views)

Figure 3.1 is part 1 of the WIG4X architecture for generating programs for integrating data from XML source documents to relational data warehouse, part 2 is for generating programs for creating XML views from the relational data warehouse system.

From the architecture in Figure 3.1, *Data warehouse table generator* is a tool for generating target programs in certain languages (Java in WIG4X); the program can be used for creating data warehouse tables and operational data store (ODS) tables. This generator accepts user query information about what table name and attributes of that table are needed or of interest, checks the information validation and generates data warehouse table creation programs with certain target language, e.g., Java table creation programs (named creating.java).

XML data integration rule generator is a tool for creating the XML data integration rules for future data integration purposes. This generator gets XML source documents' schema from the XML Schema file, accepts operational data store tables definitions from the data warehouse table generator, metadata about table definition and their attribute information from source databases and the user's assignments about the relationship among these attributes between source XML documents and operational data store. It generates a set of data integration rules based on the virtual data integration approach to create one attribute to one attribute (1 to 1) mapping between data in the source data and the data warehouse.

The XML data extraction and integration subsystem accepts data integration attribute mapping expressions from the XML data integration rule generator and integrates these attributes based on the set of XML data integration mappings. Schema integration is done here. Data extraction and integration will be done with Java programs. After the data integration subsystem, a set of properties will be output to the code generator and used to create the target data warehouse data integration programs.

Code generator is a tool for generating the target programs. It accepts variable property information from the data integrating subsystem to create the target code. The Code generator uses a template based code generation approach. In code generator, first it creates a new empty template (initial run) or renews an existing template for data extraction template (called extracting.java), creates a data cleaning template (cleaning.java) and data loading template (loading.java). It adds necessary components into each template, e.g., puts a database connector into the template for linking the database, sets the properties transferred from the data integration subsystem for each component to define what the template exactly does. The output of the code generator is a group of programs that can be used for handling real data integration.

Java program group is a set of programs generated from the code generator for creating data warehouse tables, extracting data from source XML documents with different schemas to operational data store, cleaning redundant and dirty data in the operational data store and finally, loading data from the operational data store into the data warehouse.

The *XML views creating generator* (Figure 3.2) is a tool for generating the target programs. It extracts table definitions from the fact table and dimension tables in the data warehouse system, accepts user requirements, and then generates the Java programs. By the generated programs, many useful views in the XML format will be created from the data in the data warehouse system.

In the WIG4X system, many XML data extraction and integration techniques are involved and flexible code generation techniques are also employed by this system. These techniques are discussed in the following sections, and we also give some small examples.

3.2 WIG4X XML Data Extraction and Integration Techniques

XML data extraction and integration process is very important in WIG4X system. Many techniques are needed in this process including (1) how to extract the schemata from the source XML documents, (2) how to map the user interested elements/attributes of the XML source documents to the attributes of the relational data warehouse tables, (3) how to extract the selected content from the source XML documents, and (4) how to integrate these XML data to the target data warehouse. These techniques are discussed next.

3.2.1 XML Schema Mapping and Data Extraction

In order to integrate the data in XML source documents to the data warehouse, firstly, WIG4X will extract the schema of each XML documents. From the schema of the XML documents, WIG4X will get the structure of the source XML documents, and then use that and the definition of the target data warehouse schema to create the corresponding mapping between them. The schema of the XML document can be extracted from the corresponding XML schema file, DTD file or the XML document itself.

In previous chapters, we introduced some approaches in mapping XML schema to a relational database, most of them (actually, up until now, all of them) use XML DTD file to do the mapping. In WIG4X, we mainly use XML Schema file to get the schema of XML documents. XML Schema is a brand new technique for XML, W3C has recommended XML-Schema as the standard schema description of XML documents. A simple example will be used to present the XML data source schema mapping and integration process. Suppose the source XML documents store the order information of one shopping system, there are two branches of this shop, the Canadian branch and US branch. The source XML data of the Canadian branch is shown in Figure 3.3:

```
CAOrder.xml:

<?xml version="1.0"?>

<CAOrders>
  <CAOrder orderID="ca12345678" orderDate="2001-05-20">
    <CACust CustID="ca98765432">
      <name> Alice Smith </name>
      <street> 123 Maple Street </street>
      <city> Mill Valley </city>
      <postcode> N8B 2S6 </postcode>
      <phone> 417-5633889 </phone>
      <gender> Female </gender>
    </CACust>
    <CAItem ItemID="t12568">
      <name> Lawnmower </name>
      <brand> Welch </brand>
      <type> 4000W </type>
      <quantity>1</quantity>
      <CAPrice>148.95</CAPrice>
    </CAItem>
  </CAOrder>
</CAOrders>
```

Figure 3.3: XML data for Canadian branch CAOrder.xml

The XML schema file for CAOrder.xml is shown in Figure 3.4:

```
CAOrder.xsd:

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="CAOrders" type="CAOrdersType"/>

<xsd:complexType name="CAOrdersType">
  <xsd:element name="CAOrder" type="CAOrderType"/>
</xsd:complexType>

<xsd:complexType name="CAOrderType">
```

```
 <xsd:sequence>
  <xsd:element name="CACust" type="CACustType"/>
  <xsd:element name="CAItem" type="CAItemType"/>
 </xsd:sequence>
 <xsd:attribute name="orderID" type="xsd:string"/>
 <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>

<xsd:complexType name="CACustType">
 <xsd:sequence>
  <xsd:element name="name"  type="xsd:string"/>
  <xsd:element name="street" type="xsd:string"/>
  <xsd:element name="city"  type="xsd:string"/>
  <xsd:element name="postcode"  type="xsd:string"/>
  <xsd:element name="phone"  type="xsd:string"/>
  <xsd:element name="gender"   type="xsd:string"/>
 </xsd:sequence>
 <xsd:attribute name="CustID" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="CAItemType">
 <xsd:sequence>
  <xsd:element name="name"  type="xsd:string"/>
  <xsd:element name="brand" type="xsd:string"/>
  <xsd:element name="type" type="xsd:string"/>
  <xsd:element name="quantity" type="xsd:positiveInteger"/>
  <xsd:element name="CAPrice"  type="xsd:decimal"/>
 </xsd:sequence>
 <xsd:attribute name="ItemID" type="xsd:string"/>
</xsd:complexType>

</xsd:schema>
```

Figure 3.4: XML schema file for CAOrder.xml

The source XML data of US branch is shown in Figure 3.5:

```
USOrder.xml:

<?xml version="1.0"?>

<USOrders>
  <USOrder orderNum="23985432" orderDate="2001-03-17">
    <USCust  CustKey="us45732432">
      <fname> Bill </fname>
      <lname> Johnson </lname>
      <address> 567 Rankin St. Detroit 14838 </address>
      <phone> 513-48657 </phone>
      <gender> M </gender>
    </USCust>
    <USItem ItemNum="7777345" >
      <name> Laser Printer </name>
      <brand> HP </brand>
      <type> 3000 </type>
      <quantity>3</quantity>
      <USPrice> 300 </USPrice>
    </USItem>
  </USOrder>
</USOrders>
```

Figure 3.5: XML data for US branch USOrder.xml

The XML schema file for USOrder.xml is shown in Figure 3.6:

USOrder.xsd:

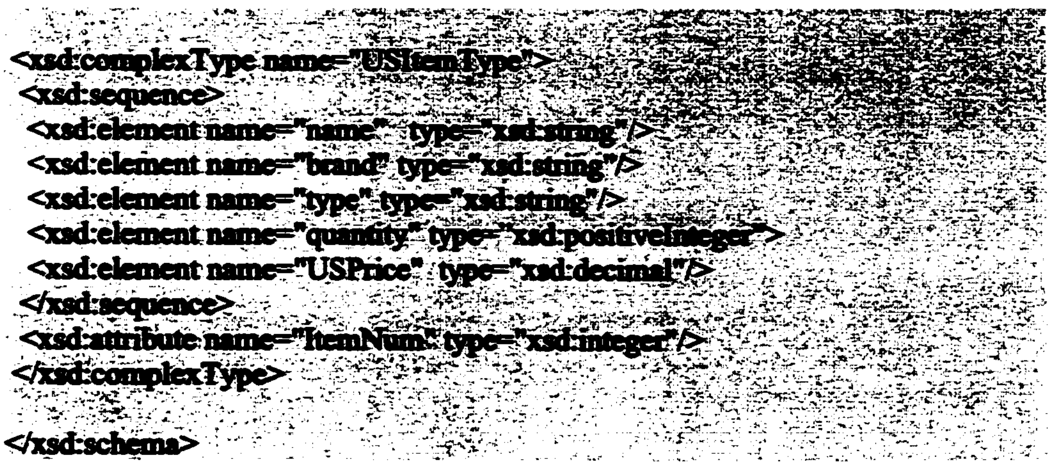
```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="USOrders" type="USOrdersType"/>

<xsd:complexType name="USOrdersType">
  <xsd:element name="USOrder" type="USOrderType"/>
 </xsd:complexType

<xsd:complexType name="USOrderType">
 <xsd:sequence>
  <xsd:element name="USCust" type="USCustType"/>
  <xsd:element name="USItem" type="USItemType"/>
 </xsd:sequence>
 <xsd:attribute name="orderNum" type="xsd:integer"/>
 <xsd:attribute name="orderDate" type="xsd:date"/>
 </xsd:complexType>

 <xsd:complexType name="USCustType">
 <xsd:sequence>
  <xsd:element name="fname"  type="xsd:string"/>
  <xsd:element name="lname"  type="xsd:string"/>
  <xsd:element name="address" type="xsd:string"/>
  <xsd:element name="phone"  type="xsd:string"/>
  <xsd:element name="gender"  type="xsd:string"/>
 </xsd:sequence>
 <xsd:attribute name="CustKey" type="xsd:string"/>
 </xsd:complexType>
```



```
<xsd:complexType name="USItemType">
 <xsd:sequence>
  <xsd:element name="name"  type="xsd:string"/>
  <xsd:element name="brand" type="xsd:string"/>
  <xsd:element name="type" type="xsd:string"/>
  <xsd:element name="quantity" type="xsd:positiveInteger">
  <xsd:element name="USPrice"  type="xsd:decimal"/>
 </xsd:sequence>
 <xsd:attribute name="ItemNum" type="xsd:integer"/>
</xsd:complexType>

</xsd:schema>
```

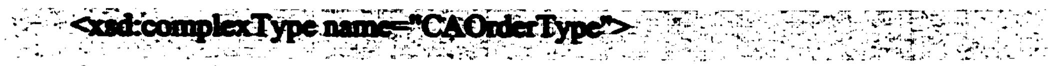
Figure 3.6: XML schema file for USOrder.xml

As discussed in Chapter 2, X-Ray [KKR00] uses some mapping mechanism between XML DTD file and relational schema, the basic kinds of mappings at the data model level may be distinguished as: ET_R, ET_A and A_A. In WIG4X approach, we improve this mapping mechanism to the mapping between XML Schema file and relational schema, since XML Schema file is more powerful, we will also develop the data types mapping between XML documents and relational database.

At the data structure level, we use these kinds of mappings:

1. XET_R: in XML Schema file, an xsd:complexType or xsd:element type (XET) is mapped to a relation (R). And several xsd:element types can be mapped to one base relation.

   For example, in CAOrder.xsd (Figure 3.4),

   ```
   <xsd:complexType name="CAOrderType">
   ```

   can be mapped to a relational relation "CAOrder" in Figure 1.1. .

