DIVA-DATA WAREHOUSE INTERFACE FOR VISUAL ANALYSIS

By

PRAKASH BULUSU

A THESIS PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2003
This document is dedicated to my wonderful parents.
ACKNOWLEDGMENTS

I thank Dr. Joachim Hammer, the chairman of my thesis committee, for his guidance throughout the project. I also thank Dr. Joseph N. Wilson and Dr. Paul Fishwick for serving on my thesis committee. I thank Ms. Hongyan Zhao for successfully completing her part in this project.

I would also like to thank Mr. Jagdish Kumar, Mr. Kunal Vijaykar, and Mr. Suchindra Katageri for their encouraging support during this endeavor.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>ACKNOWLEDGMENTS</th>
<th>iv</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>Background and Motivation</td>
<td>1</td>
</tr>
<tr>
<td>System Overview</td>
<td>3</td>
</tr>
<tr>
<td>Thesis Organization</td>
<td>4</td>
</tr>
<tr>
<td>2 UNDERLYING TECHNOLOGIES</td>
<td>5</td>
</tr>
<tr>
<td>Data Warehouse</td>
<td>5</td>
</tr>
<tr>
<td>OLAP</td>
<td>7</td>
</tr>
<tr>
<td>Concept Hierarchy</td>
<td>8</td>
</tr>
<tr>
<td>Multidimensional data cube</td>
<td>8</td>
</tr>
<tr>
<td>OLAP Operations</td>
<td>10</td>
</tr>
<tr>
<td>Presentation Technologies</td>
<td>12</td>
</tr>
<tr>
<td>Virtual Reality Modeling Language (VRML)</td>
<td>12</td>
</tr>
<tr>
<td>External Authoring Interface (EAI)</td>
<td>14</td>
</tr>
<tr>
<td>Java Related Technologies</td>
<td>14</td>
</tr>
<tr>
<td>Java Servlets</td>
<td>15</td>
</tr>
<tr>
<td>Java Server Pages</td>
<td>16</td>
</tr>
<tr>
<td>XSLT</td>
<td>16</td>
</tr>
<tr>
<td>3 OVERVIEW OF ARCHITECTURE</td>
<td>18</td>
</tr>
<tr>
<td>Description</td>
<td>19</td>
</tr>
<tr>
<td>Client Side Module</td>
<td>19</td>
</tr>
<tr>
<td>VRML Enabled Browser</td>
<td>19</td>
</tr>
<tr>
<td>EAI + Applet VRML Engine</td>
<td>20</td>
</tr>
<tr>
<td>Relational Data Generator</td>
<td>20</td>
</tr>
<tr>
<td>Server Side Module</td>
<td>21</td>
</tr>
<tr>
<td>Meta Info Server</td>
<td>21</td>
</tr>
<tr>
<td>Main Engine</td>
<td>21</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Star schema of a data warehouse for sales</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Snowflake schema of a data warehouse</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>A concept hierarchy of the dimension location. Due to space limitation, not all the nodes of the concept hierarchy are shown.</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>A 3-D data cube representation of a sample data cube</td>
<td>9</td>
</tr>
<tr>
<td>2.5</td>
<td>Lattice of cuboids, making up a 3-D data cube</td>
<td>10</td>
</tr>
<tr>
<td>2.6</td>
<td>Examples of typical OLAP operations on multidimensional data</td>
<td>11</td>
</tr>
<tr>
<td>2.7</td>
<td>A sample VRML file</td>
<td>13</td>
</tr>
<tr>
<td>2.8</td>
<td>An example of a VRML tree</td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>Conceptual architecture of DIVA</td>
<td>18</td>
</tr>
<tr>
<td>4.1</td>
<td>Car Sales Star Schema used for testing in DIVA</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>Meta Info schema</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>A sample XSL stylesheet used to transform the input XML to SQL</td>
<td>29</td>
</tr>
<tr>
<td>4.4</td>
<td>Sample query for 3-D scene creation</td>
<td>30</td>
</tr>
<tr>
<td>4.5</td>
<td>Sample query for 3-D scene creation with lower concept hierarchies</td>
<td>31</td>
</tr>
<tr>
<td>4.6</td>
<td>Sample query for 3-D scene creation with lower concept hierarchies and drilldown</td>
<td>32</td>
</tr>
<tr>
<td>4.7</td>
<td>Scene depicting the construction of a data cube along the Z- axis</td>
<td>34</td>
</tr>
<tr>
<td>4.8</td>
<td>Scene depicting the construction of a data cube along the Y- axis</td>
<td>34</td>
</tr>
<tr>
<td>4.9</td>
<td>Rendering of nodes in the VRML browser</td>
<td>37</td>
</tr>
<tr>
<td>5.1</td>
<td>Starting page for DIVA</td>
<td>38</td>
</tr>
</tbody>
</table>
5.2 Data cube setup page. ..................................................................................................39
5.3 Depicting a user choosing an initial concept hierarchy level for each dimension......40
5.4 Data cube after initial selection of hierarchies, dimensions, and other operational parameters per Figure 5.2 and Figure 5.3.................................................................41
5.5 Slice operation. ............................................................................................................42
5.6 Dice operation..............................................................................................................43
5.7 Roll up operation..........................................................................................................44
5.8 Drilldown operation.......................................................................................................45
Multidimensional analysis of consolidated enterprise data is a major component of decision support systems. Online analytical processing (OLAP) enables such analysis on summarized data. Data warehouses employ OLAP tools, which view data in the form of a data cube. This data cube represents dimensions and facts for a typical industrial process.

The problem with some existing information visualization tools for OLAP is that they present data in a 2-D tabular format. Technologies like VRML have opened new possibilities for visualization of large amounts of information in a new way. This research is focused on exploring a fusion among VRML, data warehousing, and Java to create new intuitive tools to navigate, analyze and query the warehouse contents.

In this thesis we present an approach that utilizes 3-D features of VRML to visualize an OLAP cube. The architecture associated with this prototype is DBMS independent and will allow visualization of any data warehouse with minimal pre-installation efforts. We have designed an easy-to-use graphical interface with
visualization cues in which the user can perform various OLAP operations such as slice, dice, roll up and drill down. The design abstracts data from most parts of the project, hence making the application very extensible and modular.
CHAPTER 1
INTRODUCTION

This chapter describes the motivation and background behind this thesis and is followed by an overview of our system, Data warehouse Interface for Visual Analysis (DIVA). It concludes with an outline of the thesis.

Background and Motivation

Information management in an integrated, credible, and timely manner is highly important. Evolution of the Internet and global economies has seen extremely large amounts of data being stored and transferred on a regular basis. This data can be operational or strategic. Operational data is the data currently being used by the transaction processing systems. Strategic data is gathered by decision support systems to answer our strategic requirements. Organizations have lately begun to recognize the need for strategic data to serve as the basis for their decision support systems. The current situation is very similar to what the authors claim [1:1] “We are drowning in data, but starving for knowledge.” Thus, knowledge extraction has become a field of great interest.

