

Building a Corporate Warehouse Management and Deployment System

Chris Lane, Lane Systems Group Ltd., Whitehorse, Yukon (CANADA)

ABSTRACT

Managing data warehouses in a complex environment has its challenges. Dealing with multiple databases/servers, data model consistency, security, business information and access to data were the problems identified by the Government of Yukon when they wanted to create an open, enterprise-wide data warehousing solution. Industry standard access techniques, a managed centralized metabase and an end-user deployment system were identified as the solution.

HOUSING THE DATA

The metabase stores information about what hosts exist, what warehouse servers exist on those hosts and what logical libraries exist within the warehouse servers.

Warehouse servers store libraries of data. They can be a SAS/Share® server, a relational database or a local client (accessing data located on a LAN).

The following configurations are supported at the customer site:

Host	Server	Use
HP-Unix	SAS/Share (or DBMS such as Oracle)	Heavy processing required (many users)
Windows NT	SAS/Share (or DBMS)	Moderate/Light processing (moderate number of users)
Local Client (data on LAN server)	PROC SQL	Moderate/Light processing (need fast network)

Data is accessible via remote libraries or pass-through SQL.

Why SQL Servers?

Fast Processing

SQL processing can be done on the client (PROC SQL), a SAS/Share server or a DBMS (via pass-through SQL). Good performance is achieved in three ways: the processing power of the server is used, the data is processed where the data is located and network traffic is minimized because smaller result sets are returned. Some benchmarks we performed indicated that these activities, when performed on the server, were 10 to 100 times faster than when processed on a local machine accessing remote libraries.

Open Data Access

An open architecture implies that different components of the architecture can be implemented with different technology. By using the industry standard way to access relational data, we can use non-SAS query tools or use different servers such as Oracle Server®.

Using ODBC allows SAS datasets to be made available to non-SAS clients. SAS/Share servers provide this capability with a product SAS/Share*Net®.

Scalability

Using multi-processor servers with software such as SAS/SPDS Server® (Scalable Performance Data Server) or Oracle's multi-threaded servers can achieve scalability.

To further improve performance we support partitioning or replication of our data across servers. Local processing of data is achieved by allowing departmental servers to hold complete copies or subsets of data. The metabase stores the location of *library instances*, libraries that contain these replicated *table instances* and user access information. The *data API* (see below) uses this information to transparently connect users to the appropriate server/library.

What about SAS/Connect?

SAS/Connect® was not used for end-user access for two reasons: security and resource utilization. SAS/Connect sessions require a user login to the remote host, which is not allowed in the client's environment. Resource utilization becomes an issue

because every user that has a remote session was a SAS session running on the server. This causes a lot of resource overhead (e.g. each session consumes about 6MB of virtual memory). With an expected 100-200 users connected to our Unix servers, this technique was thought to cause too much overhead.

DATA MODELS

The Government of Yukon has data in a diversity of subject areas: vehicle registrations, student records, medical records, criminal proceedings, financial information, property management data, liquor sales and worker's compensation information to name a few. While much of this information is departmental in nature a significant amount of data needs to be exchanged amongst the various departments for common purposes. In order to maintain consistency of both departmental and corporate data, modeling information is included in our metabase.

The metabase stores information about *element domains*, *elements*, *libraries*, *tables* and *columns*. Elements are in essence global columns. An element can be implemented in many tables as columns. For example, an element PERPOST (representing the accounting period when a transaction was posted), can exist in many tables as the column PERPOST. An element domain represents the domain of element. For example, an element domain ACCTPER representing an accounting period of which PERPOST is a member. Element domains, elements and columns all share the following information: name, eis/type, type, length, label and format. Consistency is achieved by having columns inherit from elements and elements inherit from element domains. This can be illustrated as follows:

	Name	Label	Format
Element Domain	ACCTPER	Accounting Period	\$ACCTPER.
Element	PERPOST	Posting Period	* inherited
Column	* inherited	* inherited	* inherited

This is fine in theory but it is difficult to have conventions for all data as some departmental data is modeled separately from corporate data. Independent projects cannot always wait for corporate standards and legacy systems data is mostly incompatible with the corporate model. To get around this we

introduced *models*. A model represents a collection of objects (tables, elements etc.) in which naming consistency exists. A model represents a datamart or departmental data. So, how do we integrate these models?