   <xsd:element name="CACust" type="CACustType"/>

   can also be mapped to a relation "CACust" in Figure 1.1.

2. XET_A: An xsd:element (XET) type is mapped to a relational attribute (A), whereby the relation of the attribute represents the base relation of the element type. And several xsd:element types can be mapped to a relational attributes of one base relation. For example, in CAOrder.xsd (Figure 3.4).

```
<xsd:element name="name"  type="xsd:string"/>
<xsd:element name="street" type="xsd:string"/>
<xsd:element name="city"  type="xsd:string"/>
```

each of the above xsd:element can be mapped to one attribute in the "CACust" relation in Figure 1.1.

3. XA_A: An XML xsd:attribute (XA) is mapped to a relational attribute (A) which has its relation representing the base relation of the XML attribute. Again, several attributes can be mapped to the attributes of one base relation.

   For example, in CAOrder.xsd (Figure 3.4):

```
<xsd:attribute name="CustID" type="xsd:string"/>
```

   The xsd:attribute "CustID" can be mapped to an attribute in the "CACust" Relation in Figure 1.1.

The above mapping methods can also be combined together to extract the structure schema of the XML documents and build up the mapping between the XML elements/attributes and the data warehouse tables' attributes.

Since the XML Schema file has a great many data type definitions, we also develop the data types mapping between the XML Schema file and relational database. For example, in CAOrder.xsd (Figure 3.4):

```
<xsd:element name="type" type="xsd:string"/>
<xsd:element name="quantity" type="xsd:positiveInteger"/>
<xsd:element name="CAPrice" type="xsd:decimal"/>
```

The data type definition for each xsd:element is provided, so from the data type of the XML Schema file, we can develop the mapping between XML data types and relational database attribute data types.

For example, for the Oracle database system, we can develop the data type mapping in Figure 3.7. Above is just an incomplete sample of the mapping between XML Schema data types and Oracle data types; there are still some complex data types. For example: the <xsd:complexType> data type means the current element contains some sub-elements. In this case, we will go to its sub-elements to extract the primary data types as in Figure 3.7. From the data type mapping, we can develop not only the data structure mapping, but also the data types of XML documents to the relational data warehouse. In Figure 3.10, the data type of each attribute in the fact table is extracted by the XML schema data type mapping.

XML Schema data types	**Oracle data types**
decimal, integer, positiveinterger	Number
string, Name	Varchar2
NOTATION, language, token	Char
long, float, double	Long
date, recurringDate, time	Date

Figure3.7: XML Schema data type mapping

Having the XML schema, we can set up the programs for XML data content extraction and integration.

The algorithm for XML tree structure extraction from an XML schema file is given as Figure 3.7a:

This is a recursive algorithm, it begins with the XML schema file's first node, and then parses it and its children's nodes recursively until all the nodes have been parsed. All the related nodes will be added to a tree structure with the corresponding parent–children relationship.

```
Input: XML schema file (xmlfile)
Output: Tree structure (xmltree)

Begin:

Main {
Document doc = null;
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
doc = db.parse(xmlfile);
Node root = (Node)doc.getDocumentElement();
ParseNode (root);
}

ParseNode (Node currentNode) returns JTree
{
 if currentNode is element
    {
     if currentNode's attribute type is typical data type
       {
        add currentNode to the xmltree's current node
        }
      else if currentNode's type is complexType
           {
            parse to find the complexType element which is named as the currentNode;
            if found
               {
                ParseNode ( its children nodes);
               }
            }
    }
  if currentNode is attribute
    { add currentNode to the xmltree's current node  }

  if currentNode has children nodes
    { ParseNode (its children nodes) }

  return xmltree;
 }

 End
```

Figure 3.7a: The algorithm for XML tree structure extraction from an XML schema file

3.2 .2 WIG4X XML Data Integration Mapping Expression Generator

XML Schema can be mapped to relational schema, using proper attribute mapping expressions; and can then integrate source XML data the same way as integrating relational source data. In previous work of Warehouse Integrator Code Generator for relational database (WIG), data integration techniques and code generation techniques have been discussed. The following strategy just comes from the previous WIG [Liu01] architecture design and is based on DATALOG, a version of first order logic used by INFOMASTER [GKD97].

From the XML source data and warehouse schema descriptions and XML data conversion constraints, the WIG4X data integration tool generates the transformational mapping expression for the XML data integration. The expressive language is based on a type of first order logic for defining the mapping between XML data sources and integrated schema [DG97][Liu01]:

1. The mapping between the XML source data schema (ϕ) and the integrated warehouse schema (ψ) is expressed as a set of definitions, each of the form $\phi = \psi$, where ψ is the head and ϕ is the body of the definition. The head ψ is an atom or an n-ary predicate of the form $p(\tau 1, \ldots, \tau n)$. Each tuple τ_i of the head atom can be either a constraint or a variable from integrity or data conversion constraints. The body of the definition ϕ is a positive Boolean expression of one of the following form: an atom, a conjuction ($\wedge$) or a disconjunction ($\vee$). Definitions are required to be safe in the sense that variables appearing in the head must appear in the body and variables in the body but not in the head are assumed to be existentially qualified. The bodies of definitions are also required to be uniform such that disjuncts in every disjunction should contain the same head variables. Definitions are unique so that a predicate occurring in the head of one definition is not allowed to appear in the head of another definition. Definitions are also non-recursive because if predicate, x, occurs in definition of y, then y cannot occur in the definition of x or any predicate used to define x.

2. Integrity constraints are written in the form $\neg\psi$, where ψ is a positive Boolean expression. Variables in integrity constraints are assumed to be universally quantified. Data conversion constraints are written in the form $\psi = \phi$, ψ represents a variable of a table in the warehouse, ϕ (body) represents variable(s) in the XML source's mapping relations. The body of the data conversion constraint is an arithmetic, string, or Boolean expression.

An example of a conversion constraint that can be developed appears in Figure 3.6. In "USOrder", the customer's name is composed by <firstname> and <lastname> elements; then the name field in the customer table of the warehouse comes from:
Warehouse.customer.name = USOrder.USCust.fname +" "+USOrder.USCust.lname

Following the above definitions of the expressive language, we can give a summary of the algorithm used to produce the transformational rules:

1. Reduction: WIG4X re-writes each atom (e.g., Relation) in the data warehouse structure using the definition of the associated predicate (mapping relation) of the XML source data. The process is repeated until an expression is obtained in terms of base relations. For example, suppose the source data are XML documents in Figure 3.3 and Figure 3.5, and the warehouse fact table is: Orders (OrderID, ItemKey, CustKey, Date, Quantity, Price). The fact table is obtained from the integration of the two different XML mapping relations by applying the reduction schema through the following mapping between the warehouse fact table and the XML source relations:

Orders = CAOrder.OrderID $\wedge$ "CAOrder" $\wedge$ CAOrder.CAItem.ItemID $\wedge$
CAOrder.CACust.CustID $\wedge$ CAOrder.OrderDate $\wedge$ CAOrder.CAItem.Quantity
$\wedge$ CAOrder.CAItem.CAPrice
Orders = USOrder.OrderNum $\wedge$ "USOrder" $\wedge$ USOrder.USItem.ItemNum $\wedge$
USOrder.USCust.CustKey $\wedge$ USOrder.OrderDate $\wedge$ USOrder.USItem.Quantity
$\wedge$ USOrder.USItem.USPrice

Figure 3.8: fact table mapping after reduction

The algorithm for XML mapping generation is given as Figure 3.8a:

```
Input: data warehouse table attribute (attr) and XML tags (Stag[])
Output: XML tag which mapped to the input attr (mtag)

Begin
Mtag = Stag[0], maxmatch =0, temp =0;
For each stag ∈ Stag[]
{
 if stag is exactly the same as attr
   {
     return stag;
   }
  else
   {
    if (stag.length > attr.length)
       {
        longstring = stag;
        shortstring = attr;
       }
    else
         {
           longstring = attr;
           shortstring = stag;
           }
        if (longstring contains shortstring)
           {
            temp = shortstring.length;
            }
         else
            {
              check whether longstring contains substring of shorstring and
              assign the length of the longest matched substring to temp
             }
       if (temp > maxmatch)
         {
           mtag = stag;
            maxmatch = temp;
          }
   }
  return mtag;
}
End
```

Figure 3.8a: Algorithm for XML mapping generation

This algorithm in Figure 3.8a generates the mapping pairs between data warehouse table attributes and XML documents' tags. It uses semantic analysis, checks each XML tag's name, finds the one which most likely maps to the data warehouse table attribute, and maps it to that attribute. This algorithm is available when data warehouse table attributes and XML tags use similar names.

2. Inclusion of Data Conversion Constraints: Data conversion constraints are included in the mapping between the elements/attributes of the XML source relations and the warehouse tables obtained from step 1 above.
   For example:

```
Warehouse.customer.name = USOrder.USCust.fname || USOrder.USCust.lname
warehouse.orders.price (CAN$) = USOrder.USPrice * 1.5
```

Figure 3.9: Sample data conversion constraints

After the above steps, WIG4X will get a primary data warehouse which has the integrated XML source data. Generally, there still exist some redundancies at this stage. Then WIG4X will run its data cleaning process, which uses many approaches to find the records from different XML data sources that belong to the same entity. During the data cleaning process, redundant conjunctions and disjunctions will be dropped. This process also includes content integration, which confirms that there is no content redundancy. For example, if one customer made two orders, one was in the US branch, one was in the Canada branch, then in the data warehouse, these two orders should have the same "custKey".

From source XML in Figure 3.3 and Figure 3.5, the fact table after the primary integration is shown in Figure 3.10.

Fact table: (US represents USOrder relation, CA represents CAOrder relation)

Fact table Attributes (data type)	OrderID (Varchar2)	Branch (Varchar2)	ItemKey (Varchar2)	CustKey (Varchar2)	OrderDate (Date)	Quantity (Number)	Price(CAN$) (Number)
USOrder.xsd (data type)	OrderNum (xsd:string)	US	ItemNum (xsd:string)	CustID (xsd:string)	OrderDate (xsd:date)	Quantity (xsd:positiveInteger)	USPrice*1.5 (xsd:decimal) (US$)
CAorder.xsd (data type)	OrderID (xsd:string)	CA	ItemKey (xsd:string)	CustKey (xsd:string)	OrderDate (xsd:date)	Quantity (xsd:positiveInteger)	CAPrice (xsd:decimal) (CAN$)

Figure3.10: Example of a target data warehouse fact table

After the data cleaning, the different elements which contain the same content will be assigned a common attribute name in the fact table. For example, the USOrder.USPrice and CAOrder.CAPrice will be assigned to the fact.Price attribute.

Following the same strategy, WIG4X will also integrate information from separate XML documents to the data warehouse dimension tables, then to complete building the entire data warehouse system.

Based on these steps above, some relation definitions that are used in the virtual data integration approach are introduced. In the virtual data integration approach, the user interface and the available information sources are modeled by a set of relations, which is a tool that will be used for finding the one to one mapping between attributes of source relation and the data warehouse [DGe97].

(1) *Interface relations* are the forms of a user defined target data warehouse table. Interface relations conceptualize the interaction between the target data warehouse table and the source XML document.

(2) *Site relations* are the data available from an XML data source. Site relations represent the XML data that are actually stored in the available data sources.

(3) *Base relations* are for expressing both interface relations and site relations. Base relations are used as a means to describe both interface and site relations and are crucial in order to simplify adding new information sources.