Tools based on data warehouses, data mining, and data visualization facilitate knowledge extraction. A data warehouse is a subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management’s decision-making process [1]. There are compelling reasons to separate data warehouses from operational databases. A major reason is that both have different design goals and performance issues. There are numerous advantages in using a data warehouse:

- Immediate information delivery.
- Data integration across heterogeneous sources.
- Concurrent processing of multiple transactions is not required.
- Extrapolation of future trends based on history.
- Provides tools for looking at the data in new ways.

As data warehouses grow bigger and more powerful there is a greater requirement for tools, which provide visualization of data [2]. Data visualization tools increase the magnitude of information that comes out of information processing systems. These visualization tools are based on data analysis techniques like OLAP [1]. Online analytical processing (OLAP) deals with tools and techniques for data analysis that can give answers to many queries requesting summarized data. The basic SQL aggregation and grouping facilities are not enough for complex data analysis. Examples include finding aggregates over sliding windows on sequentially ordered data [3]. Data visualization tools require cross-domain inputs from database and 3-D research groups.

Several proposals had been made on the convergence of Virtual Reality (VR) and databases, resulting in an increase in the amount of information that users can perceive. The Xerox PARC User Interface Research Group has conducted extensive research in this field. This group focuses on visualization of hierarchical information by introducing perspective walls and cone trees [4]. A large-scale data visualization application is New York Stock Exchange’s 3-D Trading Floor, which uses 3-D graphics to monitor all business activities [5]. Libvrmldb is an open source implementation of an interface to create VRML worlds for a relation [6]. Noser and Stucki [7] discuss embedded SQL extensions for VRML to provide access and interaction with a 3-D spatial database.

Benson and Smith [2] discuss various OLAP tools currently available in the industry like Alta Analytics, IBM’s Visual Data Explorer etc. but these are mainly data mining tools. Arbor Essbase Web Gateway [8] is an interactive interface for viewing
multidimensional data from a Web browser using HTML templates. There are products similar to Essbase like Information Advantage WebOLAP, Brio.Web.warehouse by Brio Technologies etc. but these products do not use the power of Virtual Reality to visualize data. Most of the available commercial tools use proprietary technologies and are not generic enough. Our solution is to use a client server model over WWW that is based on:

- Java Server pages for dynamic page generation.
- JDBC for database connectivity and manipulation.
- XML based Meta information management.
- Plug and Play architecture.
- Easy navigation through External Authoring Interface (EAI) for VRML.

**System Overview**

The visualization and visual analysis tool DIVA, described in this thesis is a prototype system to allow an easy and interactive presentation of large amounts of data in a multidimensional format. DIVA also provides an extensible application interface to other knowledge extraction systems. Our goal is to prove the concept of using Virtual Reality and OLAP techniques, to allow interactive interpretation of vast amount of data.

DIVA is a Web-based tool that can be deployed as a data warehouse extension. The data is then visualized as a 3-D data cube, where each of the dimensions is represented along the edges of a cube. The user can then choose various aggregate operations that can be performed on summarized data. These operations are designed to be intuitive and straightforward. All the operations defined by DIVA can be performed with few clicks on the buttons provided in the user interface. The relational queries generated to facilitate the 3-D scene creation are transparent to the user. DIVA allows the user to input Meta information (information about the data warehouse) in XML format. The user can even change the look and feel of the 3-D scene by changing configuration files at application
DIVA uses standard technologies like VRML, XSLT etc. to provide scalability and wider system support. DIVA uses color as a visual cue to encode the result of OLAP operations to reveal more information [9]. The user is also provided with an option of viewing relational data (corresponding to the VRML scene) at all operational stages. DIVA makes all attempts to reduce resource consumption on the server as well as the client side by using mechanisms like *connection pooling* etc.

This tool can be used for various practical purposes like teaching tool, stock tracker, student performance analyzer etc. These cases are separately discussed in Chapter 6.

This project is a collaborative effort with another student, Ms. Hongyan Zhao. Hongyan is responsible for the client side of the system that allows the user to interact with the 3-D cube and perform slice, dice, and rollup operations [9].

**Thesis Organization**

This thesis consists of six chapters. Chapter 2 presents a brief description about the technologies that are necessary to understand this project. In Chapter 3 we discuss the architectural aspects of this tool. Chapter 4 places an emphasis on the implementation details, focusing on how the technologies described in Chapter 3 were applied to the problems presented earlier. Chapter 5 provides an operational description of the tool. In Chapter 6 we discuss the evaluation of this system and summarize our research results.
CHAPTER 2
UNDERLYING TECHNOLOGIES

This chapter provides an overview of the technologies used in this research project. This includes an overview of data warehousing, OLAP, VRML and Java related technologies.

Data Warehouse

Data warehouses and OLAP tools are based on multidimensional data model [1,3]. This data model views data in form of a *data cube* [1], which is a generalization of a two-dimensional cross-tab (summarized information for the dimension list) to n dimensions. Dimensions are the global contexts with respect to which a business concept can be measured. Facts are quantitative aspects or measures to analyze the relationships between dimensions.

There are many ways in which we can model a data warehouse using the relational data model. In this subsection we will discuss two such approaches. In both these approaches, dimensions have a table associated with them called *dimension table*. Facts are associated with one or more *fact tables*. A theme is generally used to describe a fact table.

The *star schema* model contains a fact table in the middle, connected to a set of dimension tables. The fact table has no redundancy and forms the center of the radial pattern formed by the dimension tables and hence the name star schema.
Figure 2.1 Star schema of a data warehouse for sales [10].

Figure 2.1 depicts an example of a star schema. In the star schema only one table, that has a few intentional redundancies, represents each dimension. These redundancies improve the query performance by reducing the number of joins, required among different tables.

The snowflake schema is a refinement over star schema, where the constraint on dimension table is removed. In the snowflake schema, dimension tables can be normalized into a set of smaller dimension tables. Using normalized dimension tables can optimize maintenance and storage space. However in data warehouses the primary concern is query optimization and effective browsing, not storage space. Hence, the snowflake schema is less popular compared to the star schema (which provides more efficient data browsing).

