

A Quick Evaluation of the Options Regarding the Generic Query Interface Support for MIR Service

M. Nedim Alpdemir
04 October 2004

Possible Options

There has been recent discussions regarding the nature of the generic query access mechanism to the MIR via a service-based interface. There are several options to be considered:

1. A generic SQL query interface based on WS-I¹ implementation of OGSA-DAI Grid Data Services (GDS).
2. A Hibernate Query Language (HQL) interface
3. An SQL query interface using simple JDBC connection to the underlying relational database.

Five criteria have been identified for the evaluation of the options given above. Those are the expressiveness of the query language, user familiarity with the query language, structure of the query result format and the compatibility of the query result format with the information model, the amount of development effort required, and the added value in terms of facilitating integration with other services. The options are considered with respect to each criterion in Sections 1 to 5 below. Note that these options are only being considered for *service-based access* to the MIR. It is always possible for a developer to access a MIR database locally via JDBC or Hibernate layer to implement a particular application. However one should be aware of the consequences of using ad-hoc query interfaces, in particular when the interface is used for updating the database. MIR service APIs are designed to preserve different levels of semantics with respect to data manipulation.

1 - Expressiveness of the query language.

There are two distinct options as the query language: SQL and Hibernate Query Language (HQL), which are considered briefly for their expressive power.

SQL

SQL is the most commonly used commercial query language, with the current most widely supported standard being SQL-92 (adopted in 1992). An important core of SQL-

¹ WS-I refers to Web Services Interoperability, which defines the basic set of standards for Web Services such as WSDL, SOAP, UDDI etc.. OGSA-DAI will deliver a WS-I implementation of the GDSs in mid-October 2004, along side the WS-RF implementation.

92 is equivalent to relational algebra² although there are many important features that go beyond what is found in relational algebra such as aggregation (e.g. sums, counts) and database updates. There are also different proprietary dialects developed by different vendors (e.g. T-SQL from Microsoft, DB2 SQL etc). In summary: Along side the basic select-from-where queries, standard SQL supports nested queries, set manipulation constructs (e.g. union, intersect), set comparison operators (e.g. <, >, =>), aggregate operators (e.g. SUM, AVG, MAX, MIN) and grouping constructs (e.g. GROUP BY and HAVING) and various forms of joins (i.e. inner join, right outer join, left outer join, full outer join).

HQL

Queries written in HQL have a very similar syntax to SQL. Inner and all forms of outer joins are supported, as are various functions such as aggregation constructs. HQL also supports many other SQL-like functions and operations such as DISTINCT and LIKE. Sub-queries are also supported if supported by the underlying database. In summary the following features are known to exist:

- Support for SQL functions and operators
- Support for SQL aggregate functions SUM, AVG, MIN, MAX, COUNT
- Support for left/right outer join,
- Full join Support for GROUP BY, HAVING and ORDER BY
- Support for sub-queries (i.e. nested queries) on databases which support SQL sub-selects
- Support for returning arbitrary data objects via the SELECT NEW construct
- Support for polymorphic queries

It appears that as far as standard SQL is concerned HQL roughly matches its expressive power. However, if the MIR was to be implemented in an industrial strength DBMS (such as Oracle or DB2) the vendor specific dialect of SQL would outweigh HQL in many respects.

2 - User familiarity with the query language

SQL is a well established and a well known query language and is used widely, whereas HQL is a relatively new language and it is proprietary. In addition it is object-oriented which means that the user will need to be familiar with the concepts of the object-oriented technology (such as inheritance and polymorphism) to make use of the full power of the language.

3 - Structure of the query result format, and the compatibility of the query result format with the information model.

² Relational algebra consists of some simple but powerful ways of constructing new relations from old ones, by applying relational operators such as union, intersection, selection, projection, join etc.

SQL queries return a list of rows each having a number of columns determined by the select query. Since there is no an automatic marshalling and unmarshalling (serialization / deserialization) support for the JDBC ResultSet object in Apache Axis, the query results need to be transformed into an XML format or a convenient java object (such as Map) for which automatic serialization support exist.