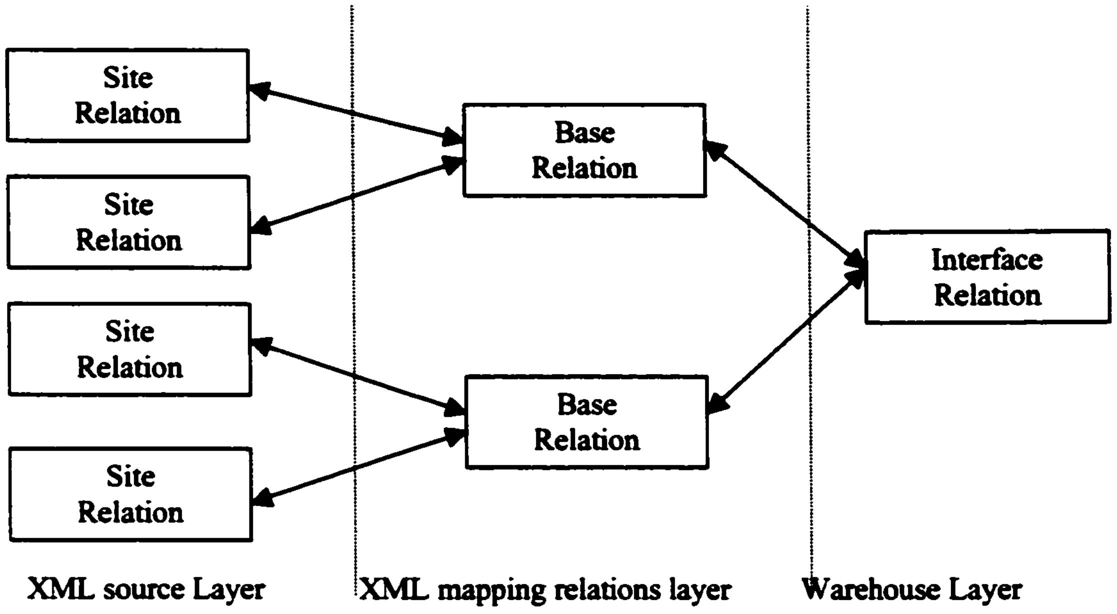


Figure 3.11: Interface Base, Site Relation Diagrams

In Figure 3.11, the site relation describes the relationship between the XML source schema and the mapping relations. Base relations represent a conceptual description of data which link site and interface relations. This architecture is flexible when a data warehouse corresponds to an unknown number of source XML documents (flexible to increase source or delete sources).

From the discussion above, the data integration process for XML data in Figure 3.3 and Figure 3.5 can be summarized in the following steps:

1. Extract XML data source schema from the XML schema file or DTD file, if neither of them exists, extract it from the XML file directly.
2. Combine XET_R, XET_A and XA_A mapping strategies; map XML elements/attributes to target table attributes, develop corresponding transformational attribute mapping expression which are defined according to first order logic. The developing algorithm includes reduction (Figure 3.8) and Inclusion of Data Conversion Constraints (Figure 3.9) process. After this step, a primary data warehouse and XML data source mapping with some redundancies is obtained. At this stage, the target data warehouse fact table contains the

attributes which belong to the same entity but come from CAOrder.xml and USOrder.xml respectively.

3. Run the data cleaning process to obtain the target data warehouse tables. Such a target fact table schema and its mapping to XML data source schemata are shown in Figure 3.10. Dimension tables also created with these strategies.
4. Load XML data into the corresponding data warehouse tables.

A sample fact table integrated from CAOrder.xml and USOrder.xml is shown in Figure 3.12:

OrderID	Branch	ItemKey	CustKey	Date	Quantity	Price (CAN$)
us23985432	US	7777345	us45732432	2001-03-17	3	450
ca12345678	CA	t12568	ca98765432	2001-05-20	1	148.95

Figure 3.12: A sample target fact table

3.3 XML views creation techniques

After integrating the XML data into the data warehouse, we also need to build up some views in XML format from the relational data warehouse. In order to do that, we will use some schema mapping strategies to transform data of interest in the data warehouse to XML format views.

1. R_XET: in XML Schema file, an xsd:complexType or xsd:element type (XET) is mapped from a relation (R) and several xsd:element types can be mapped from one base relation.
2. A_XET: An xsd:element (XET) type is mapped from a relational attribute (A), whereby the relation of the attribute represents the base relation of the element type and several xsd:element types can be mapped from a relational attribute of one base relation.
3. A_XA: An XML xsd:attribute (XA) is mapped from a relational attribute (A) whose relation represents the base relation of the XML attribute. Again, several attributes can be mapped from the attributes of one base relation.

By the above basic kinds of mappings, we can set up XML Schema for the new XML documents, and following the XML Schema, WIG4X can generate programs for creating XML format views.

For example, from the fact table in table 3.3, we can create an XML view which is used to show each item sold in one branch in a certain month:

```
<?xml version="1.0"?>
<CAOrder OrderDate="2001-05" >
    <Items ItemKey="1234567">
        <Quantity> 12345    </Quantity>
        <TotalPrice> 987654321 </TotalPrice>
    </Items>
    <Items ItemKey="1234568">
        <Quantity> 12340    </Quantity>
        <TotalPrice> 987654300 </TotalPrice>
    </Items>
</CAOrder>
```

In the above XML view, the element <Quantity> comes from the attribute fact.Quantity directly by the A_XET mapping approach, attributes "ItemID" comes from the attributes of fact.ItemKey by the A_XA mapping approach. The "OrderDate" attribute comes from the dimension Time table, which groups the Order date by year, month and day. <TotalPrice> is a newly created tag and its content is the aggregated value for each item's sale prices in a certain month.

3.4 The WIG4X Code Generation Tool

We have discussed some code generation approaches in chapter 1, WIG4X code generation tool adopts the template based code generation approach. There are currently four templates for XML data integration, each of which would use a set of template components. The templates are:

- Creating: This template is responsible for creating target warehouse tables and operational data store (ODS) tables. XML schema mapping components will be used in this template. It will be used to generate "creating.java".

• Extacting: This template is responsible for extracting XML data from XML sources to the transitional ODS. It will be used to generate "extracting.java".

• Cleaning: This template is responsible for data cleaning the transitional ODS. It will be used to generate "cleaning.java".

• Loading: This template is responsible for loading data from the cleaned transitional ODS to the target data warehouse tables. It will be used to generate "loading.java".

Besides that, WIG4X also has a template, "ViewCreating" which is responsible for creating XML views from the data warehouse tables. It will be used to generate "viewcreating.java".

Accepting the target language syntax and data cleaning rules expressed as constraints, the WIG4X will produce the target programs.

Chapter 4: Implementation

WIG4X code generator is a system that creates a group of output programs, which can be used for creating data warehouse (DW) and operational data store (ODS) tables, extracting different format data from different XML documents into ODS, cleaning the dirty data in ODS, e.g., removing redundant data, loading data from ODS into DW tables if the data does not exist in the DW, and then creating XML views from the data warehouse tables.

In this chapter, we will discuss the WIG4X project aims and system environment, present the WIG4X project architecture and finally, demonstrate an example. For more detailed operation, please see the *user manual* in Appendix A.

4.1 Project Aims and Restrictions

The aim of the WIG4X project is to implement a code generation system for data warehouse data integration based on virtual *data integration approach* and *template based code generation approach*. We suggest that this project follow these restrictions below:

Location: this thesis focuses on local data warehouse generation. That means all of the XML source documents and data warehouse are on the same machine and the output language is for Java. An integration code can however be generated for a remote machine for future use on that machine.
Source: assume this code generator is to solve a problem when the sources include only XML documents.
Generation: assume this code generator is to solve a problem based on general architecture of data warehouse (reference Figure 1.2).
Extension: assume this code generator is to solve a problem that can be extended to distributed or web systems in the future and multi generated language output. Extensions

to accommodate other different data source like HTML, and textual data in the future should be possible too.

4.2 System Introduction

The implementation of the Warehouse Integrator Generator for XML data source focuses on five different major areas:

(1) Data warehouse (target) creation

(2) XML data integration

(3) Target code generation

(4) Data processing

(5) XML views generation

The implementation of WIG4X consists of three types of processes:

(1) *Front processes* with *graphic user interface* (GUI) to accept information on user requirement from outside, to generate new information and display them back to the user.

(2) *Background processes* without input information from front processes to generate new information and output the information for future processing; this is the part which takes the most work on this system.

(3) *Target processes* are for integrating warehouse XML data source. In this implementation, WIG4X can be used for multi-user systems and personal computers. In a multi-user system, front processes were designed using JBuilder 4.0 under Sun Solaris. Background processes were designed and coded on Java under UNIX. On a personal computer, they work with Java2 platform under Window98 and Oracle8i.

The system components of WIG4X are shown in Figure 4.1.

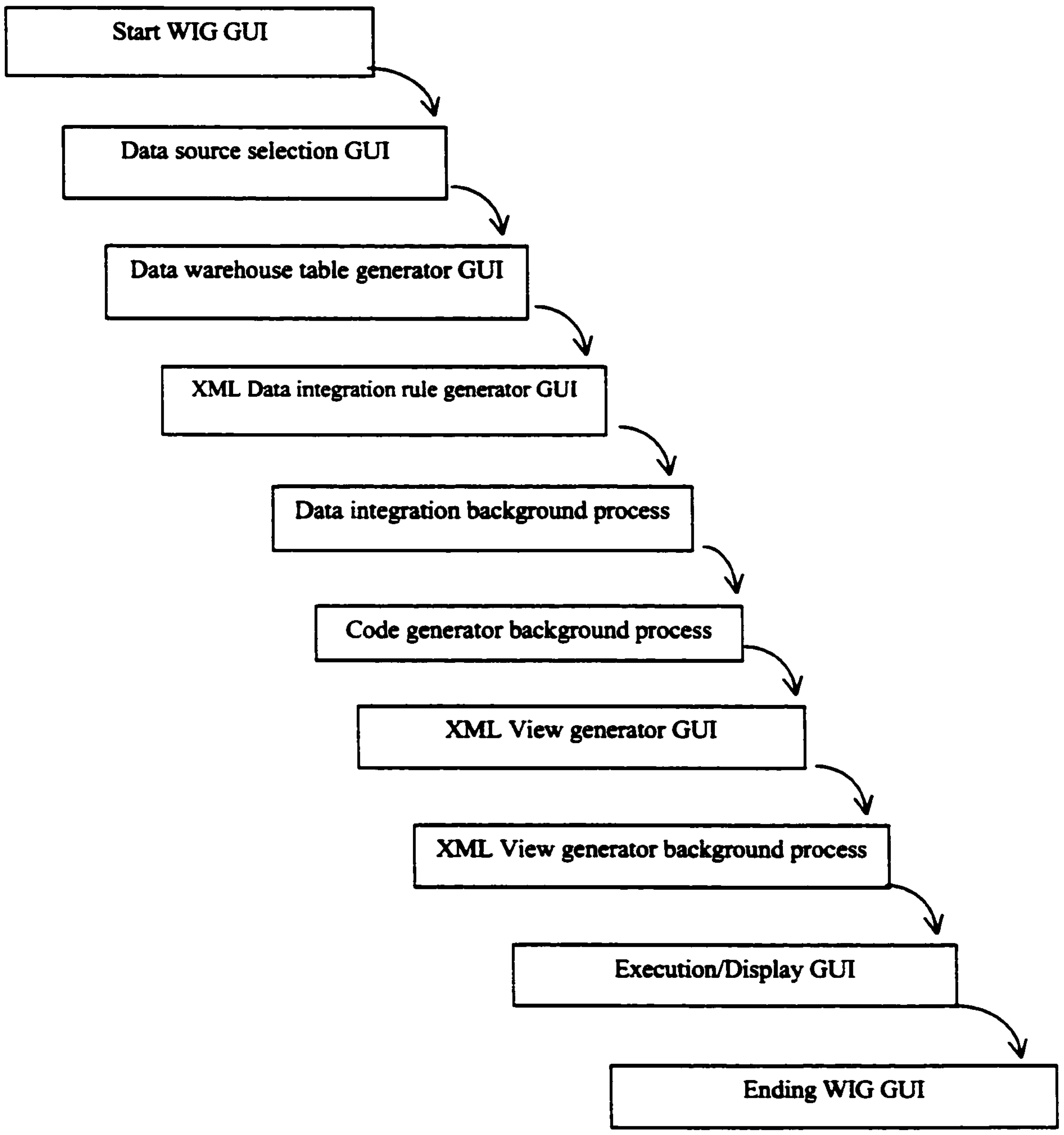

Start WIG GUI
Data source selection GUI
Data warehouse table generator GUI
XML Data integration rule generator GUI
Data integration background process
Code generator background process
XML View generator GUI
XML View generator background process
Execution/Display GUI
Ending WIG GUI

Figure 4.1 WIG4X System Components

There are seven major GUIs in WIG4X system:

(1) *Start WIG GUI* is an interface to express the system's beginning.