Figure 2.2 shows an example of a snowflake schema. The main difference between the two schemas is the definition of dimension tables. The single dimension tables for item and supplier tables are normalized, resulting in two new tables supplier and city.
OLAP

Multidimensionality described in the previous subsection, is at the core of an OLAP system. In 1993, Dr. E. F. Codd, in his paper entitled “Providing On-Line Analytical Processing to User Analysts,” formalized the term OLAP. In this paper, Dr. Codd proposes 12 guidelines for OLAP tools (cited in Gray and Watson [11]):

1. Multidimensional view
2. Transparent to the user
3. Accessible
4. Consistent reporting performance
5. Client server architecture
6. Generic dimensionality
7. Dynamic sparse matrix handling
8. Multi-user support
9. Cross-dimensional operations
10. Intuitive data manipulation
11. Flexible reporting
12. Unlimited dimensions aggregation

In plain words OLAP systems allow a multidimensional view of data and facilitate interactive query, summarized comparisons, and complex analysis. We will discuss how DIVA adheres to these guidelines in Chapter 6.
Concept Hierarchy

We need to define the term “concept hierarchy” before we can proceed to various OLAP operations in the next subsection. A concept hierarchy defines a sequence of mappings from a set of low-level concepts to higher-level and more general concepts. For example consider a concept hierarchy for a dimension location consisting of levels called city, state, and country. State level values for location include Florida, New York, Illinois. Each city level value can then be mapped to the corresponding states. Generalizing this, we can say that in a concept hierarchy a lower level concept can be mapped to a higher-level (and more general) concept. Figure 2.5 depicts one such concept hierarchy, the topmost level “All” represents the highest level of generalization.

![Figure 2.3 A concept hierarchy of the dimension location. Due to space limitation, not all the nodes of the concept hierarchy are shown.](image)

Multidimensional data cube

A data warehouse views data in the form a data cube, introduced earlier in this section. A data cube allows data to be modeled and viewed in different dimensions. Figure 2.4 depicts a sample data cube for the schema shown in Figure 2.1 and Figure 2.2.
Please note that we have shown only three dimensions (the branch dimension is not shown) in this figure. The dimensions in this figure are Product, Location and Time. The theme and measure used here are sales and dollars_sold respectively. The Time dimension is at Quarter level and the Location dimension is at Country level. The Item dimension has no concept hierarchies defined in the schema.

Figure 2.4 A 3-D data cube representation of a sample data cube.

We call an n-D base cube a **base cuboid**, the top most 0-D cuboid an **apex cuboid** and the lattice of cuboids forms a **data cube**. Figure 2.5 depicts one such lattice of cuboids for the same dimensions shown in Figure 2.4.
OLAP Operations

In OLAP applications, data is organized in a multidimensional format and each dimension contains several levels of abstractions. These levels of abstractions are defined by concept hierarchies and provide different perspectives to the user for data analysis.

OLAP applications provide a set of useful operations including dice, slice, roll up, drill down, etc. We first explain these operations in this chapter and then use these definitions to describe the functionality of our tool.

Each of the operation described below is illustrated in Figure 2.6. At the center of the figure is a data cube for AllElectronic sales. The cube contains the dimensions location (aggregated with respect to city values), time (aggregated with respect to quarters), and item (aggregated with respect to item types). The measure depicted is dollars-sold (in thousands).

- Roll-up: The roll-up operation means rolling up to higher hierarchical levels of aggregation either by climbing up a concept hierarchy for a dimension or by dimension reduction. Figure 2.6 depicts a roll-up on dimension location from city to country (climbing up a concept hierarchy).
Figure 2.6 Examples of typical OLAP operations on multidimensional data.

- **Drill-down**: Drill-down is the reverse of roll-up. It allows the user to drill down to the lower levels of detail either by stepping down a concept hierarchy for a dimension or introducing additional dimension. Figure 2.6 depicts the result of a drill-down by stepping down a concept hierarchy for dimension time (from quarter to month).

- **Slice**: The slice operation is selection by restriction on one dimension. Figure 2.6 shows a slice operation where data is restricted by the criterion time=Q1.
• Dice: The dice operation is selection by restriction on two or more dimensions. Figure 2.6 shows a dice operation where data is restricted by criteria that involves three dimensions.

OLAP offers many other operations not mentioned above and thus provides the ability to model and analyze real world data with business metrics and dimensions. The reader is referred to [12] for detailed description of existing OLAP operations.

**Presentation Technologies**

The OLAP operations that DIVA supports can be categorized into client side operations and server side operations. Client side operations such as slice or dice do not require any additional data from the server whereas server side operations like drill-down require a server request.

All the OLAP operations that are supported by DIVA need a 3-D modeling or scene description language. Furthermore, client side operations require a technology that can provide interaction with these 3-D scenes. This section describes the presentation technologies used in DIVA to address the requirements mentioned above.

**Virtual Reality Modeling Language (VRML)**

VRML is a text based file format for describing 3-D scenes and objects. VRML was revised in 1997 to VRML 2.0, which changed the file structure and added a lot of new features. VRML 2.0 became an ISO standard in 1997. VRML is now capable of supporting a broad range of applications, which require static and dynamic 3-D, and multimedia objects.

VRML files have four main components [13,14] as shown in Figure 2.7:

• The VRML header: indicates that this is a VRML file (line 1 in Figure 2.7).
• Nodes: describe shapes and their properties (lines 7 to 11 in Figure 2.7).
• Comments: include extra information (lines 3 to 5 in Figure 2.7).
• Fields: define the attributes of a node (lines 8 and 9 in Figure 2.7).
VRML files can be viewed in a browser using a plug-in like Cortona or Cosmo Player. VRML files can be created using any text editor, but it is recommended to use VRML specific editors such as VrmlPad [15]. VRML files have the extension .wrl.

```plaintext
#VRML V2.0 utf8
#The above line is VRML header
#This is a comment—Prakash Bulusu
#Below are few nodes.
WorldInfo {
   title "DIUA Thesis Example"
   info "Copyright Prakash Bulusu"
   "DIUA"
}

DEF FBOX Shape {
   appearance Appearance {
      material Material {
      }
      geometry Box {
      }
   }
}
```

Figure 2.7 A sample VRML file.

A VRML file is a collection of nodes arranged in a particular order. The node is the building block of a VRML tree. Individual nodes can describe features like shapes, colors, lights etc. [13]. These nodes form a hierarchical tree as shown in Figure 2.8.

Figure 2.8 An example of a VRML tree.
The VRML browser reads the tree from top to bottom and from left to right. The three main classes of nodes that provide user interaction are sensors, interpolators, and scripts. For instance, a touch sensor for a specified object generates an event, when a user clicks on the object. The sensor node is often routed to a Script node that runs a little Java program when the sensor generates an event.

There are some alternatives to VRML like X3D and Java3D. X3D is a next-generation open standard for 3-D on the Web [16]. X3D is an XML-encoded language and addresses limitations of VRML. However, X3D is not a mature standard yet and has very little support and few development tools available. Java3D is a low level 3-D scene-graph based, graphics programming API for the Java language [17]. Java3D provides a platform independent support for Web based interactive environments. However, Java3D is not as popular as VRML.

**External Authoring Interface (EAI)**

The EAI allows the user to control the contents of a VRML browser window embedded in a Web page from a Java applet on the same page [18]. There are two methods to use Java with VRML, either through EAI or a Script node. The EAI offers a generalized method to access the nodes and events in a VRML scene and lets the user, trigger the events on a dynamic basis. The Script node consists of predefined fields and events. We have used EAI for implementing the client part of the project. A more detailed description of how to use EAI with applets is described in Hongyan [9].

