1 Overview

This note describes the thought process the registry team has been through on the subject of ad hoc query support. It discusses the alternatives that have been considered and finally makes a recommendation to the team on how to support ad hoc queries in the registry.

1.1 Definitions

This section defines some terminology that will be used in subsequent section.

1.2 Focused Query

A focused query is one where the query interface is fixed to do a specific query task. A focused query approach pre-supposes what the client would wish to wish to search for. Focused queries are defined statically by the Registry interface. Focused queries are analogous to calling statically defined methods that have a fixed signature.

The current “browse and drill down” queries of the Registry are examples of focused queries. The UDDI find_business_by_name query is another example of a focused query.

1.3 Totally Ad Hoc Query

An totally ad hoc query is a completely open ended query where no a priori knowledge exists on what the client would wish to search for nor is there any prior knowledge about the schema or information model upon which the queries would be based. Totally ad hoc queries allow for complex queries to be composed of simpler predicates based logical operations (e.g. AND, OR, NOT) and allow for comparison operations (e.g. >, <, = etc.) as well as string pattern matching. Totally ad hoc queries also allow for unrestricted joins across different schema types. Totally ad hoc queries have the potential for excessive utilization of system resources.

SQL queries in relational databases, XPATH and XML Queries on XML content are examples of totally ad hoc queries.

1.4 Constrained Ad Hoc Query

A constrained ad hoc query is somewhere in between focused query and totally ad hoc query. Constrained ad hoc queries operate against a fixed schema that is known a priori. They are similar to focused queries in the sense that they only allow certain allowed predicates. They are similar to totally ad hoc queries in the sense that they allow for complex queries to be composed of simpler predicates based logical operations and allow for comparison operations. Constrained ad hoc queries also allow for restricted joins across different schema types.

1.5 Content Based Queries

Typical Registry query requirements search for content based on metadata submitted on the content as defined by Registry Information Model [RIM].

A query mechanism that supports the ability to search for content based on data that is part of submitted content is referred to as supporting content-based queries.

Note that Content Based Queries are very similar in nature to totally ad hoc queries.

2 Registry Query Requirements

The Registry schema is fixed by the information model. As such totally ad hoc queries are not required for the registry. That leaves us with having to choose between constrained ad hoc queries (hitherto referred as *ad hoc queries*) or focused queries.

The following sections describe various requirements that have been considered. Each is rated with its relative importance based on a High, Medium, Low scale.

2.1 Ease of Use By Clients (High)

The single most important requirement for query support for the registry IMHO is that the query interface must be simple to use from the perspective of Registry clients. This is based on the fact that there are many more registry clients than there are registry services.

2.2 Support For Constrained Ad Hoc Query (High)

Recall that the analysis left us with having to choose between Constrained Ad Hoc Query and focused queries. Focused queries have the following problems:

1. They pre-suppose what the client will wish to query since they do not have the ability to build complex queries from simpler predicates.

2. They are a maintenance headache since the Registry must support a separate interface for each supported query
3. They are not easy to use because the client has to remember separate query syntaxes for each query.

Based on the reasons above, a constrained ad hoc query is recommended as a requirement for Registry query interface.

2.3 Content Based Query Support (High)

IMHO, the query mechanism must also allow for the ability to search for content based on data that is present in the content itself. Content-based query requirement does not imply a specific approach. It is simply stating that it is a valid use case regardless of how we choose to support it. Consider the need to search for all Capps that include “Seller” as a specified role. This use case is fairly common and was part of the Tokyo POC scenario. Other use cases of content-based queries abound.

2.4 Based on Existing or Emerging Standards (High)

It is desirable to have the query interface be based upon an existing or emerging standard for obvious reasons.

2.5 Ease of Acceptance by ebXML Membership (High)

This requirement is pragmatic and not technical. We need to choose a query interface that will meet the most important requirements and still be palatable to the membership of ebXML.

2.6 Flexibility of Implementation In Registry (High)

It is important that whatever query interface we define, it allows Registry implementers to implement the Registry in any technology they see fit (e.g. relational database, object database, LDAP, flat files etc.)