A special model, known as the *corporate model*, is used to store corporate wide element domains and elements. Individual columns within models can inherit elements from their own model or the corporate model. Over time the corporate model will expand and be used by other models extensively.

The information stored within the metabase is used to define the physical attributes of the tables and columns (see *Deployment Metabase Functions* below).

SECURITY

Having many users sharing sensitive information across subject areas poses a serious security problem. How can data be protected and made available in a transparent fashion?

The first problem, securing the host on which the data resides, is accomplished by applying operating system security to lock end-users out of the hosts. The next problem, securing the data, is accomplished by password protecting and encrypting the datasets with sensitive data. Finally, SAS/Share server passwords are hidden from end-users.

So how do we allow access? Access is delivered via *secure SAS views*, which are generated by a *data API*.

* It should be noted that for data that does not require row level security, direct libraries are created to the data source (RLS library to a SAS/Share server or a local library to LAN data). The following discussion applies only where row level security is applied.

The *secure SAS views* are SAS SQL views of two types: inline views and complex views.

* While this system supports DBMS views, SQL pass-through views are illustrated below. The technique for accessing DBMS is similar.

In-line views

In-line views are generated to give a one-to-one mapping of every library/table a user has access to in the data warehouse. The views access native data

Applications and Publishing

warehouse tables via direct or remote libraries (to any number of servers), supply any needed password and apply any required *where* clause. The in-line views are generated via an API call. This happens when a user logs into the *Corporate Warehouse Navigator* (the main application for end-user access - see *deployment* below).

Behind the scenes, a view of the following form is created:

```
Create view library.table (alter=unknownpassword)
as
select columns from
remotelib.table(read=readpassword)
where where-clause
```

where:

<i>Library</i>	the library (located on the local machine in a temporary area - under SASWORK)
<i>Table</i>	the table name
<i>Columns</i>	columns a user can access (or * for all)
<i>Remotelib</i>	the RLS library to the SAS/Share server or library to a LAN server
<i>Readpassword</i>	password needed to access the data tables
<i>Where-Clause</i>	the where clause used to filter the rows to the allowed set
<i>Unknownpassword</i>	the password used to hide the information in view

Complex views

Complex views are generated by calls to the API on an as needed basis. This allows applications and users to take full advantage of the processing power of the server by sending SQL to the server. Unfortunately, the servers are not aware of the security requirements, so the statement must go through the API. The API takes as parameters an arbitrary SQL clause with special tokens representing the tables and a string identifying the tokens and the corresponding tables they represent. It then generates a pass-through view to the data. Calls by applications are done via SCL® or by advanced end-users using macros. A typical application would be

an end-user access tool. A view created from a two-table join would be created in the following form:

```
Connect to remote (remote connect parms)
Create view view (alter=unknownpassword)
as
select * from connection to remote (
select * from
( select columns1 from
remotelib.table1(read=readpassword)
where where-clause1) t1,
(select columns2 from
remotelib.table2(read=readpassword)
where where-clause2) t2
where where-clause
```

where:

<i>Remote Connect Parms</i>	Parameters to connect to the SAS/Share server
<i>Library</i>	the logical library (located on the local machine)
<i>Table1, Table2</i>	the tables names
<i>Columns1, Column2</i>	columns a user can access for table1, table2
<i>Remotelib</i>	the RLS library to the SAS/Share server or local library to LAN server
<i>Readpassword</i>	Password needed to access the data tables
<i>Where-Clause1, Where-Clause2</i>	the where clauses used to filter the rows to the allowed set
<i>Where-Clause</i>	the specified where clause in the query
<i>Unknownpassword</i>	Password used to hide the view info.

DEPLOYMENT

In order to make the information available to end-users, deployment information is stored in the metabase and an application called the *Corporate Warehouse Navigator* was developed to allow users to access data in the metabase.

Deployment metadata includes *subjects*, *infomarts*, *table relationships*, *hierarchies* and the modeling information (described under *models* above).

Infomarts represent programs such as: EIS/reports, saved queries, SAS base code, saved reports or AF applications. Subjects are used to group tables, infomarts and other subjects together. *Business definitions* provide end-user descriptions of subjects, tables and infomarts. Two element/column labels are defined in our metabase; one describes the SAS label and the other describes technical information about a column/element such as the original data source and transformation information. This information is used to exploit the data in the warehouse.