(2) *Data source selection GUI* is an interface to let the user select the data sources, that includes relational data, XML, HTML and text data sources.

(3) *Data warehouse table generator GUI* is an interface for accepting user interested DW definition.

(4) *XML Data Integration rule generator GUI* is an interface that accepts the user's assignment of the relationship among attributes of different XML source documents and those of the data warehouse.

This GUI is designed using dynamic coding technique. It accepts the target table definitions based on the information from DW table generator, and extract the XML file structure from the XML data source, checks the integrating validation in order to guarantee the rule correction. Finally, it outputs the integration mapping expression.

(5) *XML View generator* is an interface to extract the table schemata from the data warehouse and accepts user requirement about the target views, then generates codes for creating data warehouse XML views.*GUI Execution/Display GUI* is a GUI for displaying the resulting programs based on the dynamic programming technique and compiling, running the target data integration programs automatically. Any target program under certain language can be displayed from this interface.

(6) *End WIG GUI* is a GUI to tell user the target programs have already been done.

There are more interactive windows for information input and output such as template name input and help information output. For more detail, please check Appendix A - user manual.

There are three major background segments in WIG4X system:

(1) *XML data integration* is a segment that takes the rule as input, and generates the variable properties as output.

(2) *Code generator* is another background segment that generates the target code based on the template and rule based code generation approach. The components that will be added into the template and the properties will be modified for each component to define the exact function of the components.

(3) *XML View generator* background process is a background segment that generates the target XML views creating code based on the template and rule based code generation approach. The components that will be added into the template and the properties will be modified for each component to define the exact function of the components.

4.3 WIG4X system programs

WIG4X system consists of 19 Java programs, 6 Components, 7 graphics and 6 XML testing file. After compiling and running the WIG4X system, it will generate 5 target Java source code files and 5 executable Java class files. For detailed program codes, please refer to the Appendix B: WIG4X codes.

WIG4X has 17 Java programs, they are listed in Figure 4.2. The system starts from WIG.java and ends with endWIG.java. It accepts user requirements and generates the target data warehousing system and XML data integration programs.

WIG4X system has 6 different components, they are listed in Figure 4.3. The component files are defined by the programmer as the target program templates. Each component has a particular name and purpose, which is very important to create the target programs. For example, the mainComponentX is designed for creating the program header; it can be combined with other template components such as cleaningComponentX to create new target programs. Part of a sample component file is shown in Figure 4.6.

In WIG4X project, this system is tested with a small shopping order data warehousing system, whose structure has been introduced in section 1 (Figure 1.2). There are 3 rule text files for testing and 7 graphic files for GUI; they are listed in Figure 4.4.

Program Name	**Program Function**
WIG	Bath file for starting WIG4X
WIG.java	Start GUI
FrameX.java	Data source selection GUI
TableGeneratorX.java	Input target table definition
Frame1X.java	Target table generation GUI
RuleGeneratorX.java	Start to generate mapping rules
Frame2X.java	XML Rule mapping GUI
SourceGeneratorX.java	Extracting structure from XML schema file
SourceGeneratorMX.java	Extracting structure from XML file
GuessMapping.java	Generating XML mapping automatically
ConsGeneratorX.java	Input XML mapping constraint by manual
DataIntegrationX.java	XML data integration
MainGeneratorX.java	Generating the extracting program
CodeGeneratorX.java	Generating other programs for DW
Frame7X.java	XML view creating GUI
ExecutionX.java	Compiling and executing target programs
Frame5X.java	Running GUI for the target programs
DisplayX.java	Display the target programs
Frame3X.java	The GUI for displaying
EndWIG.java	The system ends

Figure 4.2: WIG4X system programs

Component Name	**Component Function**	**Belonging**
CreatingComponentX	Creating DW tables	Creating program
mainComponentX	Creating program header	All program
factComponentX	Generating extracting program	Extracting program
cleaningComponentX	Generating creating program	Cleaning program
loadingComponentX	Generating loading program	Loading program
viewComponentX	Generating view creating program	View creating program

Figure 4.3: Basic component templates

Shopping order testing file	**Purpose**
Targetx.txt	Data warehouse definition
Rulex.txt	XML mapping rule definition
Viewrulex.txt	XML view mapping rule definition

Figure 4.4: Shopping order testing rule files

Graphic file
Plant.gif
Boy.gif
Butterfly.gif
Windsor.gif
Angry.gif
Gificon.gif
Middle.gif

Figure 4.5: Graphic files

```
{
Connection connection;
Statement selectStatement;
String driver="sun.jdbc.odbc.JdbcOdbcDriver";
private String url="jdbc:odbc:pwok";

private String uname = //**$$UID
private String upwd = //**$$PWD

static int inteSize;
static String[] arrayFactReady;
static int[] arrayCount;

static ArrayList consCount = new ArrayList();

String attValue, elemValue, inteValue, inteValuep;
int elemposi;
int inteCount = 0;

public static void main (String[] args)
{
Document doc = null;

Vector Sxml = new Vector();
Vector SPtag = new Vector();
Vector Stag = new Vector();
Vector Constraint = new Vector();
Vector uniqSxml = new Vector();
Vector uniqSPtag = new Vector();
Vector uniqStag = new Vector();
Vector uniqCons = new Vector();

Vector Tables = new Vector();
ArrayList uniqTables = new ArrayList();

String Sourcef = new String();
String tempS = new String();

//**$$VFILL

..........
```

Figure 4.6: Part of the template for extracting program

Variable properties used in the template are shown as follows:

$$UID: Database User ID

$$PWD: Database User Password

$$CNAME: Class Name

$$VFILL: XML data source and data warehouse table descriptions, such as XML file names, tags, data warehouse table names, attributes.

4.4 Target Programs of WIG4X System

The target product of WIG4X code generator is a set of XML data integration programs. There are five generated Java programs:

(1) creating.java: creating target data warehouse

(2) extracting.java: extracting XML source data to data warehouse ODS tables

(3) cleaing.java: validating the data in ODS tables and removing the redundancy data from ODS tables

(4) loading.java: loading cleaned data from ODS tables to data warehouse tables

(5) viewcreating.java: creating XML views from data warehouse tables

4.4.1 Data Warehouse Creating Program

The data warehouse creating program, first accepts user requirements, then creates the data warehouse fact table and dimensional tables according to user definitions, then it also creates data warehouse operational data store (ODS) tables. The ODS tables have the same table schemata with the data warehouse tables, which stores temporary data before the data loads to data warehouse tables.

4.4.2 XML Data Extracting Program

The extracting program, first, goes to the XML documents, parses the XML data and extracts the user required data from the XML documents.

Since the source XML data may have a different format from that of the target data warehouse table attributes, some transformation will be made according to data integration rules.

The extracting program calculates XML elements for measurement integration based on the constraints of integration rule. The calculated data are a basis for future data integration, e.g., the amount in saving account database is represented by US dollar and in target data warehouse is Canadian dollar. Extracting program first calculates all *'price'* in saving account database by multiplying the exchange rate in order to translate the US dollar to Canadian dollar.

The extracting program deals with structure integration. When integration rules include the sentence such as "firstName + lastName", it tells us that the data in two tags will be combined together, e.g., the firstName is "Jeffrey" and the lastName is "Liu" in Customer ODS table, after structure integration, it will become "Jeffrey Liu".

The extracting program will also deal with some format transformation rules. For example, "gender: M=Male; F=Female" means when the program extract data from tag "gender", the data "M" will be transformed to "Male", "F" will be transformed to "Female", then the data in the ODS tables will be in uniform format "Male" or "Female".

For some special attributes in the data warehouse tables, there are not the direct data mapping to them. For example, the "branch" attribute in fact table, this is an integration attribute, no tags contain the data directly mapped to it, then according to the data integration rules, some special data will be put in such table attribute. For example, "CA" or "US" will be put in the "branch" attribute according to different data source "CAOrder.xml" or "USOrder.xml".

4.4.3 Data Cleaning Program

After extracting the program, the XML source data are extracted into ODS. The data in ODS contain some duplicate and incomplete data. In the shopping order system, one customer's information maybe shown in both the US branch and the Canada branch order XML documents. Such redundant data need to be cleaned and reconciled in the integrated warehouse data.

Cleaning programs first make the data validation for each dimension table. For example, <"ca1000000", "Jeff Liu", "33 Isabella Street", "Toronto"> and <"us999030031", "Jeff Liu", "33 Isabella Street", "Toronto"> refer to the same customer. In order to validate these data, the cleaning program must check the attribute to see if it is primary key or not. If only related non-primary key attributes appear, it validates the dimension table data. If it relates the primary key, it not only needs to validate the dimension table but also the fact table because any primary key in the dimension table must be in a fact table according to DW theory. For example, it converts "us999030031" to "ca1000000" in the ODS table and then validates the fact table's attributes because the key in the dimension table always links to one attribute foreign key in the fact table and that attribute in fact also needs validation.

Some incomplete data also exist in the ODS tables; for example, the time table (keydate, year, month, day). At first, only the primary key attribute "keydate" contains the data. Then we need the program to parse the date information in the "keydate" attribute and fill the corresponding data to the "year", "month" and "day" attributes.

After validation, the cleaning program will remove the redundant tuples from ODS. There are two different types of redundancy. One is there are two tuples are same in ODS. Another is one tuple in ODS is same as one of the tuple in DW. In this case, we only keep single (can not be duplicated in ODS), newer information (not in DW) in ODS. Up to now, we can say that the data in ODS are cleaned.

4.4.4 Data Loading Program

After data cleaning, the data loading program will load the cleaned data into data warehouse tables; this entails moving just the changed data in order to reduce the amount of data that has to be moved. The loaded data include both newly obtained transaction information and the changes from the source system. The loaded data will also have an index that will be used for efficient query performance. The loading program will also update the data warehouse metadata to show how many records are in each data warehouse table, how many new records are in each data warehouse table.

4.4.5 XML View Creating Program

The XML view creating programs will create views in XML format from data warehouse tables. The program is based on certain mapping rules between data warehouse table attributes and XML document tags, e.g. the data within XML tag < Itemkey> comes from the attribute "itemkey".