**Java Related Technologies**

Java was introduced by Sun Microsystems in 1991. Java was originally intended for use in small, embedded devices but businesses were quick to recognize Java’s potential on the server. Java is well known for its cross-platform portability and it is
inherently suited for large client-server applications. Java is directly derived from C++ but does not have the confusing aspects (of C++) like pointers, multiple inheritance, operator overloading and explicit memory management. It is a true object oriented language with several utility classes like hashtables and data structure management.

As described in the previous subsection we used VRML and EAI as the front-end for DIVA. Java provides an implementation for EAI specification. There is a VRML97 development library called CyberVRML97 [19] for Java programmers. This library makes dynamic VRML scene creation very easy. DIVA is a Web-based application and the server side is written using two important technologies: Java Servlets and Java Server Pages (JSP).

**Java Servlets**

Java Servlets are server extensions that can be loaded dynamically. They are better alternatives for CGI scripts on a Web server. Unlike CGI, Java Servlets do not spawn multiple processes to handle separate requests and are handled by separate threads in the same server process. This makes Servlets scalable and efficient. Another advantage of Servlets is the portability that comes with Java. A Java Servlet runs inside a sandbox and hence it is safe and portable. Servlet code is strikingly elegant, object oriented, tightly integrated with the server, and extensible and safe.

Another popular server-side programming language is Microsoft Active Server Pages (ASP). ASP is proprietary and is not supported on Web servers accept Microsoft IIS server. ASP is written in VBScript and is not object oriented. We discuss JSP as a better alternative to ASP in the next subsection.
Java Server Pages

The concept of Java Server Pages (JSP) is similar to ASP. The main objective of JSP was to separate presentation and business logic. JSP allows for the direct insertion of servlet code into HTML. Unlike servlets, no compilation is required for JSP. A JSP file is translated into a servlet the first time it is invoked by a client.

JSP allow Java Beans to be included directly in the code. Java Beans are reusable Java classes whose methods and variables follow specific naming conventions to give them added abilities [20]. A server can automatically set the bean’s properties using the parameter values in the incoming request. Thus a Java Bean is an elegant way to write separate modules like data access module or VRML generation module.

Java Database Connectivity (JDBC) is a standard database access method designed specifically for Java programs. It is a Java API that enables Java programs to execute SQL queries and interact with any DBMS supporting a JDBC driver. Thus, JDBC makes it possible to write a DBMS independent database application.

Java Servlets and JSP are powerful server-side programming technologies. However, care should be taken while choosing between these two. Servlets are especially appropriate for applications or parts of application that do not require a user interface. JSP on the other hand, should be used when the pages involve presenting HTML in response to a user request.

XSLT

Extensible Stylesheet Language Transformations (XSLT) is part of a larger specification called Extensible Stylesheet Language (XSL) [21]. XSL allows users to write transformations from XML to other formats. An XSL program is a set of template rules. To extract data from an XML document one can write a program in various
languages like Java or C++ that works with XML parsers, but it is not so straightforward. XSLT requires no programming but a specification of rules. Each rule consists of a template and a pattern. XSL starts from the root node and tries to apply the pattern to that node. If it succeeds, it executes the corresponding template. We use XML related technologies to generate Meta information about the data warehouse.
CHAPTER 3
OVERVIEW OF ARCHITECTURE

This chapter provides an overview of the system including the functionality of main modules. The architecture is depicted in Figure 3.1.
Description

The system can be identified in three subsystems:

- The Client Side Module providing the user interaction (indicated as A in the Figure 3.1).
- The Server Side Module as the application server (indicated as B and D in the Figure 3.1).
- The data warehouse storing all the required data (indicated as C in the Figure 3.1).

This design is adapted to give the system scalability and interoperability. The three subsystems mentioned above will be discussed further in the following sections. As indicated in Chapter 1, this project is a collaborative effort. The subsystems developed by Hongyan are described in more detail in Hongyan [9]. We will focus more on the server side modules, namely Meta Info Server, VRML View Handler and, the monitor subsystems in this section. For more information on the Client Side Module please refer to Hongyan [9].

Client Side Module

The Client Side Module provides the user interaction to our system and interacts with server. This module consists of three main components as shown in Figure 3.1, namely VRML Enabled Browser, EAI + Applet VRML Engine, and Relational Data Generator. There is also a Help Generator component written in JavaScript.

VRML Enabled Browser

The VRML Enabled Browser displays the entire frameset of HTML and VRML scenes. As mentioned earlier, VRML requires browser plugin to display the 3-D world. There are a number of freely downloadable VRML browser plugins like Cosmo Player, Cortona VRML Client etc. We require the Cortona VRML Client from Parallel Graphics for DIVA. Cortona supports EAI II, and is currently supported by Parallel Graphics unlike Blaxxun or Cosmo, which are no longer in production.
**EAI + Applet VRML Engine**

This engine is implemented as a Java applet and communicates with the 3-D scene on the client side. The EAI is an interface specification that allows an external program to communicate with a VRML scene. To perform the required OLAP operations the user events are wired to the applet. The sensor nodes in the VRML scene are responsible for wiring these events to the applet.

The operations that can be completed on the client side without a round trip to the server are *slice*, *dice* and *roll up*. To perform *slice* and *dice*, all the user has to do is click on the buttons (sensor nodes) with specific values of interest. For the *roll up* operation the user has to choose from the concept hierarchies (discussed in the previous chapter) displayed in a text area. The VRML Engine then performs the required aggregation and alters the node values in the existing scene.

For *drilldown* the client side must go to the server as we are moving from coarser-granularity data to finer-granularity data (as discussed in previous chapter). The user therefore has to choose the options from text area and the client sends the required information to the server.

**Relational Data Generator**

The *Relational Data Generator* is responsible for generating the relational view of the data in the current scene. The relational view of data means presentation of data in a tabular format. The VRML Engine prepares this view by taking into account all the nodes currently visible and uses JavaScript to pop up a new window with this information (in HTML).
Server Side Module

The Server Side Module is mainly responsible for processing client requests, prepare queries, query the data warehouse, apply rules, generate VRML scenes and return them to the client. This module consists of four components: Meta Info Server, Main Engine, Database Interface, and VRML View Handler.

Meta Info Server

We require information about the data warehouse before the application start-up. It is therefore the user’s responsibility to run the Meta Info Server before the application can start. The main page displays the link to generate this Meta information.

Meta information includes the information about dimensions, measures, concept hierarchies, themes, and associated tables and keys. This information is then stored in the system specific database.

