Developing a general purpose OLAP client prototype using XML for Analysis

Igor Mekterović , Mirta Baranović
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3, 10000 Zagreb, Croatia
Phone: +385-1-6129-790 Fax: +385-1-6129-915 E-mail: igor.mekterovic@fer.hr

Data warehouse is a copy of transaction data specifically structured for query and analysis [1]. Within data warehouse, data is usually stored in a dimensional model.

On-Line Analytical Processing (OLAP) is an approach to analysis and reporting that enables a user to easily and selectively extract and view data from different points of view based on a multidimensional data structure called a cube [2].

There are numerous data warehousing platforms available on the market. Exhaustive data warehousing platform comprises of data warehouse server, OLAP server and some data mining functionality.

There are no present standards to support interoperability between different OLAP systems. In this article, two major initiatives towards standardization are examined: XMLA and JOLAP.

Web service is a piece of business logic, located somewhere on the Internet, that is accessible through standard-based Internet protocols such as HTTP or SMTP [3]. Web services are neither language nor platform-specific.

Using Java programming language, general purpose client tool prototype for standardized OLAP data access is developed. Client communicates with OLAP data source through web service and is based on XMLA 1.1. specification.

It is concluded that the client encompasses satisfactory set of functionalities.

I. INTRODUCTION

Today's information systems mostly rely on relational databases for transactional data storage and manipulation. Throughout the years, massive amounts of data have been gathered. Those data, if properly analyzed, could serve as a basis for strategic decisions. For the purposes of analysis data is transformed from relational model (database) into dimensional model and stored in a data warehouse. Informations gained from gathered data provide an advantage that, in the conditions of sharp business competition, could be distinguishing. Followed by intense hardware development, that was the main reason for rapid data warehousing development and acceptance in the past years.

However, data warehousing platform includes more than relational database holding data in a dimensional model: exhaustive data warehousing platform comprises of data warehouse server, OLAP (On-Line Analytical Processing) server and some data mining functionality. OLAP stands for a type of application that attempts to facilitate multidimensional (i.e., data that has been aggregated into various categories or "dimensions") analysis. OLAP should help a user synthesize enterprise information through comparative, personalized viewing as well as through analysis of historical and projected data. The main characteristic of OLAP systems is providing multi-dimensional view of the data and answers to queries at high response rates.

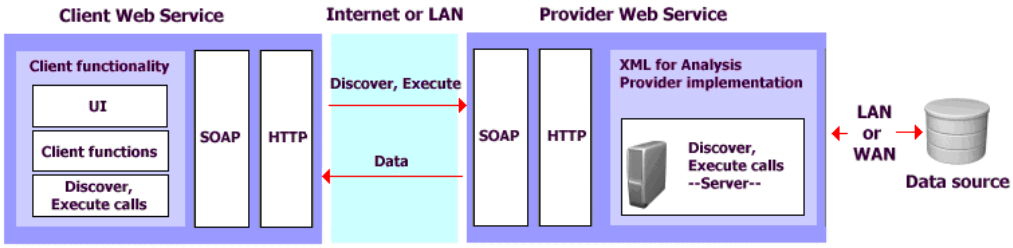
Unlike relational databases that show some degree of uniformity through the SQL and ODBC standards, there are no generally accepted standards for communicating with OLAP databases. Such situation makes development of OLAP client tools more complicated and expensive. This results in platform specific and more expensive products that are, therefore, less acceptable to the end users.

In this article two major initiatives for OLAP standardizations are discussed. Also, an OLAP client tool based on the XML for Analysis (XMLA) protocol with a sufficient set of functionalities was developed.

II. EMERGING STANDARDS

Two major standardization initiatives for communication with OLAP data sources present today are Java OLAP (JOLAP) and XMLA.