The data in an XML view is always selected from the data warehouse tables according to some conditions, for example, the items whose price is higher than $100. The program goes to the corresponding data warehouse table attribute, compares the value, and chooses the value based on the certain condition, and put them into the XML views.

Some data in XML views are aggregate values, for example, in an XML view which contains the sale information of one month, the data within tag <totalquantity> represent the total sale quantity for a certain item, this value is not shown in the data warehouse table, the program will add all the quantities of the item in that month, and put the aggregate value in the tag.

Some XML views need the data within them in certain order, for example, the data are ordered by date, or grouped by price. After extracting all the required data from the data warehouse tables, the program will organize the data in XML views based on certain requirements.

Chapter 5: Conclusions And Future Work

This chapter presents conclusions and future research directions in the WIG4X project.

5.1 Conclusions

In this thesis, WIG4X, a code generator for data warehouse XML data integration, based on XML schema data integration approach combined with virtual data integration approach and semantic data integration approach, is presented. A template driven based code generation approach is first presented, which is used for generating the Java programs for the data warehouse creation, XML schema mapping, XML data extraction, cleaning and loading. And then the WIG4X will also generate the programs for creating the XML format views from the data warehouse system.

The main contributions of this thesis are:

(1). Proposing techniques of WIG4X code generator for XML, which can generate programs for data warehouse XML data integration with high efficiency and flexibility.

(2). Proposing a brand new approach for applying XML Schema in the XML documents schema mapping for XML data integration.

(3). Applying a new technique DOM2 API in XML data content extraction, that makes this process very efficient.

(4). Extending the previous data warehouse integration approach of WIG for a relational database to WIG4X, to accommodate XML schema or DTD.

(5). Extending the previous code generation approach of WIG code generator for a relational database to WIG4X, which employs the template based code generation approach to generate the Java codes.

(6). Try to design and implement brand new reusable Java classes which are specially used for integrating XML data into the data warehouse.

(7). Implementing the components of the WIG4x that integrates XML data source into data warehouse by generating Java code.

5.2 Future Work

We should mention, that this thesis focuses on local data warehouse generation. That means all of the XML source documents and the data warehouse are on the same machine and the output language is for Java. An integration code can however be generated for a remote machine for future use on that machine.

- *WIG for HTML/TEXT*: A great deal of data is still in HTML or text files today, how to integrate needed information within HTML data source is a very challenging task. WIG code generator will try to extend its ability to deal with these kinds of data sources for future work outside the scope of this thesis.
- *Distribution/web*: How to create a code generation for the distributed/web data warehouse system is still a hot topic because the databases in today's market are distributed. Data warehouse needs the ability to extract the data from different site of data sources. So does code generator for much future research.
- Incorporating more efficient data cleaning and schema integration techniques to the code generator project are also interesting future work outside this thesis.

References

[AKH96] Y. Arens, C. A. Knoblock and C. Hsu, *"Query processing in the SIMS information mediator"* in Advanced Planning Technology, editor Austin Tate, AAAI press, Menlo Park, CA, 1996

[Bo97] Jon Bosak. *"XML, Java, and the future of the web."* http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm Sun Microsystems March, 1997

[Bo99] Ronald Bourret. *"XML and database"*. Technical university of Darmstadt September 1999

[BS97] Alex Berson, Stephen J. Smith *"Data Warehousing, Data Mining & OLAP"* ISBN 0-07-006272-2, McGraw-Hill, 1998

[Bu97] Peter Buneman, *"Semistructured data"*, Proceeding of ACM symposium on principles of database systems pp. 117-121, 1997

[BZ96] Ramon C. Barquin, Herbert A. Zdelstein *"Planning and designing the data warehouse"* Prentice Hall PTR 1996, P150, ISBN 0-13-255746-0.

[Ca94] R. G. G. Cattell, *"The object database standard: ODMG-93"*, Morgan Kaufmann, San Francisco, CA, 1994

[CGL+98] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi and R. Rosati, *"A schema analysis and reconciliation tool environment for heterogeneous database"* in Proceedings of the sixth international conference on principles of knowledge representation and reasoning (KR-98), p2-13, 1998

[CHS+94] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, D. Petkovic, J. Tomas, J. H. Williams and E. L. Wimmers *"Towards mutimedia information system: the Garlic Approach"* IBM Almaden research center, San Jose, 1994

[CR01] Kajal T. Claypool and Elke A. Rundensteiner *"Sangam: Modeling Transformations for Integrating Now and Tomorrow"* Computer Science Department, Worcester Polytechnic Institute, Technical Report WPI-CS-TR-01-04, March. 2001

[CRX+01] Kajal T. Claypool, Elke A. Rundensteiner, Xin Zhang, Su Hong, Harumi Kuno, Wang-chien Lee and Gail Mitchell *"Sangam: A Solution to Support Multiple Data Models, Their Mappings and Maintenance"*, Software system demonstration, Proceedings of SIGMOD'01, Santa Barabra, CA USA, May 2001.

[CSS+00] M. Carey, J. Shanmugasundaram, E. Shekita, S. Subramanian, D. Florescu, Z. Lves, Ying Lu. *"XPERANTO: Publishing object-relational data as XML."*, Workshop on the Web and Databases (Informal Proceedings), May 2000

[CKS+00] M. Carey, J. Kiernan, J. Shanmugasundaram, E. Shekita, R. Barr, B. Lindsay, H. Pirahesh B. Reinwald, *"Efficiently publishing relational data as XML documents"*, Proceedings of the VLDB Conference, Egypt, Sept. 2000

[De97] Barry Devlin, *"Data Warehouse from Architecture to Implementation"*, ISBN 0-201-96425-2, Addison-Wesley Pub Co., 1996

[DFD+99] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, Dan Suciu, *"XML-QL: A Query Language for XML"*, 8th International WWW Conference, Toronto, May 1999

[DFS99] Alin Deutsch, Mary Fernandez, Dan Suciu, *"Storing Semistructured Data with STORED"*, SIGMOD Conference, Philadelphia, Pennsylvania, June, 1999

[DG97] Oliver M.Duscka and Michael R. Genesereth, *"Query Planning in infomaster"* 1997 ACM Symp. On Applied Computing, February 1997

[DOM00] DOM 2, http://www.w3.org/DOM/, World Wide Web Consortium,2000

[EB98] Ezeife, C.I, and Baksh, S., A Partition-Selection Scheme for Warehouse Views , *Proceedings of the 9th International Conference on Computing and Information (ICCI 98)* , Winnipeg, Canada, IEEE Computer Society publication, pp. 9-16, June 17-20, 1998.

[ELLS02] C. I. Ezeife, Yi Liu, Chunsheng Liu, Ingo Schimitt, "WIG – A Warehouse Integrator Code Generator", submitted to NSF/NSERC Researcher's Conference, Jan., 2002

[Ez01] Ezeife, C.I, "Selecting and Materializing Horizontally Partitioned Warehouse Views", *Elsevier journal of Data and Knowledge Engineering*, Vol. 36, No. 2, pp. 185-210, 2001.

[FK99] D. Florescu and D. Kossmann, *"Storing and Querying XML Data Using an RDBMS"*, In *Bulletin of the Technical Committee on Data Engineering*, pages 27–34, Sept. 1999.

[FLM98] Florescu, D., Levy, A., Mendelzon, A, *"Database Techniques for the World Wide Web: A Survey"*, ACM SIGMOD Record, Vol. 27, No. 3, September, 1998

[FTS00] [38] M. Fernandez, Wang-Chiew Tan, Dan Suciu, *"SilkRoute: Trading between relations and XML"*, 9th international WWW conference, May 2000

[Ge97] GENTLE project http://www.sas.com/products/wadmin/doc

[GKD97] Michael R. Genesereth, Arthur M. Keller, and Oliver Duschka, *"Infomaster: An Information Integration System,"* in proceedings of 1997 ACM SIGMOD Conference, May 1997

[GW97] Roy Goldman, Jennifer Widom, *"DataGuides: Enabling query formulation and optimization in semistructured databases"*, Proceedings of the 23th international conference on VLDB, 1997

[GMW99] Roy Goldman, Jason McHugh, Jennifer Widom, *"From semistructured data to XML: Migrating the Lore data model and query language"*, Proc. of the WebDB workshop, Philadelphia, 1999

[GW98] Paul Gray, Hugh J. Watson *"Decision support in data warehouse"* Prentic Hall PTR Inc 1998 ISBN 0-13-796079-4

[In99] W .H. Inmon *"Building the operational data store"* John Wiley & Son Inc 1999 ISBN 0-471-32888-X

[JAXP] Sun Microsystems, http://java.sun.com/xml/xml_jaxp.html

[Ke98] Sean Kelly *"Data Warehousing-the route to mass customization"* John Wiley & sons 1998 ISBN 0-471-96328-3

[KKR00] G. Kappel, E. Kapsammer, W. Retschitzegger, *"Architectural Issues for Integrating XML and Relational Database Systems - The X-Ray Approach"* Accepted for publication at the workshop XML Technologies and Software Engineering (XSE2001) Toronto, Canada (co-located with ICSE2001), 15 May 2001

[KM00] Kanne, C.-C., Moerkotte, G, *"Efficient Storage of XML Data"*, Proc. Of the 16th Int. Conf. On Data Engineering (ICDE), San Diego, March, 2000

[Ko92] Philip Koopman, Jr. *"A preliminary exploration of optimized stack code generator"* the fourth conference, Rochester, 1992

[LC00] D. Lee and W. W. Chu. *Comparative Analysis of Six XML Schema Languages*". ACM SIGMOD Record, 29(3):76--87, Sep. 2000

[Liu01] Yi Liu, *"Code generator for integrating Warehouse data source"*, M.Sc. thesis, University of Windsor, 2001

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, J. Widom. *"Lore: A database management system for semistructured data"*, SIGMOD Record, 26(3), Sept. 1997

[MSXML] Microsoft Company, http://msdn.microsoft.com/xml/default.asp

[QRS+95] Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Jennifer Widom, "*Querying Semistructured Heterogeneous Information*", DOOD 1995: 319-344

[Re92] C. van Reeuwijk, "Tm: a code generator for recursive data structures", European Community ESPRIT project 881, 1992

[SAX00] SAX 2.0, http://www.megginson.com/SAX/sax.html, Megginson Technologies Ltd., 2000

[SRL00] Surjanto, B., Ritter, N., Loeser, H, " *XML Content Management based on Object-Relational Data-base*", Technology. Proc. Of the 1st Int. Conf. On Web Information Systems Engineering (WISE), Hongkong, June 2000

[SW00] Schöning, H., Wäsch, J., "*Tamino – An Internet Database System*", Proc. of the 7th Int. Conf. on Ex-tending Database Technology (EDBT), Springer, LNCS 1777, Konstanz, March, 2000

[SWA99] SAS Warehouse Administrator http://first.gmd.de/gccs/primer.html

[Th00] Henry S. Thompson, "*XML Schema Tutorial*",
www.oasis-open.org/cover/thompsonSchemaSlides19991220_files/frame.htm
HCRC Language Technology Grou, W3C, 2000

[XE00] Xu, Mei and Ezeife, C.I., "Maintaining Horizontally Partitioned Warehouse Views", *proceedings of the second international conference on Data Warehousing and Knowledge Discovery, DaWak'00*, Greenwich, U.K, DEXA conference, published in the Lecture Notes in Computer Science (LNCS) by Springer Verlag, Sept. 4-6, 2000.