DIVA accepts the Meta information as a valid XML document. The sample XML document can be found in the Appendix. The user can also directly insert the data as valid SQL commands. DIVA processes the XML document and converts the XML tags into valid SQL expressions. After this conversion, it executes these SQL expressions and displays the results to the user.

Main Engine

The Main Engine is responsible for routing the client requests to the appropriate subsystem or component. The Main Engine constructs the query string progressively with user inputs. This method ensures that DIVA can also act like a Web service if required.

Database Interface

The Database Interface provides database access for all the components. The interface exposes several methods for accessing the data. We use an open source database
connection manager with our interface to provide connection pooling. Reusing the
customization dramatically improves the performance of database driven Web applications.
The connection string is specified in a file outside of the compiled code. The differences
between the databases are transparent to the components calling the Database Interface.

**VRML View Handler**

The Main Engine calls the VRML View Handler. The VRML View Handler then
processes the user inputs and calls the supporting components. The VRML View Handler
captures the input errors before invoking the subcomponents. The four major components
of the VRML View Handler are described in the following subsections.

**Query Generator**

The Query Generator is responsible for generating the queries required for creating
the VRML scene. To generate the required queries it first queries the Meta Info database
to find the *dimension*, *measure*, and *theme* information. The Query Generator then
compares the user input against the Meta information to determine the current concept
hierarchy level for each dimension. The master query is a result of several small queries
fired by the Query Generator in response to user inputs.

**Rule Engine**

Rules for DIVA are stored in a *conf* file in the application root directory. The Rule
Engine reads these rules and applies to the scene to be generated. The Rule Engine is
invoked just before the generation of scene and just after the required user input has been
collected. Rules are required to ensure the integrity of the system and to prevent
unwarranted system crashes. For example, the number of cuboids in the VRML world
can vary from scene to scene. These cuboids decide the memory consumption of the
system. If the system is about to generate 8000 cuboids then a system with low memory
may crash. To prevent this, a limit on the maximum number of cuboids can be set in the
conf file. Other rules and user preferences could be size of each cuboid, label size, view-
points, etc.

Modeler

The Modeler contains the main VRML generation engine. It takes its inputs and required SQL queries from the Rule Engine and the Query Generator respectively. Modeler processes, validates the query results, and formulates the structure of required VRML output. The Modeler calculates the translation strings for labels, sensors and data cuboids on the fly. It is also responsible for adding colors and other relevant information to the VRML Scene.

VRML View Generator

The VRML View Generator is responsible to provide the inputs to the Modeler and acts like a wrapper around the entire VRML View Generator subsystem. The VRML View Generator stores all relevant information about the current scene. It exposes various public methods for the other components to access this information. For example, the View Handler requires the maximum cuboids on each axis, the maximum concept hierarchy levels, display names for the dimensions etc. The VRML View Generator provides all this information to the View Handler. It is also responsible for setting the parameter values (provided by the View Handler) and resetting them once they are used. For example, if a user chooses to drilldown after the initial scene generation then the present concept hierarchy level for the drilldown axis should be reset.

In addition to the above-mentioned functionalities the VRML View Generator provides another very important interface to the View Handler. This interface allows the View Handler to monitor the view construction status. The View handler uses this
monitor to prevent the client side applet from loading until the view on the server side is ready. This is a very important and necessary feature. If no such mechanism is used, then in most cases applet will load faster than the VRML scene on the client side. Once the applet is initialized, it then starts searching for the VRML nodes. This causes a client crash if the required nodes are not found. The VRML View Generator also provides interfaces to load static VRML files and save the dynamic VRML scenes generated.

Data Warehouse

Besides the data, the Data Warehouse also stores Meta information, which describes the content of the warehouse. As described earlier in this chapter, the Meta Info Server collects information such as dimensions, themes, concept hierarchies and measures and stores it in application specific database. This data is then used by the Query Generator to generate dynamic queries. We discuss more about this in the next chapter.
CHAPTER 4
IMPLEMENTATION

In the previous chapters, we discussed the underlying technologies and how they are used in DIVA architecture. This chapter discusses implementation details from a programmer’s perspective. We start with the data warehouse and follow with detailed descriptions of server side module and client side module. The description will cover the challenges faced during the implementation of these modules. The client side implementation of this project is done by another student, Ms. Hongyan Zhao. The DIVA prototype system is currently installed in the database research center at University of Florida. It can be accessed through any browser equipped with a Cortona VRML plug-in [22].

Data Warehouse

The data warehouse for this particular test instance of DIVA is a car sales warehouse. We used the Oracle 8i Server in the CISE department for testing purposes. The data was loaded using the Oracle bulk loader.

The data warehouse is designed using a star schema as shown in Figure 4.1. In the previous chapter, we have already discussed about the popularity of star schema in warehouse design. This is not a very complex data warehouse, but useful in demonstrating the power of DIVA. There are three dimensions in this schema: Dealers, Days, and Cars. Each dimension table has a primary key (underlined in the Figure 4.1) and a set of related attributes. There is one fact table called Sales, which contains the keys to all the three dimensions and the measure price. We have two concept hierarchies,
the time hierarchy, and the location hierarchy, as shown in the Figure 4.1. This allows the user to view data at varying levels of abstraction. Through out this chapter, schema refers to the schema depicted in Figure 4.1 and described in this paragraph. This schema was used by the prototype system for testing purposes.

![Car Sales Star Schema](image)

Figure 4.1 Car Sales Star Schema used for testing in DIVA.

**Server Side Module**

As described earlier, the *Server Side Module* processes client requests, queries the data warehouse, applies rules, generates VRML scenes and data, and sends them back to the client. We will cover all the subsystems in the *Server Side Module* in the following subsections.

**Database Interface**

The *Database Interface* provides data access for all the other components in DIVA. This component is written as a wrapper for another component called Database...
Connection Manager. The advantage of using a wrapper class is to provide modularity to the application. Data access is transparent to all the other components. Error handling is done at the *Database Interface* level and propagated to the component that calls the DB interface.

The *Database Interface* is implemented as a Java class called *DBInterface*. Whenever a component in the system needs data from the warehouse, it invokes the *ExecuteS()* function of *DBInterface*. The function *ExecuteS()* in turn instantiates the *DBConnectionManager*, and executes the SQL statement and returns the result set to the caller.

The Database Connection Manager employs the *Connection Pooling* technique. Connection Pooling is a technique that maintains a pool of database connections and uses an available connection from the pool for subsequent requests. It creates a new database connection only if there are none available in the pool. This saves connection time, memory, and other resources. There is no need to hardcode the connection string; instead a special file called *db.properties* is used to specify data source name and other required parameters.

**Meta Info Server**

The *Meta Info Server* is responsible for creating the system specific database meant for storing the information about the data warehouse itself. The Meta Info schema is shown in Figure 4.2. This schema allows multiple *dimensions*, concept hierarchies, *measures* and *themes*. For example, there may be more than one fact table for a more complex data warehouse. The *fact_tbl_key* in *tblKeyRelations* in the schema ensures that multiple fact tables can co-exist.
We assume that a domain expert will generate this data. We further assume that this kind of data should be in XML, since XML is the universal choice for data interchange nowadays. A sample input XML document can be found in the Appendix. We then use XSLT to transform this XML document into a series of SQL statements.