2.7 Ease of Implementation In Registry (Medium)

IMHO, ease of implementation by clients is the highest requirement. However, it is also important to make sure that whatever query interface we define, it does not place an excessive burden on Registry implementers. IMHO, it is OK for us to accept some complexity (within reason) in Registry implementation vs. improved ease of use by the clients.

3 Alternatives For Constrained Ad Hoc Query Interface

Based on the rationale given above, IMHO we need to agree on a suitable constrained ad hoc query mechanism. The following candidate choices have been considered:

3.1 Constrained Sub-set of an Existing Query Language

In this approach we leverage an existing query language standard or emerging standard to describe the query. Alternatives considered have been a constrained sub-set of OQL and a constrained sub-set of XPATH. In either case the client would specify the query as a string attribute named queryString in a SubmitAdhocQueryRequest XML message to the Registry. The Registry must be able to process the query string and map it to an equivalent query in its underlying implementation (e.g. SQL for relational databases). This requires limited ability to parse and recognize predicates that match pre-defined predicate patters in the Registry Services [RS] document. This does not require having to implement a full-blown OQL or XPATH processor since it does not need to work for any query but only queries whose predicates match certain specified patterns. IMHO, in the scheme of things this is of moderate complexity with constrained OQL being easier to process than constrained XPATH.

As an aside it should be noted that XML Query was considered but discarded based on our requirements and the fact that it is not yet a

This is the alternative that has been discussed most in Registry group and the recent face to face meeting where an entire day was spent considering XPATH alternative and experts such as Mike Rowley were invited to guide us.

3.2 Simple XML Schema To Express Queries (Custom Query Syntax)

In this approach (referred to as custom query syntax) we specify an XML schema in the [RS] that is capable of representing the constrained ad hoc query in form of an XML document. This is essentially what NIST and OASIS have done in their Registry.

Regardless of the specific XML schema we choose in this approach, it has the perceived benefit of being simple to implement by registry implementers. It has the disadvantage that the query syntax is not as simple and familiar for Registry clients. It is also proprietary and not based on any standard. Query interfaces are non-trivial to design and get right. It is a very big wheel to re-invent even if we constrain it. I am speaking here from past experience.

Based on responses on the Registry mailing list this option has the most vocal support.

4 Recommendation

[Note] `Note that this is my personal recommendation and reconstruction of the thought process we have followed. It is humbly submitted as an individual. Sun has no articulated or hidden agenda or position on this topic (or any other Registry related topic) other than that we should reach a decision on this subject as a team following due process.`

I believe the most important thing is to have Registry support for constrained ad hoc queries based on an interface that is simple to use by Registry clients.

4.1 Transition From OQL To XPATH

Until day one of the face-to-face I believed that constrained OQL was the best alternative. I had considered XPATH but found it to be less simple from a client's perspective and also likely to force a Registry implementation where metadata was stored as XML content rather than a relational database. However, being sincere in my desire to explore all reasonable alternatives, I invited several experts on XPATH and XML Query to day 1 of the face-to-face. We spent an entire day on exploring XPATH.

At first it seemed that XPATH would not be able to deal with our most complex queries (classification queries). Mike Rowley then came up with a breakthrough that alleviated my concerns regarding XPATH alternative. The idea was to design a virtual document schema mapping from [RIM] that would be more suitable for simple XPATH queries. I liked the idea from that point since XPATH syntax also allowed for content-based queries. However, there were the following concerns:

1. Can a constrained XPATH express all our pre-defined query predicates?
2. Can a constrained XPATH be mapped to a relational query relatively easily?

I spent the weekend exploring the first issue and posted the results to the list. They were encouraging in that XPATH seemed to have the expressive capability to meet our query needs. I was planning on exploring the second issue early this week and sharing the findings with the team.

4.2 Transition Back From XPATH To OQL

Two things happened that led my thinking away from XPATH and back to OQL as the syntax for expressing constrained ad hoc queries.

4.2.1 XPATH Mapping To SQL Is Harder

As I began to think through the issue of mapping constrained XPATH queries to a relational database it became apparent that this would be more difficult than mapping OQL queries to SQL (the latter is a relatively simple Object-Relational mapping that can be applied algorithmically).