[XML4J] IBM Company, http://www.alphaworks.ibm.com/tech/xml4j

Appendix A: User Manual

This chapter presents about how to use WIG4X code generator system based on a shopping system example.

Step 1: Planning DW Definition

Before running the programs, user should plan what kind of data warehouse will be created. Please notice that every data warehousing system table should follow data warehouse definition.

(1) Tables in data warehousing system: according to data warehouse definition, the target data warehouse system should have a fact table, metadata table, timetable and at least one dimension table.

(2) Attributes in each table: attributes in each table should comes from one or more tags in source XML documents. Not to create target attribute which can not get from source XML documents. WIG4X will always integrate the information which really exists in the source XML documents.

(3) Metadata for data warehouse tables: by default, the system will offer the data warehouse metadata by last update date, table names in the data warehouse and the total record in each tables. You can add more parts as you wish, for example, you can add the last new record information or something else. Because the data warehouse we are creating is based on the Oracle database, some metadata information already are offered by Oracle database management system, e.g., the attribute for each table and their data type. So in this system, we will not consider them. But you still can add them later.

(4) Data types of table attributes: The target table attributes has data types, since there is no data type restrictions in XML document, XML document is in text format. So, there is not too much restrictions in the data type definitions of target table attributes. Only when defining the "date" or "number" type for an attribute, user should be sure that the corresponding information comes from XML document and can be converted to the "date" or "number" format.

Step 2: Planning Integration Relation

(1) Planning structure integration: Mapping between each data warehouse attributes and the corresponding XML tags will be planned. Generally, for structure integration, there are two cases:

- One to one relation: one source XML tag is mapped to one target attribute, e.g., tag <OrderID> mapped to the target attribute "orderkey".
- Many to one relation: some source attributes are mapped to one target attribute, e.g., tag <fname> and <lname> can be mapped to target attribute "name".

(2) Planning measurement conversion: For example, mile for distance in source XML tag times 1.6 can be kilometer in the target database. You have to list all of the conversions accurately because if they keep in different unit after integration, the result in target database will not be accurate.

(3) Planning other transformation rules: for some attributes, its data can't be obtained from the source XML tag directly, for example: the "branch" attribute in fact table, it is a integration attribute, no XML tag can be mapped to it directly, so such integration attribute mapping should be planned. And some data that come from XML tags need to be transformed, such as "M" or "F" in <gender> tag, need to be transformed to "Male" or "Female".

(4) If you want to use this system by some pre-defined input files, you should have these files ready:

- Target data warehouse definition file: Input all the information by the order of ***table name, data type, length, key or non key*** in each line of the input text file, each line represents one attribute information for certain table. One example input file is shown in Figure A1.3.
- Mapping expression file: The information organized in the text should follow the order as "***DW table name, attribute, data type, XML file name, node path, node name, constraint***" in each line of this file, each line represents one mapping expression. For example, in shopping system, a mapping rule file is shown in Figure A2.2.

- If you want to input the XML views definition by files, you should also prepare a text file, the information in that text file should follow the order as "***view name, tag name, constraint***" in each line of that file, each line represents one data warehouse table attribute which will be selected in the certain view. For example, in shopping order system, the content of rule file is shown in Figure 3.2:

Step 3: Start WIG4X

After the above planning, your have are ready to start this data integration code generation system. Now, what you can do is to start this system. Because this system is the platform independent, you can run this system on any machine and operation system.

Under UNIX multi user system (SUN/Solaris, Unix/SGI etc): simply type ***Java WIG*** after %. Under PC/DOS or Window platform, after c> sign type ***WIG***

The code generation system will be started. It displays a dark green window with a welcome information. This window is an interface that tells user they have started the data warehouse data integration system. Click the button "***push me***", the system will go to the next window to choose data source.

In next window, four data sources are listed on the window: relational data source, XML data source, HTML data source and text data source. Choose the "***XML data source***", then click the button "*Next* ", then the system will go to the WIG4X system's DW table generation window.

Step 4: Table Generator

Table generation window is the window that accepts the user interesting query for what kind of the data warehouse they wish to create including the table name, table's attribute, data type and the primary key constraints.

There are two ways to input the table's information from the window: manual or input by file.

(1) By Manual:

As the figure A1.1, there are four text fields for inputting the interesting table information into Data Warehouse Table Generator window. Required table can be used to input the interesting table name including the fact table and all of the dimension tables. Attribute name is a text field that is used for inputting the attribute name for each table and attribute type length is used for inputting the attribute data type and length

Follow the window GUI, input required **table names, attribute names, length**, and **Key or Non Key** features, the click "***submit***" button, the input attributes information will be displayed on the right side of the display area. As in Figure A1.2.

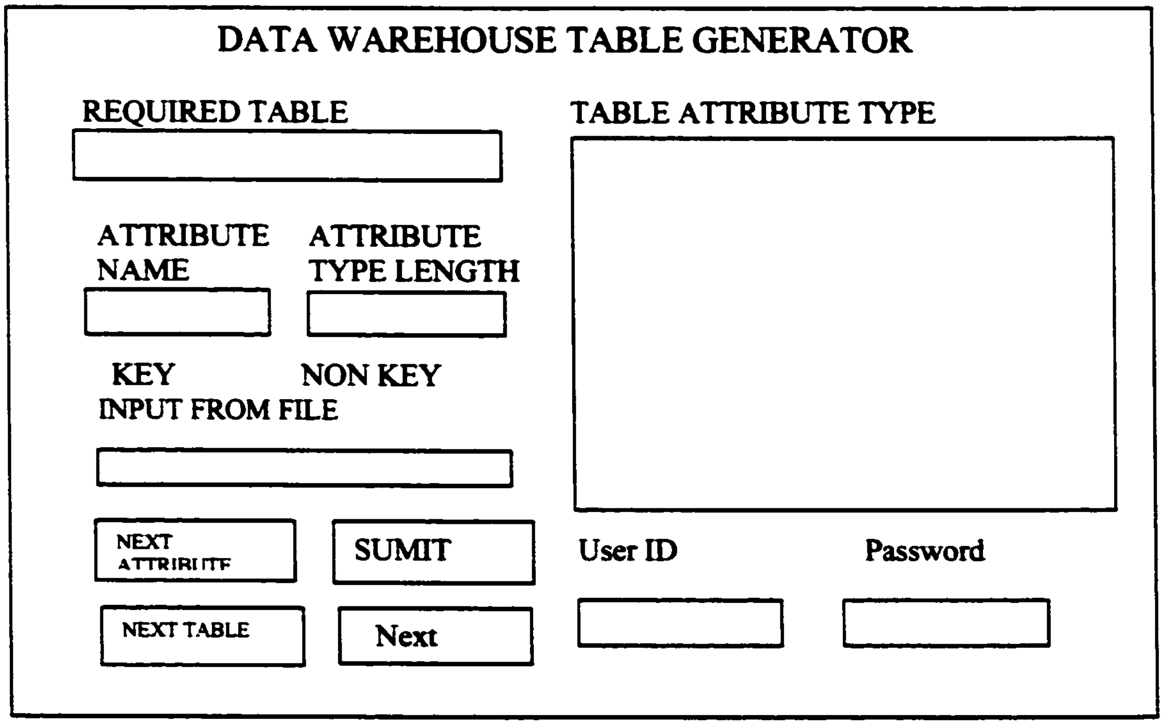
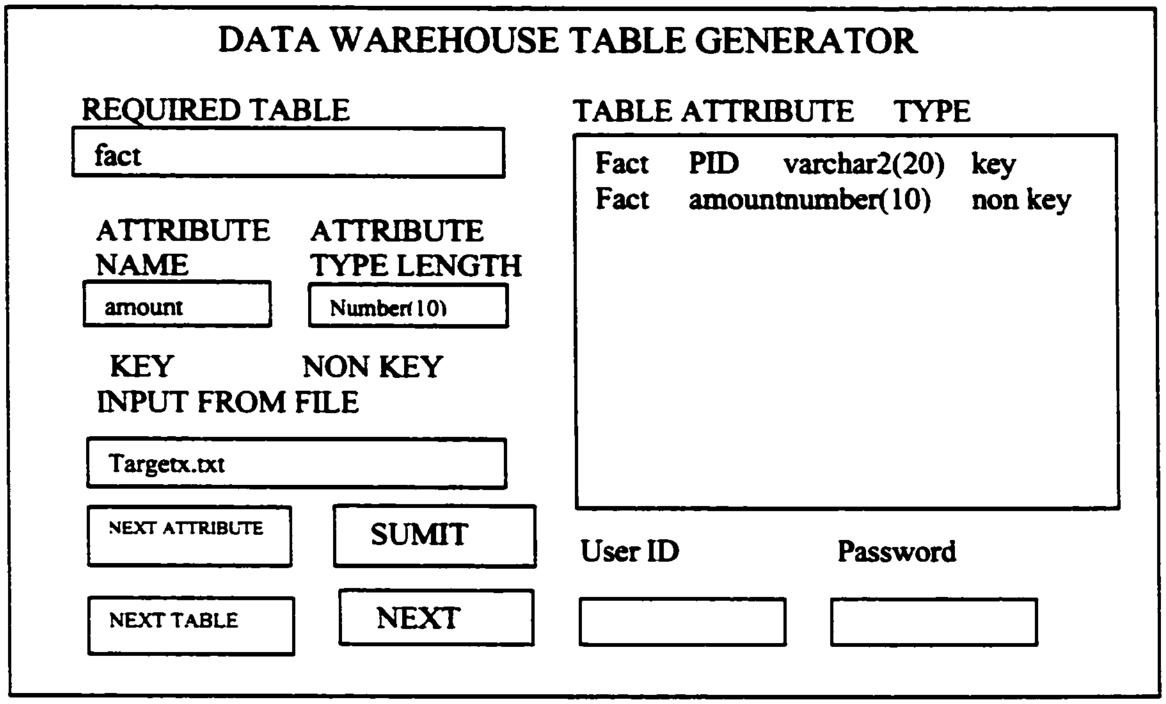


Figure A1.1: data warehouse table generator window



DATA WAREHOUSE TABLE GENERATOR

REQUIRED TABLE: fact

ATTRIBUTE NAME: amount

ATTRIBUTE TYPE LENGTH: Number(10)

KEY NON KEY

INPUT FROM FILE: Targetx.txt

TABLE ATTRIBUTE TYPE

```
Fact   PID    varchar2(20)   key
Fact   amountnumber(10)      non key
```

User ID

Password

Figure A1.2: Input data for DW tables

Input all table definition one by one and the database connection ***user id*** and ***password***, then all the DW information will be accepted by the system and shown in the GUI. Click "***Next***" to continue the system. If an application is more complex, it's better to input such information by file.

(2) Input from file:

In order to input the DW table information from file, first, we have to create a text file such as ***Targetx.txt*** by any editor available from personal computer of Unix platform. And input all the information by the order of ***table name, data type, length, key or non key*** in each line of the input text file, each line represents one attribute information for certain table. One example input file is shown in Figure A1.3.