A common XML representation for the result set is Sun's WebRowSet which essentially is a direct mapping of a JDBC result set into an XML document. OGSA-DAI, for instance, used WebRowSet as the result delivery format.

One consequence of using SQL as the query language is that the hierarchical structure of the data in the MIR is flattened.

In their most common form HQL queries return a list of Java objects. The type of the object is determined by the query. HQL queries return the query result(s) in the form of object(s)/tuples of object(s) that are ready to be accessed, operated upon, and manipulated programmatically. This is an advantage over SQL in the sense that the query results maintains the hierarchical structure of the conceptual model (i.e. the information model). Although for local access this eliminates the routine task of creating and populating objects from scratch with the "resultset" retrieved from database queried, for a web service it introduces the task of serializing the potentially hierarchical results into a common format that is easily understood and processed by the web service clients.

4 - The amount of development effort required.

The options given at the beginning of this document are listed below, in an increasing order of required development effort along side a short explanation of the reasons for the costing.

1. A generic SQL query interface based on WS-I implementation of OGSA-DAI Grid Data Services(GDS).

This is the least expensive option in terms of required development because the only cost is to install the OGSA-DAI framework and configure it to wrap the MIR database. The benefit is many other features (such as asynchronous delivery, tunneling the results to remote file stores or other web services etc.) become available along side the ability to submit SQL queries and receive the results. A pure Web-Service based OGSA-DAI implementation is due to be released by mid-October 2004.

2. An SQL query interface using simple JDBC connection to the underlying relational database.

This option involves implementing a JDBC connection to the underlying store. Since the interface is to be exposed via a web service (i.e. the MIR) implementation should probably include support for connection management and pooling to allow for multiple requests to be served efficiently; which effectively replicates what OGSA-DAI provides.

3. A Hibernate Query Language (HQL) interface

The development cost of implementing an HQL interface is probably very close to the second option. Although connection pooling and management would be handled by the Hibernate layer, there is an additional design and implementation cost of result management (e.g. cost of result serialization to XML).

5 - Added value in terms of facilitating integration with other services

One aspect of the query interface worth considering is the extent of support it provides for other services or systems to seamlessly integrate. Among the options being considered only OGSA-DAI defines access patterns to retrieve data and metadata (i.e. database schema) in a well-defined format and interaction patterns with other components to, for example, send the results to other services, stream the results in blocks asynchronously, store the results locally or remotely to file systems etc. Although similar patterns can be implemented for other options, clearly this would be re-inventing the wheel and lead to unnecessary duplication of effort.

Evaluation and Conclusion.

As far as the query language choice is concerned, the user familiarity and expressive power are two factors that promotes the usage of SQL (particularly in the case of industrial strength DBMS back-ends), although one could argue that HQL's expressiveness would be adequate for most of the cases and for some cases (e.g. polymorphic queries) HQL could offer a more powerful alternative, thus making the latter factor (i.e. expressive power) a relatively less important one. As another dimension on the query language issue; although HQL is more advantageous considering the query result format since it preserves the structure of the information model, it may introduce additional management and shipment cost if the query interface is designed to allow for configuring the level of deepness of the object hierarchy. If, on the other hand, an HQL query result is shipped only with one level of hierarchy (i.e. flat) with object references as LSIDs (to reduce the management and shipment cost), then it is not very different from an SQL result set. These considerations causes one to be inclined towards SQL as the query language, thus eliminating the second option (i.e. HQL based interface) as a viable alternative.

Considering the development cost / benefit ratio of the remaining two options (i.e. OGSA-DAI and JDBC based interfaces) as outlined in Section 4 together with the expected delivery time of WS-I implementation of OGSA-DAI; and the comparative superiority of an OGSA-DAI-based interface with respect to the added value in terms of facilitating integration with third parties as outlined in Section 5, OGSA-DAI option appears to be the most appropriate one. It is worth noting that OGSA-DAI implementation comes with a relatively mature and user-tested interface design and rich documentation. OGSA-DAI will also going to be fully integrated with OMII as it is part of the first release of OMII services.