Java™ OLAP Interface (JOLAP) is a pure Java™ API for OLAP servers and applications deployed on the Sun Java™ 2 Platform, Enterprise Edition (J2EE™) [4]. JOLAP has been developed within the Java Community Process by a number of companies dealing with the OLAP, business intelligence, and data warehousing domains, such as Oracle, Sun, IBM, Unisys, etc. Since there is no common logical model for OLAP, JOLAP leverages the CWM (Common Warehouse Metamodel) OLAP metamodel and other metamodels of CWM as the base metamodel for its own metadata model. CWM is an open industry standard for interchange of warehouse metadata developed by Object Management Group™. It provides a framework for representing metadata about data sources, data targets, transformations and analysis, and the processes and operations that create and manage



Picture 1. Client communicating with OLAP data source using XMLA protocol

warehouse data and provide lineage information about its use. JOLAP defines an object model for metadata and query manipulation. However, JOLAP does not define a linguistic interface (such as SQL in relational databases) - client application has to combine JOLAP classes (objects) to form a query. The client-side metamodel is defined as projection (subset) of the CWM OLAP metamodel that is required by JOLAP clients; that is, required by users of the JOLAP Query model for forming queries. Base class, called Schema, contains classes Cube and Dimension, Cube contains Measures, Dimension contains Hierarchy classes, etc. Members retrieval is managed using cursors which are suitable for retrieving member information from levels that contain large number of members. Query object models and languages are critical for developers who use an analytic API. In the case of existing language interface (e.g. SQL) a client application has to have some sort of internal model for holding query components and metadata so that query could be manipulated and eventually assembled.

JOLAP models all queries as collections of objects designed to reflect GUI gestures in constructing and refining a query - there is no subsequent step required of an application to traverse the objects and generate a textual query [5]. It even supports rollbacks on portions of query created by client's actions, for instance a drilldown or filter modification could be rolled back to return a query to a previous state using this API's build-in functionality.

However, JOLAP is platform dependant - it is a Java API. XMLA uses a different approach: XML for Analysis specifies a SOAP-based XML communication API that supports the exchange of analytical data between clients and servers on any platform and with any programming language. In short, clients communicate with OLAP data source through web service, i.e. via Simple Object Access Protocol (SOAP) messages. This specification is built upon the open Internet standards of HTTP, XML, and SOAP, and is not bound to any specific language or technology.

Furthermore, XMLA provider web service is very simple: it supports only two methods: discover and execute. Discover is used to retrieve metadata, such as the list of available data sources on a server or details about a specific data source, cubes and dimensions available, etc. The Execute method is used for sending action requests to the server (Picture 1.). This includes requests involving data transfer, such as retrieving or updating data on the server. Typically, execute method is used to send a query using mdXML (multidimensional XML).

The XML for Analysis specification requires that multidimensional providers support the mdXML language. The mdXML language is based on MDX (Multidimensional Expressions). MDX is the multidimensional expression language defined in Microsoft's OLE DB for OLAP specification. It is a linguistic interface that is supposed to play a role similar to SQL in relational databases. Currently mdXML consists solely of the <Statement> element that contains an MDX language statement. Future enhancements to mdXML will make additional elements beyond the <Statement> element available. However, for the time being, mdXML simply wraps MDX statements in a <Statement> tag. XMLA council, responsible for development of XMLA, is composed of number of companies such as SAS, Hyperion, Business Objects, Cognos, Microstrategy, etc. but clearly, the main role in the council and in the development of XMLA is held by Microsoft.

It is not evident which standard will prevail. There are opinions that these are even not competing standards, but rather complimentary. For instance, Hyperion supports both standards and implements XMLA web service using JOLAP. These standards serve two different needs: JOLAP is for the Java developer, there's no linguistic interface, whereas XML for Analysis is much more of a linguistic interface, a query language specification combined with Web services. However XMLA was released first and has gained a foothold in the industry. All this considered, it was decided to develop a client prototype using XMLA protocol.