The Java class responsible for transforming the input XML document is called XML2SQL. This class invokes the Saxon XSLT processor (on the server) and applies the transformation specified in an XSL file to the input XML document. A sample XSL code is listed in Figure 4.3. This transformation class also exposes an interface to populate the Meta Info tables with the data that is generated in the previous step. All these actions are triggered when the user chooses to run the Meta Info Server in metainf.jsp. The populated data can be viewed in a user-friendly format by running showTables.jsp.
We explain in subsequent sections how this Meta info is used to generate dynamic queries.

VRML View Handler

The VRML View Handler consists of four components namely Query Generator, Rule Engine, Modeler, and VRML View Generator. In this subsection we discuss the implementation details of these four components. This is the most important part on the server side from an implementation perspective.

Query Generator

The Query Generator is responsible for generating queries required by DIVA for 3-D scene creation. For this purpose, it extensively uses the Meta Info data. The Query Generator needs some inputs like dimension names, the aggregate operation to be performed on the data etc. It gets these inputs from the other components (which call the Query Generator). The Modeler, which uses the Query Generator extensively, expects the results in a consistent format. The challenging task then, is to maintain this format
unchanged for all the user operations. We will first describe a sample query with default concept hierarchy levels for all the dimensions. Figure 4.4 shows one such query, the parts of the query that are stored in Meta Info database are indicated with a dark gray color. The user input parts of the query are shown using a light gray color.

![Sample query for 3-D scene creation.](image)

This query will produce result in a format measure, X, Y, Z, where X, Y, Z represent the dimension values for the three dimensions and measure will be the aggregate value to be shown for that cuboid. The user inputs and Meta Info inputs are shown in the Figure 4.4 with different colors. The “Group By” clause of the query is constructed according to the concept hierarchy levels that are chosen by the user. By default, the highest concept hierarchy levels are chosen.

If the user chooses a concept hierarchy level lower than the highest level (for that hierarchy) then we have to readjust the query to give the output in the same format (measure, X, Y, Z). For example say the X dimension has a Year–Month hierarchy, then the X values should look like 1999-01, 1999-02 etc. We readjust the query instead of doing changes in the application code because the complexity of handling generic cases for multilevel concept hierarchies at application level is high.
To solve the problem described above, we use the concatenation operator. Now we do not have to keep track of the history of concept hierarchy levels that the user is navigating through. We simply keep track of the present level and find from the Meta Info, all the hierarchies levels that are above the present level. For example, in Figure 4.5, we form the concatenated string \( \text{Days.year||'-'||Days.month} \), if the user chooses month as the hierarchy level for Y-axis.

Having solved the query problems for normal operations we return to drilldown. Drilldown adds a few new problems, which cannot be tackled at the application level in a straightforward manner. These problems include:

- Keeping track of the operations (slice, dice, rollup), carried out by the user at the client before choosing drilldown.

- Keeping track of the null nodes. The server constructs the nodes on the fly and does not know anything about the null nodes. Handling this at application level is an expensive task.

- Keeping track of the range of nodes that are needed to be drilled down.

  It is again desirable that we adjust the query to get the output in the same format and keep the Modeler unaffected. We use the logical OR operator together with the UNION operator to solve these problems. The UNION part solves the null node problem;
the sum aggregation tricks the system in yielding the two sub queries at the same operational level. The logical OR combined with a sorted hashmap in the associated Java class solves the range problem.

Figure 4.6 Sample query for 3-D scene creation with lower concept hierarchies and drilldown.

Figure 4.6 depicts a sample query used for drilldown. Note that all the problems described above were fixed using the query itself. The second subquery generates a Cartesian product of all the relations involved, with its value of measure as zero (the actual number of tuples for a valid data cube). The fake wrapper adds this to the actual measure values generated by the first subquery, hence producing zero value cuboids instead of null nodes.

Rule Engine

The Rule Engine is invoked after the user selects all the inputs required for generating the 3-D scene. The applyRules.jsp file then sets the parameters (chosen by the
user) and invokes Rule Engine. The Rule Engine is a Java class file that reads the rules from a conf file called diva.conf. The rules are specified in simple text format (e.g., SPACING=2 MAXCUBOIDS=20 etc.). Once the rules are set, the methods exposed by Rule Engine can be used to check the validity of user inputs.

Modeler

Once the required parameters are set by VRML View Generator subsystem and the Query Generator subsystem has finished generating the required queries, the Modeler is invoked. The Modeler has two subcomponents: one to deal with normal operations and the other to deal with drilldown operations. Drilldown operations require special attention for the reasons described above. To create VRML instances, the Modeler uses a library called CyberVRML97.

The method that is responsible for generating the final view is called makeView(). It creates the nodes in Z, Y, X fashion. That means all the Z nodes for a particular X and Y value will be processed, then all the Y nodes for the X value will be processed and so on. This method of generating cuboids is illustrated in Figure 4.7 and Figure 4.8 below. The translation strings that decide the position, scale and orientation of the cuboids are computed before the node is added to the scene graph. The Modeler keeps a track of these nodes by maintaining hashmaps. These hashmaps come in handy for range calculations required in a drilldown operation. The naming conventions for the nodes, sensors and labels follow the same guidelines that were applied to the client.

The Modeler uses Billboard Transforms to make labels automatically rotate to face the user. Once the Modeler has produced all the cuboids, the headers containing navigational information are added to the scene.
The **Modeler** is responsible for generating the dynamic data required to initialize the applet. In the next subsection, we will discuss about this data.
**VRML View Generator**

The *VRML View Generator* is implemented as a set of JSP files. As described in the previous chapter, the *VRML View Generator* component wires the inputs to other components in this subsystem. The *VRML View Generator* creates an instance of the *Modeler* and then various methods are called to set the required parameters.

The client side applet requires some dynamic data to be initialized. This includes the maximum number of cuboids in each dimension, maximum hierarchy levels, names of all concept hierarchies etc. Table 4.1 shows the format in which this data is required.

Table 4.1 Dynamic data required by the applet to initialize

<table>
<thead>
<tr>
<th>Param Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>numOfDimensions</td>
<td>3</td>
</tr>
<tr>
<td>fromToValue</td>
<td>all</td>
</tr>
<tr>
<td>dimensionName</td>
<td>Time</td>
</tr>
</tbody>
</table>

The VRML View Generator supports an interface called isViewReady (). This interface indicates whether the current view is in the completed state. The client side uses this interface to halt the thread until the view is prepared. The consequences of not doing this were discussed in the previous chapter. Once the view is ready, the Server notifies the client and the client allows the applet to start execution.