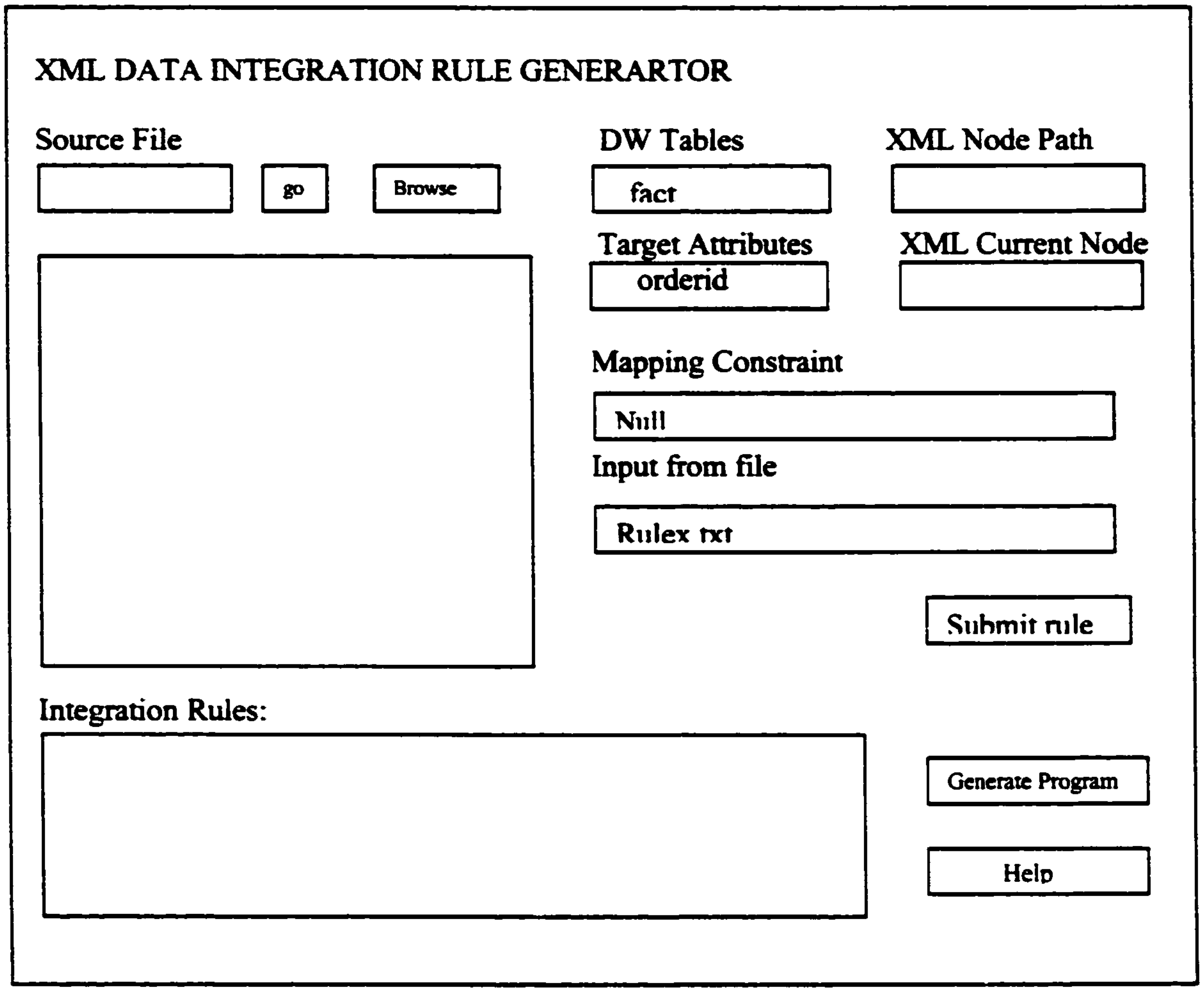
Simply input the file's name in the text field "***Input from file***" as Figure A1.2, and press "***submit***" button. The system first will check if the file name is right or not. For a wrong name, the system will ask to do it again and for a right name, the input table information will display on the right side of the window. For the shopping system, the DWS table definition can be as Figure A1.3.

```
w4xfact,orderid,varchar2(15),null
w4xfact,branch,varchar2(15),null
w4xfact,itemkey,varchar2(15),null
w4xfact,custkey,varchar2(15),null
w4xfact,orderdate,date,null
w4xfact,quantity,number(10),null
w4xfact,price,number(10),null
w4xcustomer,custkey,varchar(20),key
w4xcustomer,name,varchar(30),null
w4xcustomer,address,varchar(90),null
w4xcustomer,phone,varchar(20),null
w4xcustomer,gender,varchar(20),null
w4xitem,itemkey,varchar(20),key
w4xitem,name,varchar(30),null
w4xitem,brand,varchar(20),null
w4xitem,type,varchar(20),null
```

Figure A1.3: A DW definition input file for shopping system

After that, click the button "***Next***". Then a small window will pop up and let user determine the ***generated programs' names***. The programs include DW table creation, data extracting, data cleaning, data loading and XML view creating programs. The system provide users the default names, users can just simply accept them, or define the programs' names by themselves. And then the system will display a window called "XML DATA INTEGRATION RULE GENERARTOR" as the Figure A2.1 below.

Step 5 XML data integration rule generator


XML DATA INTEGRATION RULE GENERARTOR

Source File

DW Tables: fact

Target Attributes: orderid

XML Node Path

XML Current Node

Mapping Constraint: Null

Input from file: Rulex txt

Integration Rules:

Figure A2.1: XML integration rule generator window

This interface is used for generating data integration program. There are two ways to generate XML data warehouse integration rules, by manual or from file.

(1) By Manual

The user GUI is shown in Figure A2.1, when program goes to this GUI, the data warehouse table names and the corresponding attributes will be shown in the box "***DW***

Tables" and "***Target Attributes***". Each time user clicks the table name in the "DW Tables", the attributes for that table will be shown in the "Target Attributes".

User will input source file name in the text field "***Source file***" at first, that file should be ***XML schema*** or ***DTD file***, or ***XML file***. Having the source file name, click the button "***Go***", the program will go to find this file, extract XML file structure from this file, and show XML file structure as a tree in the scroll field below the source file name. If user wants to take a look at the input file, just click the button "***Browse***", then the input file will be displayed in a pop up window.

The tree is used to help user get the XML structure easily, user can click each node of the tree, if this node has child nodes or attributes, the tree will be extended and show its child nodes and attributes.

At the same time, the XML tag names will be shown in the "***XML Current Node***", and the XML path for the corresponding node will be shown in the "***XML Node Path***".

When user ***selects*** one DW attribute, the program will calculate a corresponding mapping XML tag, show it in the "XML Current Node". If user does not accept this mapping, he can choose another XML tag. If user accepts this mapping, and this mapping without constraint, just click the "***Submit rule***" button, the corresponding mapping rule will be shown in the text area "***Integration rules***".

If this mapping comes with constraint, the user can choose a mapping constraint from the list of "***Mapping Constraint***". When a certain constraint was chosen, a pop up small window will appear, and user can input specific constraint. For example, if user choose "Measurement integration rule" for the <usprice> tag, then user can input "*0.75" as the constraint for this mapping.

User can choose the mapping one by one, then all the integration rule mapping will be shown in the text area "Integration rules". After choosing all the mapping, user can click the "***Generate program***" button, then a phrase "Generating data integration programs, please wait..." will be shown, and the program will generate the data integration programs.

(2) Input From File:

In order to input rules from file, user should have a text file which contains the XML mapping rules, e.g. "***rulex.txt***". The information organized in the text should follow the order as "***DW table name, attribute, data type, XML file name, node path, node name, constraint***" in each line of this file, each line represents one mapping expression. For example, in shopping system, a mapping rule file is shown in Figure A2.2.

```
w4xfact,orderid,varchar2(15),USOrder.xml,usorders.usorder,ordernum,null
w4xfact,branch,varchar2(15),USOrder.xml,usorders,usorder,$$4:US
w4xfact,itemkey,varchar2(15),USOrder.xml,usorders.usorder.usitem,itemnum,null
w4xfact,custkey,varchar2(15),USOrder.xml,usorders.usorder.uscust,custkey,null
w4xfact,orderdate,date(0),USOrder.xml,usorders.usorder,orderdate,null
w4xfact,quantity,number(10),USOrder.xml,usorders.usorder.usitem,quantity,null
w4xfact,price,number(10),USOrder.xml,usorders.usorder.usitem,usprice,$$0:intevalue*0.75
w4xfact,orderid,varchar2(15),CAOrder.xml,caorders.caorder,orderid,null
w4xfact,branch,varchar2(15),CAOrder.xml,caorders,caorder,$$4:CA
w4xfact,itemkey,varchar2(15),CAOrder.xml,caorders.caorder.caitem,itemid,null
w4xfact,custkey,varchar2(15),CAOrder.xml,caorders.caorder.cacust,custid,null
w4xfact,orderdate,date(0),CAOrder.xml,caorders.caorder,orderdate,null
w4xfact,quantity,number(10),CAOrder.xml,caorders.caorder.caitem,quantity,null
w4xfact,price,number(10),CAOrder.xml,caorders.caorder.caitem,caprice,null
w4xcustomer,custkey,varchar(20),USOrder.xml,usorders.usorder.uscust,custkey,null
w4xcustomer,name,varchar(20),USOrder.xml,usorders.usorder.uscust,fname,$$1:fname+lname
w4xcustomer,address,varchar(20),USOrder.xml,usorders.usorder.uscust,address,null
w4xcustomer,phone,varchar(20),USOrder.xml,usorders.usorder.uscust,phone,null
w4xcustomer,gender,varchar(20),USOrder.xml,usorders.usorder.uscust,gender,$$2:M=Male;F=Female
w4xcustomer,custkey,varchar(20),CAOrder.xml,caorders.caorder.cacust,custid,null
w4xcustomer,name,varchar(20),CAOrder.xml,caorders.caorder.cacust,name,null
w4xcustomer,address,varchar(20),CAOrder.xml,caorders.caorder.cacust,street,$$1:street+city+postcode
w4xcustomer,phone,varchar(20),CAOrder.xml,caorders.caorder.cacust,phone,null
w4xcustomer,gender,varchar(20),CAOrder.xml,caorders.caorder.cacust,gender,null
```

Figure A2.2: Mapping rule text file for shopping system

Having the mapping rule text file, user can input the rule text file name in the text field called "***Input from file***", and then click the "***Submit rule***" button, all the mapping rules will be shown in the text area "***Integration rules***".

After choosing all the mappings, user can click the "***Generate program***" button, then a phrase "Generating data integration programs, please wait..." will be shown, and the program will generate the data integration programs. And after creating the codes for XML data integration, the "XML VIEW CREATING GENERATOR" window will appear.

Step 6: XML view generator

This interface is used to generate program for creating XML views. There are two ways to generate XML view program, by manual or input from file.

(1) By Manual

As shown in Figure 3.1, when user enters this GUI, if user does not want to create XML views, he can just click the button "***Skip***", then the program will skip this part and goes to next window.

If user wants to create some XML views from data warehouse, he can select the attributes which will be used in the XML views. At first, all data warehouse tables are listed in the box "***DW Tables***", and user can choose either one of them. When one table is chosen, all attributes of this table will be listed in the area "***Table attributes***", then user just highlights the attributes which he wants, click the "***Select***" button, then the chosen attributes will go to the area "***XML tags***". User can choose one or more attributes at the same time by pressing the "Shift" key in the keyboard.