IV. CLIENT PROTOTYPE REQUIREMENTS

Based on previous experience and existing client tools assessment, a list of required features for general purpose client tool prototype is assembled:

- platform independent
- graphical interface
- user-friendliness, usability
- no previous knowledge in IT required
- interactive queries
- a possibility to browse more than one cube
- drill-up / drill-down
- dimension members (to be displayed) selection
- ordering (by dimension members and by measures, hierarchical and non-hierarchical)
- filtering, more than one member in the filter axis
- display up to 3 axes (columns, rows, pages)
- basic data visualization (common data charts)
- secure (encrypted) communication

Foremost, an easy-to-use and user-friendly interface is required so that even a user with little or no computing knowledge should be able to use it. Combined with features listed above, that should satisfy the requirements of most OLAP users since more advanced analysis (e.g. statistical analysis, data mining, etc.) are

performed by system analyticians using specialized software.

V. IMPLEMENTATION ISSUES

During implementation several issues were encountered that will be further commented:

A. SOAP processing and secure communication

Since XMLA protocol is used client has to communicate with web service using SOAP messages. Furthermore, this communication should be encrypted. It was decided to use Apache Axis as a SOAP engine because Axis provided serializer classes (needed to transform java object into its SOAP equivalent). Also, Axis provided a tool, called wsdl2Java used to create stub classes – these classes are used to hide the details of calls and generate a wrapper classes for a web service.

SOAP communication is done over a secured (https) channel. To achieve this (for a priori unknown URL) Axis source code had to be changed to accept HttpsURLConnection.getDefaultSSLSocketFactory() class that can be then set from outside Axis. This is, of course, considered to be a drawback.

B. Results display

It is typical for OLAP tables to have merged cells on rows and columns headers. These come from cross joining different dimensions. Picture 2 shows a table with dimensions A and B cross joined on rows and dimensions C and D cross joined on columns.

		C	
		D1	D2
A1	B1		
	B2		
A2	B1		
	B2		

Picture 2. Typical OLAP result table

Table headers cells have to be responsive to user input (double-click for drillup/drilldown and right click for various other functions). This OLAP table was implemented using standard JTable for table cells (non-shaded part in the Picture 2) and set of aligned JButtons for table row and column headers (shaded part of Picture 2). For this purpose GridBagLayout class was used. Unfortunately, this layout manager class has hard-coded grid size limitation of 512*512 and so it had to be changed to accommodate tables with more than 512 rows.

As for charting the data, jFreeChart, a free Java class library for generating charts was used. All that had to be done is provide bridge classes that will implement interfaces needed for jFreeChart to draw various charts.

C. Query generation

Essentially, there are two approaches to generating a query in OLAP client tools. In the first approach, a user composes a query from available elements (dimension levels, members, etc.) in some sort of query designer and then executes a query. Second approach, that is used here, is to generate a query after every user's action. We find this approach to be more suitable to users although it consumes more network and server resources. To generate a query application has to produce MDX statement based on user's input. Form of MDX SELECT statement used to create queries was:

```
[WITH MEMBER <aggregated_member_definition>]
SELECT
  NOT EMPTY <axis_definiton>  ON ROWS
  , NOT EMPTY <axis_definition> ON COLUMNS
  [, NOT EMPTY <axis_definition> ON PAGES]
FROM <cube_name>
[WHERE <slicer_definition>]
```

where <axis_definition> defines a set of members to be displayed on an axis. For multiple members filters, a WITH MEMBER clause was used to define aggregated members that were then used in WHERE clause to filter the result set. As indicated with square brackets these clauses can be omitted as well as the third, pages axis.

Besides this, MDX provides numerous functions that facilitate query generation: ToggleDrillState function was used to implement drillup and drilldown, Except to exclude member for a set, Order for ordering member etc.