Corporate Warehouse Navigator

The Corporate Warehouse Navigator application is the main application that users use to access their data.

When users log onto the application, secure views to all the libraries/tables are generated and a hierarchical browser (like Microsoft Explorer®) is displayed. The users can explore by expanding the subject area nodes on the browser exposing other subject areas, tables and infomarts. A list of icons to the right of the hierarchical browser represents various SAS and non-SAS tools. Tables can be “dragged” onto the tools to analyze the data and infomarts can be executed by double-clicking on the representing node. Business and technical metadata about the subjects, infomarts and tables can be found by clicking on a node with the right mouse button.

In addition to table security (mentioned above), infomart and subject security controls what subjects and infomarts a user can access.

Deployment Metabase Functions

The metabase supplies several functions related to deployment. These functions help make the development and maintenance of applications much easier because metadata is maintained in one place, the metabase.

Column/Labeling

A macro can be run to create or update datasets with the appropriate column information from the metabase. This function, which uses the inheritance (described under *models* above) produces consistent datasets.

SAS/Query Window Autojoin table

The autojoin table allows the query window application to do automatic SQL joins. A metabase

function creates the autojoin from the relationship information in the metabase.

Building a SAS/EIS metabase

An entire SAS/EIS metabase can be generated for a given *model*. The column labels/formats, EIS attributes and dimension hierarchies, which are stored in the central metabase, are automatically loaded into the EIS metabase.

Metadata reconciliation

The metabase provides several reports for reconciling the metabase with the physical schema.

Documentation

The metabase contents can be printed to provide paper documentation for the warehouse.

Non-SAS Deployment

Front-end tools

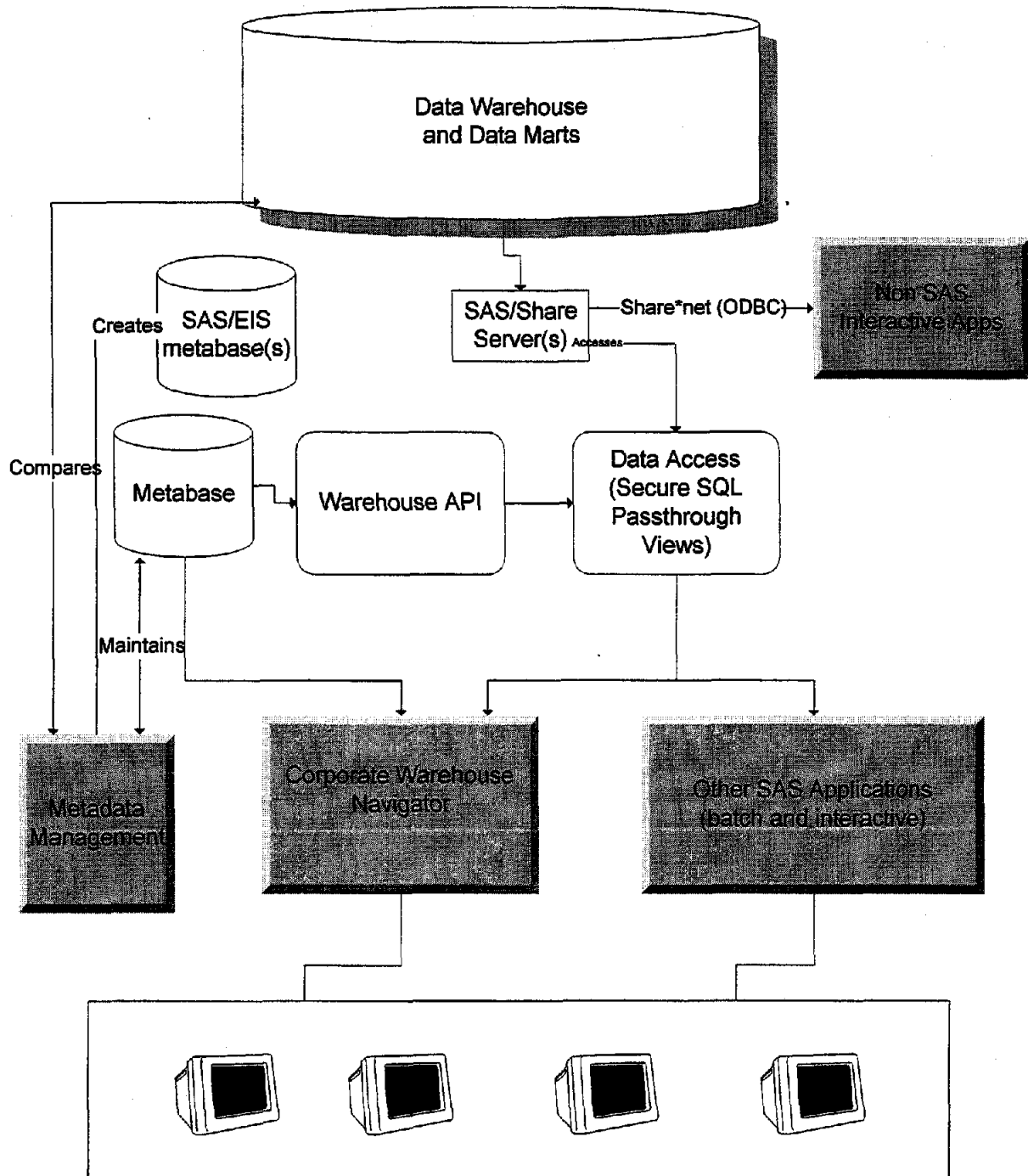
In keeping with an open architecture, third party front-end tools can access the data via ODBC. Unfortunately, the secure view mechanism cannot be used in this case.