4.2.2 Breakthrough On Content Based Queries

It occurred to me that the best way to deal with content-based queries is to use the existing classification based queries along with an automatic indexing capability. The idea is to define logical indexes as a binding between an element or attribute in the document schema with a classification scheme when the schema is submitted. Thenceforth, whenever a document instance for the schema is submitted, it is automatically classified by the Registry. This approach has the following advantages:

1. Utilizes existing classification scheme support that can be highly optimized by Registry implementations
2. Does not require searching on document content in response to queries
3. May be applicable to non-XML content as long as Registry knows how to process it at submission time

This breakthrough idea makes XPATH less interesting to me as a query language syntax since we no longer need to search document instances in response to queries.

4.3 What About Custom Query Syntax Idea?

I have already expressed the issues I see with this alternative. As always I will demonstrate an open-minded approach to this alternative if the team feels this is the approach we should pursue. If we decide to go this path as a team decision then I will devote all my focus to this option.

BTW Mike Rowley and I spent a good amount of time today exploring this option. The complexity of specifying an adequate XML syntax became apparent after about an hour. We will continue to explore this option given the support it has received on the mailing list.

5 Summary of My Position

I believe that a constrained OQL query syntax is our best alternative (See Appendix A for examples). It meets most of the requirements we have identified:

1. It is trivial and familiar to use from a clients perspective.
2. It has strong expressive power to support constrained ad hoc queries

3. It is based on a dominant existing standard (SQL) and a not so dominant yet existing standard (OQL)
4. It can be implemented with relatively moderate complexity. I have offered to explore with Sun to donate such a query processor to implementers of ebXML Registry.
5. It offers flexible choices to Registry implementers since they can use a commercial relational or object database for storing metadata.

This approach may not meet the following requirements:

1. Ease of Acceptance by ebXML Membership

With my position and thought process on the table I invite your thoughts and comments and commit to you that I will support whatever we as a team decide on this and all other Registry issues.

Thanks for your consideration.

Humbly submitted,

Farrukh Najmi

Appendix A Sample Constrained OQL Queries

Following is a cut-and-paste from earlier [RS] document.

5.1.1 Simple Meta Data Based Queries

The simples form of ad hoc queries are based upon metadata attributes specified for Registry content as specified in [RIM]. This section gives some examples of simple metadata based queries. The queries may use any accessor (get) method specified for an interface in [RIM].

For example, to get the collection of Objects that match a specified name and have version greater than 1.3, the following query must be supported:

```
//Get Objects whose name includes the word bicycle
SELECT DISTINCT obj FROM Object WHERE
        obj.name LIKE ‘%bicycle%’ AND
        obj.majorVersion >= 1 AND obj.minorVersion >= 3;
```

Note that simple queries such as above where all predicates are based on primitive attribute types may be passed to an SQL processor *as is* if the implementation is SQL.

5.1.2 Classification Queries

This section describes the various classification related queries that must be supported.

5.1.2.1 Getting Objects Classified By a ClassificationNode

To get the collection of Objects classified by specified ClassificationNodes the following query must be supported:

```
SELECT DISTINCT obj FROM Object WHERE
        obj.isClassifiedBy(<classificationNode>);
```

For example to get the collection of Objects classified by “Automotive.Industry” node and the “Geography.Asia.Japan” node the query would be:

```
 SELECT DISTINCT eo FROM ExtrinsicObject WHERE
        eo.isClassifiedBy(ClassificationNode(path: “Industry.Automotive”)) AND
        eo.isClassifiedBy(ClassificationNode(path: “Geography.Asia.Japan”));
```

Note that AdHocQueryResponse will include a ManagedObjectList which will include heterogeneous elements (e.g. ExtrinsicObjects, Classification etc.) representing the classes specified in [RIM].

5.1.2.2 Getting ClassificationNodes That Classify an Object

To get the collection of ClassificationNodes that classify a specified Object the following query must be supported:

```
SELECT cn FROM ClassificationNode WHERE

        cn.classifies(<object’s id>);
```

5.1.3 Association Queries

This section describes the various Association related queries that must be supported.