VI. CLIENT PROTOTYPE

The basic idea is to translate user's GUI actions into MDX queries, run queries against an OLAP server, and display results in a table. Upon starting an application, a client has to enter XMLA URL. Using discover method, information about available data sources is retrieved and displayed in a tree. User then chooses one or more cubes to analyze. Cubes are displayed in different tabs. To run a query against a cube user has to chose dimension (levels) from the dimension tree (Picture 6, left side) and put them on rows, columns or pages axis. Such actions generate MDX statements that are then executed using XMLA's execute method. If, for instance, a user puts (using a mouse) Gender level (from Gender dimension) and Unit Sales measure on column axis, and Year level (from Time dimension) on rows axis, then the following MDX statement would be generated:

```
SELECT
NON EMPTY
{
     {[Gender].[Gender].Members}
   * { [Measures].[Unit Sales]}
}  ON COLUMNS,
NON EMPTY
{
    {[Time].[Year].Members}
```

} ON ROWS
FROM [Sales]

In this simple statement set of Gender level members ({[Gender].[Gender].Members}) is cross joined (operator *) with a set that contains only one member ({ [Measures].[Unit Sales]}) and placed on columns axis. Set containing all members from the year level of dimension Time({[Time].[Year].Members}) is placed on rows axis. In this example there is only one year in database: 1997. "NON EMPTY" clause is used to omit empty rows or columns. Query is run against Sales cube ("FROM [Sales]").

This statement is wrapped in a Statement tag of an XMLA soap message and run against the data source.

Result is returned as an XML that, when parsed and displayed in a table, looks like Picture 3.

	F	M
	Unit Sales	Unit Sales
1997	131.558,00	135.215,00

Picture 3. A simple query result displayed in a table

Drilldown functionality is implemented as double click on dimension member, e.g. if a user was to double-click member "1997" a more detailed table would be shown with quartiles of year 1997 (Picture 4).

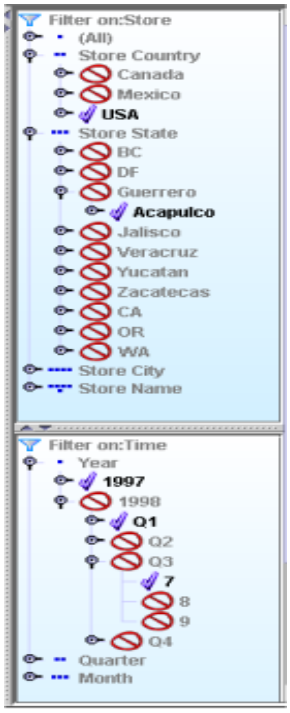
	F	M
	Unit Sales	Unit Sales
1997	131.558,00	135.215,00
Q1	32.910,00	33.381,00
Q2	30.992,00	31.618,00
Q3	32.599,00	33.249,00
Q4	35.057,00	36.967,00

Picture 4. A simple query result displayed in a table

OLAP clients usually support two axes: columns and rows. This client supports a third, pages axis, which is implemented as a tab for each member on pages axis. Obviously this axis is suitable for use only with dimensions or levels with low member count (e.g. gender, year, marital state, etc.). Since double click is reserved for drillup/drilldown operations other operations have been made available on right-click:

- changing the order of dimension on an axis
- removing dimension or member from query
- sending member to filter
- sorting by dimension (hierarchical and non-hierarchical, ascending and descending)

Filter members selection is implemented with separate tree component that displays dimension members in a hierarchical manner (Picture 5). Since dimension levels can have big member count (e.g. day level of Time dimension) lazy loading was used. Picture 5 shows filter tree where two dimensions are involved: Store dimension is restricted to USA and Acapulco, and Time dimension is restricted to 1997, first quarter of 1998 and July of 1998.