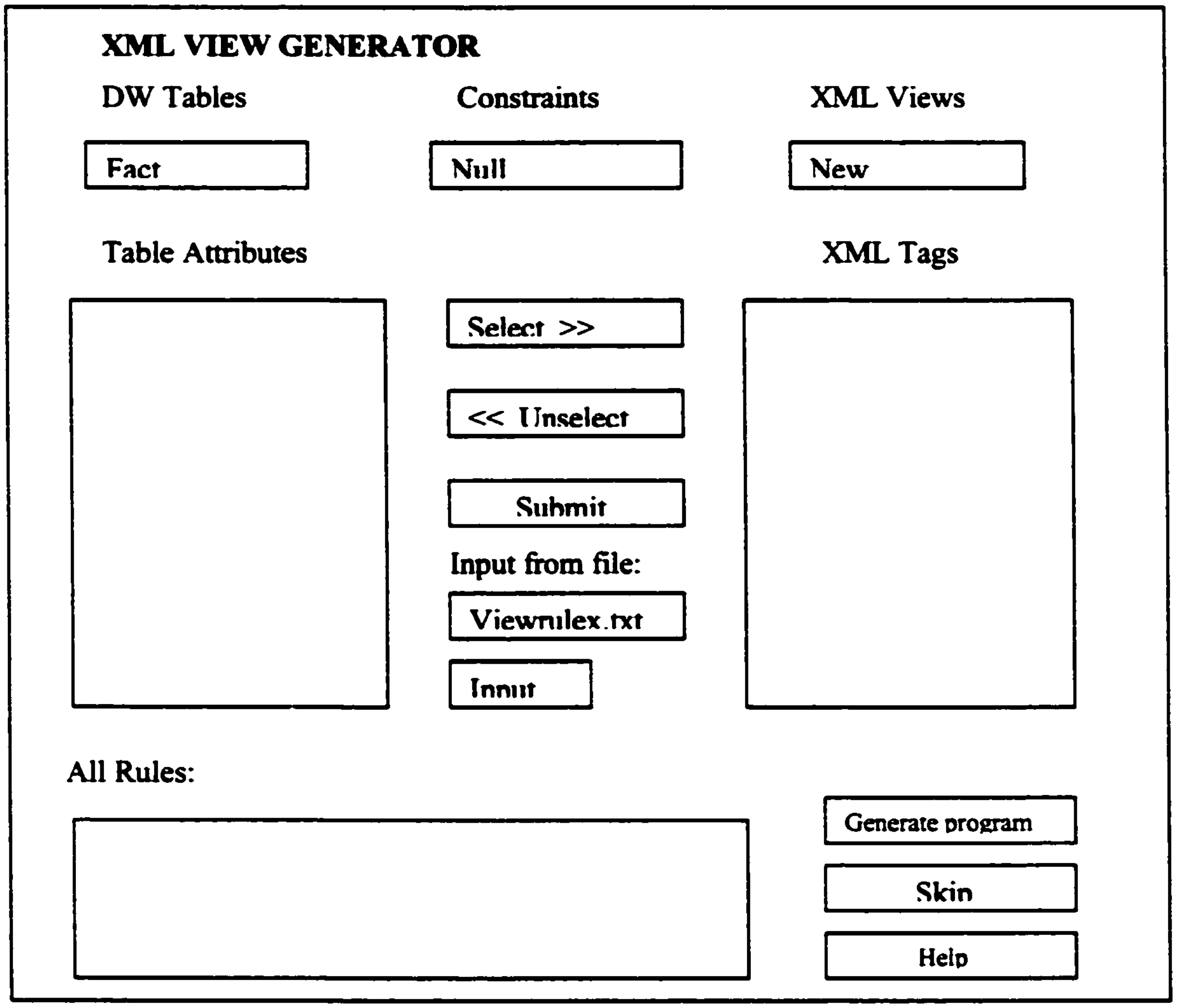


Figure 3.1: XML View Generator window

If there is constraint while selecting the certain attributes, user can choose the constraint from the box "***Constraints***". There are six kinds of constraints in this box: "group by", "sum", "average", "where", "nowhere", "null" and "other". User can choose either of them. For some constraints, when they are chosen, a pop up small window will appear, which ask user to input detailed information. For example, when choosing "where", user can input detailed information such as "where price > 75".

If user chooses the wrong attributes, he can undo that by highlighting the corresponding XML tags and clicking "***Unselect***" button, then the tag will be back to the table attributes. User can unselect one or more XML tags by pressing the "Shift" key in the keyboard.

After user selects all the tags for one XML view, he can click the "***Submit***" button, all the mapping rules for this XML view will be shown in the text area "***All Rules***". Then user can continue to select attributes for next XML view.

After all rules have been selected for that data warehouse, user can click the "***Generate program***" button, then system will begin to generate the codes for XML view creating.

(2) Input From File

In order to input XML mapping rules from file, user should have a text file which contains the XML view mapping rules, e.g. "***viewrulex.txt***". The information in that text file should follow the order as "***view name, tag name, constraint***" in each line of that file, each line represents one data warehouse table attribute which will be selected in the certain view. For example, in shopping order system, the content of rule file is shown in Figure 3.2:

```
w4xfact001,orderdate,nowhere orderdate>'30-apr-2001' and orderdate<'1-jun-2001'
w4xfact001,itemkey,group by
w4xfact001,quantity,sum(quantity)
w4xfact001,price,sum(price*quantity)
w4xfact002,orderdate,nowhere orderdate>'30-mar-2001' and orderdate<'1-jun-2001'
w4xfact002,branch,group by
w4xfact002,quantity,sum(quantity)
w4xfact002,price,sum(price*quantity)
```

Figure 3.2: XML view mapping rule file for shopping system

User can input the rule file name in the text field "***Input from file***", then click the "***Input***" button, then the program will go to find this file, and extract mapping rule information from this file, and shows all the mapping rules in the text area "***All Rules***".

After all rules have been selected for that data warehouse, user can click the "***Generate program***" button, then system will begin to generate the codes for XML view creating. After generating the XML view creating program, the Execution/display window appears.

Step 7 Execution/Display Window

The Execution/display window is used for compiling and running the target program to integrate source XML data into data warehouse and create XML views from the data warehouse tables. The GUI is shown in Figure A4.1.

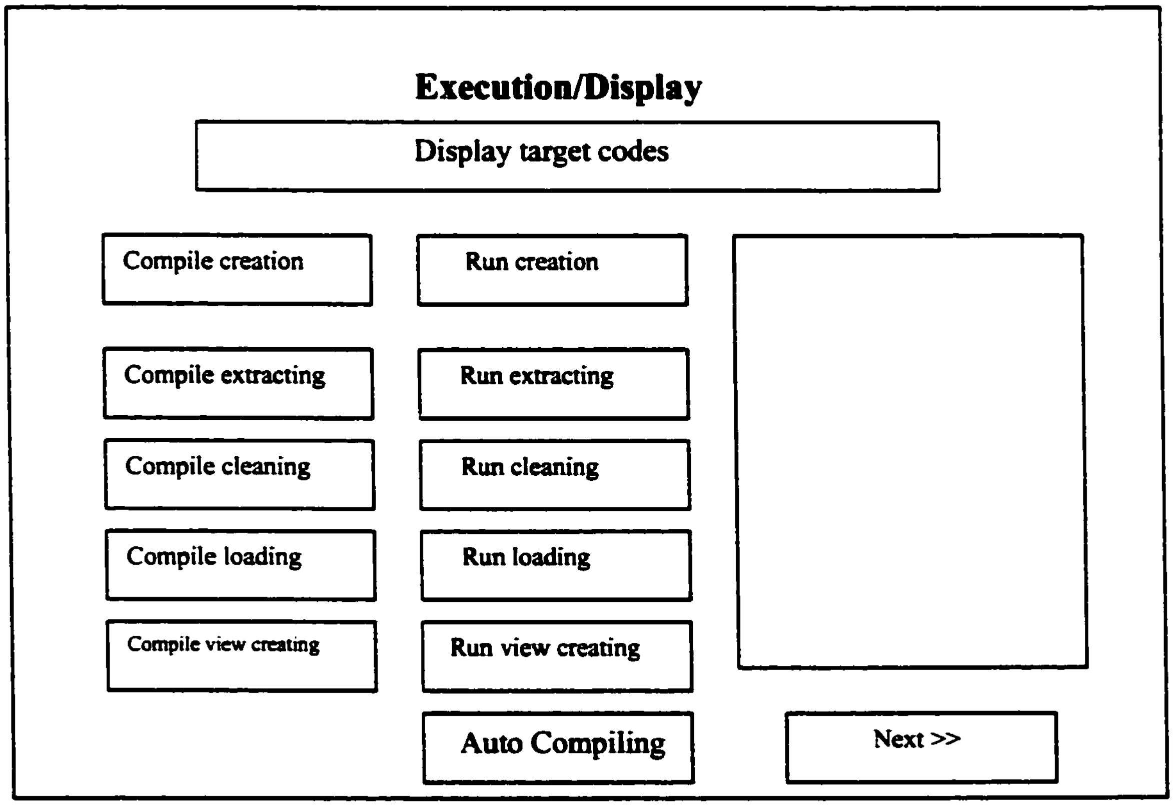


Figure A4.1: Execution/Display GUI

As shown in Figure A4.1, user can click the "***Display target code***" button to display the generated codes of WIG4X system. User can clicks the ***compiling buttons*** to compile each generated programs. User can also clicks the "Auto Compiling" button to compile all the generated programs automatically.

After user click the "*Next >>*" button, the system will goes to the ENG WIG4X window, which indicates that the WIG4X system is ending and the data warehouse has been created successfully. Click the "***END WIG***" button, the system will be end.

System Installation

In order to install WIG4X system, you need to have following software installed on the machine:

(1) Java JDK (version 1.3 or higher)
    Java JDK can be downloaded from java.sun.com website.
(2) XML API (Sun JAXP 1.1)
    Sun JAXP can be downloaded from java.sun.com website .
(3) Database (Oracle 8 or higher version)
    Oracle database can be downloaded from www.oracle.com website.

You should also set up your operating system environment for these above software. Suppose you install Java JDK under directory "<jdk1.3.1>", Sun JAXP under directory "<jaxp>".

For Window OS, you should add following statements in your "autoexec.bat" file:

```
PATH <jaxp>]\bin;%PATH%
set JAXPHOME=<jaxp>
set JAVA_HOME=<jdk1.3.1>
set CLASSPATH=.;<jaxp>\crimson.jar;<jaxp>\xalan.jar ; . ; %CLASSPATH%
```

For Unix OS, for example, Solaris, you should add following statements in your .cshrc file:

```
Set env PATH = <jdk1.3.1>/bin: ${PATH}
Set env CLASSPATH ${CLASSPATH}: . : <jaxp>/crimson.jar: <jaxp>/xalan.jar
```

Having installing the above backbones, follow the steps below:

(1) Suppose a home directory for your WIG4X system, say d:\wig4x.
(2) Copy a compressed file called "wig4x.zipz' in that directory.
(3) Uncompress that file, using command: "unzip wig4x.zip" for windows OS, "unzip ./wig4x.zip" for Unix OS.
(4) After compression, all WIG4X software programs will be put in that directory, a directory called "com" is also put in that directory, which contains the parse for XML DTD file.
(5) Make a directory for the generated codes, say d:\wig4x\generated, make a directory called "views" under the generated directory, which will contains the generated XML views.

The WIG4X system home directory structure is like following:

```
—Wig4x
     |
     |____Com
     |
     |____Generated
     |         |
     |         |______Views
     |         |
```

If all the previous steps run successfully, the system is ready to run, just go to the WIG4X home directory, type "wig", and the system starts.

VITA AUCTORIS

NAME	Chunshenng Liu
YEAR OF BIRTH	1970
PLACE OF BIRTH	Changchun, P. R. China
EDUCATION	M.Sc., Computer Science University of Windsor Windsor, Ontario Candada 2000 – 2001
	B.Sc., Computer Science Jilin University Changchun China 1988 -- 1992