**Main Engine**

The *Main Engine* consists of a number of JSP files. The first JSP file is called main.jsp. It shows the system requirements for DIVA, checks the VRML browser installation, and provides an option to run the *Meta Info Server*. If the user chooses to run the *Meta Info Server*, the option to view the Meta Info data (dynamically generated) is also displayed.
The index.jsp contacts the data warehouse for the information about the Meta data dimensions, measures, and themes. The information returned from the warehouse is then displayed by the file in the form of a list so that the user can select dimensions, measure, and theme to view the data cube.

The Main Engine forwards the user input to subsequent JSP files in a consistent format. This parameter forwarding is applied to the Rule Engine as well. This method is discussed at length in Hongyan [9].

**Client Side Module**

The task of this module is to provide a user interface for DIVA. This part of the project was implemented by Hongyan and the implementation details can be found in her thesis [9]. However, a few changes were made to the client side, which will be discussed here.

As described earlier, a monitor prevents the applet from loading before the view is ready to render. However, there is one more problem associated with the loading of the applet, which is rendering delay. Figure 4.9 depicts three sample nodes and their rendering timelines. This problem of rendering delay is more obvious in systems with low memory. The VRML browser does the node rendering. We will assume that it renders these nodes based on the VRML code (generated by the server). The Node generation scheme was discussed in previous subsections. In Figure 4.9, we can clearly see that Node 2 and Node 3 will be loaded after Node 1. The applet loads faster than the VRML and tries to initialize various node properties in its data structures. Once it finds Node 1 there is nothing that prevents the applet from trying to initialize properties for Node 2. This will cause Null pointer Exceptions, unless handled. This behavior was therefore changed and the applet is now forced to wait until the last node is rendered.
This solves another problem, that of hitting *refresh* once the scene is rendered. The *refresh* problem occurs because the applet (being in browser cache) loads much faster than the scene itself.

Figure 4.9 Rendering of nodes in the VRML browser.
CHAPTER 5
OPERATIONAL DESCRIPTION

This chapter presents an operational description of DIVA. It explains how to use this system to perform OLAP operations. Figure 5.1 shows the starting page for the system. This page provides the information required for system startup. The user can check for VRML browser installation by clicking the first hyperlink (as indicated in the figure). There is an option for generating the Meta information by running the Meta Info Server. The response page, which is shown to the user after running Meta Info server, is not shown in the figure. The response page provides further information about the Meta information generated.

Figure 5.1 Starting page for DIVA.
User Input

The user is shown a set of input pages to gather the prerequisite information about the VRML scene. We start with the index page that is the normal start page for the system. In this page we provide the user with the dimensions and measures that are available in the system. The user can then choose a maximum of three dimensions, one theme, one measure, and an aggregate function to be applied on that measure. The user can hit the next button to continue. This input sequence is depicted in Figure 5.2.

Figure 5.2 Data cube setup page.
The next screen shows the user all the concept hierarchies for the dimensions chosen in the previous page. The user can use the combo box for each dimension to choose the desired initial concept hierarchy level. Figure 5.3 presents one such snapshot.

![Image of the user interface for selecting concept hierarchies](image)

Figure 5.3 Depicting a user choosing an initial concept hierarchy level for each dimension.

The response page for this input step is not shown in the figure. The rules are then applied to the user input collected and an error is signaled to the user if any rules specified in the configuration file are violated. If the rules are successfully applied then a success message is shown to the user. The user can then proceed with the generation of the initial scene.

### Initial Scene Generation

Figure 5.4 shows a snapshot of the data cube before any operations have been performed (initial scene). The initial scene consists of three frames. The upper left-hand frame (indicated as 1 in the figure) is loaded with the VRML browser that displays the data cube. The lower left frame (indicated as 2 in the figure) is loaded with the applet...
user interface to control the VRML scene. The lower right frame (indicated as 3 in the figure) provides the user with some help information (about the operations).

Figure 5.4 Data cube after initial selection of hierarchies, dimensions, and other operational parameters per Figure 5.2 and Figure 5.3.

As we have presented in Chapter 4 the already implemented operations include slice, dice, roll up, and drill down. In the rest of this chapter we provide a description of how to perform these operations.

**Slice**

To perform a slice operation the user has to click one of the sensor nodes that represents a specific value. The sensor turns green to indicate the activation. Only the cuboids carrying the selected dimension value are visible after this operation. In Figure 5.5, the slice operation is performed for the time dimension with value 1998.
To undo a *slice* operation the user has to click the activated sensor again. The sensor turns white, indicating the deactivation. The scene is restored to the pre-slice state.

**Dice**

The *dice* operation performs a selection on two or more dimensions. In order to carry out a *dice* operation, the user makes a selection on two or more dimensions by selecting at least two white sensor buttons. This sets the range of operation in the data cube.

Figure 5.6 depicts the selection on all the three dimensions. The *dice* is similar to a sequence of *slice* operations. The selection criterion for the dimensions is indicated in the figure. The order of selection does not matter. The *undo* operation for *dice* is very similar to the *undo slice* operation, which is done by turning green sensors back to white.
Roll up

A roll up operation changes the data cube for a particular dimension from a finer granularity to a coarser granularity.

The applet user interface (AUI) is shown in the lower left section. There are three lists called DIMENSION, FROM, and TO. The DIMENSION list shows the available dimensions for this VRML scene. The FROM and the TO lists hold the hierarchy levels. The user specifies the dimension for which he/she would like to roll up by choosing its name from the DIMENSION list. The FROM and the TO lists will be populated with the hierarchy levels for the specified dimension. The user then specifies a level in the FROM list to indicate the starting point and choose a level in the TO list to indicate the level at which he wants to view the data. According to the definition of roll up, a valid pair assumes that the selected item in the TO list is at a higher level in the concept hierarchy.
than the item in the FROM list. The user clicks on the ROLL UP button in AUI to perform the *roll up* operation. The *roll up* operations can be mixed with *slice* and/or *dice* operations. The *undo roll up* operation restores the scene to the pre-*roll up* state.

Figure 5.7 describes the *roll up* operation. The concept hierarchy level for the *time* dimension is collapsed to show only the aggregate values for all years. This *roll up* was performed after a *dice* operation on the *z-axis* by selecting only CT and FL states. The color scheme described in Hongyan [9] is turned off. If it is turned on, the cuboids show colors according to the pre-specified color scheme.

![Figure 5.7 Roll up operation.](image)

**Drilldown**

For a *drilldown* operation, the user relies on the same set of lists as a *roll up* operation, namely DIMENSION, FROM, and TO. In the DIMENSION list, the user
specifies the dimension of interest. In the FROM list, the appropriate item to choose is the current level in the concept hierarchy, while in the TO list, only one of the levels that are lower in the hierarchy (than the present level) should be chosen. After the user clicks on the DRILLDOWN button, a refreshed scene with finer-granularity data is loaded to replace the old scene. Then the user can examine this new data cube by performing slice, dice, and roll up operations in the same way as we have described earlier in the chapter.