Picture 5. Filter tree on dimensions Store and Time

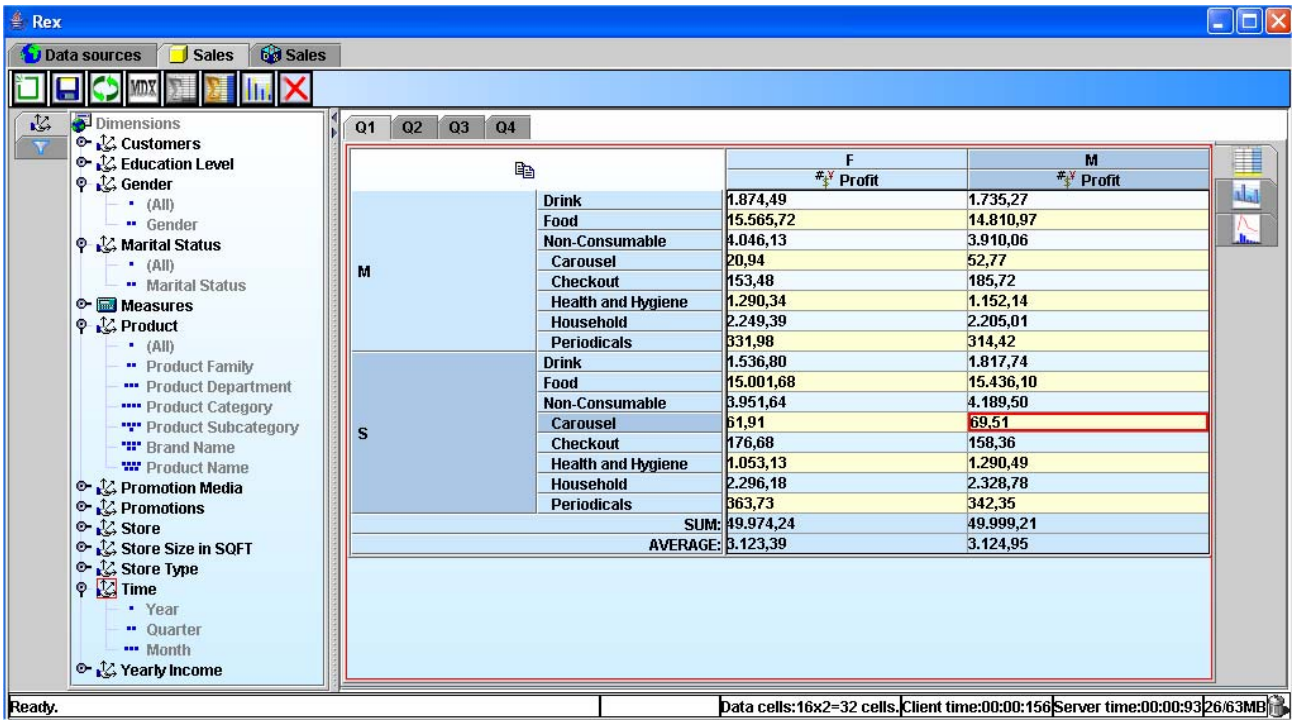
Such filter tree produces MDX with member clause:

```
WITH MEMBER [Store].[Filter members from (Store)]
  AS 'Aggregate(
      {
       [Store].[All Stores].[USA]
      ,[Store].[All
Stores].[Mexico].[Guerrero].[Acapulco]
      }
   )'
 MEMBER [Time].[Filter members from (Time)]
  AS 'Aggregate(
      {
       [Time].[1997]
      , [Time].[1998].[Q1]
      , [Time].[1998].[Q3].[7]
      }
   )'
```

that is then user in WHERE clause:

```
WHERE
 (
   [Store].[Filter members from (Store)]
  , [Time].[Filter members from (Time)]
 )
```

Picture 6 shows main application window for a query that cross joins "Marital state" and Product category dimension or rows, Gender dimensions on columns and Quarter level of Time dimension on pages. Filter and dimension trees are displayed in tabs (left side). Result table and two graphs (3D and combined graph) are also displayed in tabs (right side).



Picture 6. Main application window

VII. TECHNICAL CONSIDERATIONS

Client tool prototype is developed using Java 2 Platform, Standard Edition, v 1.4.2 (J2SE). Client tool uses Apache Axis 1.1. [6] as a SOAP engine. Also, jFreeCharts are used for displaying data charts. As an OLAP data source Microsoft Analysis Services 2000 SP3 (with XMLA 1.1. Software Development Kit) was used running under Windows 2000 Server operating system. All examples in this article are shown on Foodmart 2000 database which is distributed with Analysis Services.