An additional method for providing non-SAS application support is via OLE automation. For example the navigator application allows user to drag tables onto a tool called “Excel Spreadsheet”. When this is done, the table is exported to Microsoft Excel® and Excel is launched with OLE calls.

Batch Interface

A Java® based interface has been developed to allow SAS reports to be generated in batch mode. Requests from Java clients are sent to a Java broker which in turn launches SAS with a start-up program. The start-up program specifies the user id, password and report parameters. Secure views are then established in the SAS session (with an API call) and the report is run. The resulting report can be a text file or an HTML document which is e-mailed back to the end-user.

Warehouse Architecture



FUTURE

Security Architecture

Row level security is difficult to implement in an open environment (I have not seen a product that accomplishes this). The security must be handled on servers (or on a middle-ware layer) that does external authentication and governs the rows returned from a query.

Metadata Exchange

The metadata in the metabase could be integrated with other tools such as SAS/Warehouse Administrator™ and modeling tools.

The metabase could be moved to the Microsoft Repository™ to achieve better metadata integration.

Publish/Subscribe

Support for the publish/subscribe paradigm could be used to allow users to exchange data and infomarts in the warehouse. This would likely be implemented on the WEB.

CONCLUSIONS

Architecting a warehousing environment around a centralized metabase and industry standards has brought many benefits. Warehouse administrators, data modelers and developers all benefit because the solution is open, maintainable, secure and scalable. Users benefit by having transparent access to organized and documented information.

CONTACT INFORMATION

Chris Lane, an independent data warehousing consultant, can be reached at: (867) 633-2242 or clane@yknet.yk.ca

ACKNOWLEDGEMENTS

Many thanks to Chris Almstrom, Government of Yukon, for his project facilitation, design and modeling work, to Andreas Marx, SAS Institute Canada Inc., for his ideas and development efforts on the Corporate Warehouse Navigator (built upon his "Rapid Warehouse Toolbox"). Additional thanks to Karen Edwards, SAS Institute (European headquarters) for her modeling help and data warehousing knowledge.

REFERENCES

SAS Institute Inc. (1991), *SAS/Share Software, Usage and Reference, Version 6, First Edition*, Cary, N.C.: SAS Institute Inc.

SAS Institute Inc. (1993), *SAS/Access Interface to ORACLE: Usage and Reference, Version 6, First Edition*, Cary, N.C.: SAS Institute Inc.

Jack Wallace, "New Features of Remote and Open Access to Enterprise Data/ Optimizing Distributed Data Access," *Client/Server computing with the SAS® System: Tips and Techniques (1995)*, Cary N.C.: SAS Institute Inc.

Almstrom, C and Lane, C, "Decision Support Architecture, Government of Yukon" (1996).

TRADEMARKS

SAS, SAS/Access, SAS/Share, SAS/Connect and SAS/SPDS Server, SAS/Warehouse Administrator are registered trademarks of SAS Institute Inc. USA.

Microsoft Windows and Windows Explorer are a registered trademarks of Microsoft Inc.

Java is a registered trademark of Sun Micro Systems Inc.