Figure 5.8 (in the next page) describes a drilldown operation. The year level in the time dimension is drilled down to month level.

Figure 5.8 Drilldown operation.
CHAPTER 6
EVALUATION

In the previous chapters, we described the architectural and implementation details of our system DIVA. In this chapter, we evaluate the application based on its features.

Intuitive Interface

DIVA transforms the data into an interactive 3-D world. The user need not know OLAP queries and other data handling procedures. The conventional data analysis tools provide 2-D representation of data. Adding another dimension increases the amount of information displayed. With 3D cubes, the user can visually locate areas of interest and then go deeper to uncover further detail. Therefore it is much easier for him to find trends, determine patterns, and spot anomalies within the data warehouse [9]. This interface can be considered as an abstraction over SQL queries.

On-the-fly Processing

The data cubes are generated on the fly after the user specifies the parameters. The data cubes are neither stored on the files system nor cached; they are generated on the fly. The slice, dice, and roll up operations can be completely performed on the client side. This saves round trips to the server and these operations can be completed in an acceptable amount of time. However, the client side operations increase the memory requirements for the system.

Use of Visual Cues to Improve the Interface

DIVA uses color as a visual cue to enable the user to gain more information from the same scene. The colors chosen are based on a color spectrum embedded in the Applet
User Interface. Smaller values are towards the red end of the spectrum and larger values are towards the purple side. However this operation increases the memory requirements of the client and is turned off initially. The user needs to change the *conf* file settings to turn this feature on. As a future extension, we recommend adding more visual cues such as *size of the cuboids*.

**Modular Design**

DIVA is a highly modular system. This high modularity ensures that technical advances in this area can be easily accommodated in the system, with minimal changes. For example if the system has to output in a different markup language like X3D (introduced in Chapter 2), then we just need to change the interface for the server module. The VRML library used in DIVA already supports X3D to some extent. The client side VRML engine should be modified to replace the built-in functions in EAI with their equivalents in the Scene Access interface [9].

**Suitable for Many Applications**

DIVA allows a business user to quickly answer a series of related questions. These questions may include:

- Over the last six years, how were the sales of *Toyota* cars in US?
- How does it compare with same figures in Canada?
- What was the state wise distribution?
- What models of *Toyota*, did not perform to expectations in Florida?
- What were the months in which US performed better than Canada?

These are the questions that can be easily answered by DIVA because of its multidimensional features. Since DIVA is a Web based tool, it can be ported to computer
notebooks or handheld devices. The user can give dynamic presentations with latest snapshots of the business data.

DIVA can be used as an excellent teaching tool. The OLAP operations can be explained visually through DIVA. Data mining applications rely on pattern recognition; DIVA can be used as a basic building block for these applications. DIVA can be used as a Web service to provide 3-D snapshots (in 2-D format like gif) to other applications. DIVA can also be used by decision support systems.
CHAPTER 7
CONCLUSIONS

This chapter summarizes the contents of this thesis, its contributions to the state-of-art in visual analysis, and discusses the directions of future work.

Summary of Thesis

In this thesis we have described an important prototype tool for visual analysis using OLAP techniques. The goal of our system is to provide a Web based, easy to use 3-D visualization and analysis tool to the user. The system allows the business users to quickly answer a series of related questions (about their business data). In Chapter 1 we discussed the motivation for this project. We also described various commercial tools available in the market with similar functionalities. We then described the underlying technologies, important for this architecture. These technologies include data warehousing, OLAP, VRML, EAI, Java Servlets and XSLT. Chapters 3 and 4 describe the working of this system. We covered the important aspects of all the components of this system and discussed the challenges faced (during implementation). Chapter 5 provided an operational manual for DIVA. Finally, we evaluated the system based on its features. The most desirable features of our system are Web-based OLAP analysis with an intuitive user interface, on-the-fly processing of data, application of visual cue to help reveal more information, and high modularity design for system flexibility and maintainability.
Contributions

We believe that DIVA is a significant tool that can be applied to various knowledge extraction problems. DIVA introduces a new client-server based 3-D visualization system that includes some novel features like Meta Info management. This prototype also proves that 3-D modeling languages can be applied to databases in a pragmatic way to develop tools for next generation. The X3D community can also use this tool to build prototypes for developing database tools based on their new standard. E-commerce sites like airline reservation systems or rental car booking systems can try DIVA for better representation of their status information. We have developed DIVA as a framework for integrating many presentation and database technologies in a very modular fashion. These modules can accommodate future advances in related areas.

Future Work

There are few interesting features that can be added to DIVA to extend its functionality. We discuss these features in this subsection.

The current system only supports three dimensions; an additional dimension can be added in many ways. One way is to show a series of 3-D data cubes in multiple windows. Another way is to use some kind of animation to build a slider functionality. We can associate the fourth dimension with a visual cue like size of a cuboid. The slider will represent different values for the fourth dimension. When the slider changes its position to a different value the size of the cuboids change to reflect associated values.

Another interesting feature is to display multiple measures in the same scene. We could again rely on visual cues to display multiple measures in the same scene. We can also use VRML animation for this purpose. Each cuboid acts like a file cabinet, when the user opens the cabinet, measures can be viewed as files in that cabinet. Besides the
operations implemented in this system, OLAP offers many more operations. These operations can be included in the application.
APPENDIX
SAMPLE XML INPUT FOR META INFO GENERATION

```xml
<?xml version="1.0" ?>
<root>
  + <dimension>
    - <dimension>
      <id>2</id>
      <name>Dealers</name>
      <tbleme>Dealers</tbleme>
      <defaultlevel>state</defaultlevel>
      <facttbleme>dealerID</facttbleme>
      <dimtbleme>dealerID</dimtbleme>
      <facttblemeName>Sales</facttblemeName>
    - <concept>
      <level>1</level>
      <name>state</name>
      </concept>
    - <concept>
      <level>2</level>
      <name>city</name>
      </concept>
  </dimension>
  + <dimension>
    - <measure>
      <id>1</id>
      <facttblemeName>Sales</facttblemeName>
      <coiname>price</coiname>
    </measure>
  </dimension>
</root>
```

Only one <dimension> node is in expanded mode.
LIST OF REFERENCES


BIOGRAPHICAL SKETCH

Prakash Bulusu was born in Raipur, India. He received a Bachelor of Engineering degree in electronics and communications from Regional Engineering College, Trichy, India, in May 1998. After graduation, he worked with Wipro Technologies as a system analyst.

In May 2001, he was accepted into the Department of Computer and Information Science and Engineering at the University of Florida. He will receive his Master of Science degree in December 2001. His research interests include Web-based database driven applications, content management applications, and database modeling.