VIII. CONCLUSION

This paper points out the problem of lack of standardization in the area of multidimensional databases both in metadata representation and client server communication. Efforts towards standardizing OLAP communication are inspected. It is found that there are two emerging standards: JOLAP and XMLA. These standards are different in nature: JOLAP is a Java API that has to be used from Java applications while XMLA specifies linguistic interface – a query language that is then used in conjunction with web services. XMLA is consequently platform independent. XMLA was chosen as a standard for developing a general purpose OLAP client prototype. A list of client tool requirements is proposed. Client implementation is described and features implemented are commented. It is concluded that client prototype implements proposed requirements. Client prototype can be downloaded at: http://i.zpm.fer.hr/rex

LITERATURE

[1] R. Kimball, The Data Warehouse Toolkit. John Wiley, 1996, New York

[2] Cognos Business Intelligence Guide, URL: http://www.cognos.be/biguide

[3] D. Chappell, T. Jewell, Java Web Services, O'Reilly 2002

[4] Hyperion Solutions, Sun Microsystems, Java™ OLAP Interface (JOLAP), Version 1.0, 2003, URL: http://www.jcp.org/aboutJava/communityprocess/first/jsr069/

[5] G. Spofford, Access to Intelligence: The New OLAP APIs, Intelligent Enterprise, 2002

[6] Apache Software Foundation, Axis User's Guide, 2003, URL: http://ws.apache.org/axis/

[7] W.H.Inmon, Building the Data Warehouse Third Edition, John Wiley, 2002, New York

[8] R. Kimball, A Dimensional Manifesto, DBMS Online, 1997

[9] P. Dean, Browsable OLAP apps on SQL Server Analysis Services , Intelligent Enterprise Custom Analytic Apps, 2001

[10] D. Everett, Compare and Contrast JOLAP and XML for Analysis, Hyperion Resource Library, URL:http://dev.hyperion.com/resource_library/articles/jolap_xmla.cfm

[11] S. Grimes, API Wars, Intelligent Enterprise, 2003

[12] S. Grimes, XML for Analysis Decoded, Intelligent Enterprise, 2001

[13] B.King, XML for Analysis: A Sneak Peak Inside the Skunk Works, DM Review, 2003

[14] The Java Tutorial, Sun Microsystems, 2001

[15] B.Eckel, Thinking in Java, 3rd Edition, Prentice Hall, 2003

[16] E.Armstrong, S.Bodoff, D.Carson, M.Fisher, S.Fordin, D.Green, K.Haase, E.Jendrock, The Java Web Services Tutorial 1.1., 2003, URL: http://java.sun.com/webservices/

[17] D.Almaer, Creating Web Services with Apache Axis, 2002, URL: http://www.onjava.com/pub/a/onjava/2002/06/05/axis.html

[18] SOAP Tutorial, URL: http://www.w3schools.com/soap

[29] WSDL Tutorial, URL:http://www.w3schools.com/wsdl/

[20] S.Shin, Web Services Programming using XML and Java(tm) Programming Language, URL: http://www.javapassion.com/webservices/

[21] H. Bhagwat, Web Services Tutorial, URL: http://hrishikeshbhagwat.tripod.com/webservices/TableOfContents.html

[22] E. Newcomer, Understanding Web Services: XML, WSDL, SOAP, and UDDI, Addison-Wesley, 2002

[23] R. Jacobson, Step By Step, Microsoft SQL Server 2000 Analysis Services, Microsoft Press, 2000

[24] C.Nolan, Introduction to Multidimensional Expressions (MDX), 1999

[25] Microsoft Corporation, Hyperion Solutions Corporation, XML for Analysis Specification, Version 1.1, 2002, URL:http://www.xmla.org/

[26] M. Kramer, A Comparison of Business Intelligence Strategies and Platforms, Green Hill Analysis, 2002

[27] N.Pendse, R. Creeth, The OLAP Report, URL: http://www.olapreport.com/

[28] N.Pendse, OLAP Market Review, DM Review, 2001

[29] G.Spofford, Basic Statistical Calculations in MDX and Analysis Services, DSSlab, 2001, URL: http://www.dsslab.com/MDXSolutions.html/