5.1.3.1 Getting All Association From Specified Object

To get the collection of Associations that have the specified Object as its source, the following query must be supported:

```
SELECT assoc FROM Association WHERE

        assoc.sourceObject = <uuid>;
```

5.1.3.2 Getting All Association To Specified Object

To get the collection of Associations that have the specified Object as its target, the following query must be supported:

```
SELECT DISTINCT assoc FROM Association WHERE
        assoc.targetObject = <uuid>;
```

5.1.3.3 Getting Associations Based On Name, Role, Type

To get the collection of Associations that have specified Association attributes, the following queries must be supported:

The following query selects Associations that have the specified name:

```
SELECT DISTINCT assoc FROM Association WHERE
        assoc.name = <name>;
```

The following query selects Associations that have the specified source role name:

```
SELECT DISTINCT assoc FROM Association WHERE
        assoc.sourceRole = <roleName>;
```

The following query selects Associations that have the specified target role name:

```
SELECT DISTINCT assoc FROM Association WHERE
        assoc.targetRole = <roleName>;
```

The following query selects Associations that have the specified association type, where association type is a string containing the corresponding field name described in [RIM].

```
SELECT DISTINCT assoc FROM Association WHERE
        assoc.associationType = <associationType>;
```

5.1.3.4 Complex Association Queries

The various forms of association queries may be combined into complex predicates. The following query selects Associations from object with uuid sourceObj that have the sourceRole “buysFrom” and targetRole “sellsTo”:

```
SELECT DISTINCT assoc FROM Association WHERE
```

```
        Assoc.sourceObject = “sourceObj”
        assoc.sourceRole = “buysFrom” AND
        assoc.sourceRole = “sellsTo”;
```

Note that simple queries such as above where all predicates are based on primitive attribute types may be passed to an SQL processor *as is* if the implementation is SQL.

5.1.4 Package Queries

To find all packages that a specified object belongs to, the following query is specified:

```
SELECT DISTINCT pkg FROM Package WHERE
        pkg IN obj.getPackages();
```

To find all objects in a specified package, the following query is specified:

```
SELECT DISTINCT obj FROM Object WHERE
        obj IN pkg.getMemberObjects();
```

5.1.4.1 Complex Package Queries

The following query gets all packages that a specified object belongs to, that are not deprecated and name contains RosettaNet.

```
SELECT DISTINCT pkg FROM Package WHERE
        pkg IN obj.getPackages() AND
        pkg.name LIKE ‘%RosettaNet%’ AND
        pkg.status != ‘DEPRECATED’;
```

5.1.5 ExternalLink Queries

To find all ExternalLinks that a specified object is linked to, the following query is specified:

```
SELECT DISTINCT link FROM ExternalLink WHERE
        link IN obj.getExternalLinks();
```

To find all objects that are linked by a specified ExternalLink, the following query is specified:

```
SELECT DISTINCT obj FROM Object WHERE
        obj IN link.getLinkedObjects();
```

5.1.5.1 Complex ExternalLink Queries

The following query gets all ExternalLinks that a specified object belongs to, that are contain the word ‘lega;’ in their description and have a URL for its externalURI.

```
SELECT DISTINCT link FROM ExternalLink WHERE
        link IN obj.getExternalLinks() AND
        link.description LIKE ‘%legal%’ AND
        link.externalURI LIKE ‘%http://%’;
```

5.1.6 Audit Trail Queries

To get the complete collection of AuditableEvent objects the following queries are specified:

```
SELECT DISTINCT ev FROM AuditableEvent WHERE
        ev IN obj.getAuditTrail();
```

5.1.7 Queries Involving Restricted Joins

Some queries may involve a restricted join between more than one class. For example the following query gets all Objects that have been deprecated and are members of a package whose name contains the word ‘Acme’.

```
SELECT DISTINCT obj FROM ManagedObject, Package WHERE
        obj.status == ‘DEPRECTAED’ AND
        exists pkgs IN obj.getPackages() : pkgs.name LIKE ‘%Acme%’
```

A.1 Mapping OQL To Relational Implementations

While this is an implementation detail, this section is added to explain why I feel that OQL is the easiest option to map to relational queries. The constrained OQL I propose is a minor extension to a subset of SQL and therefore most closely related to SQL.

Note that I do not feel that the XPATH or homegrown XML syntax approaches would be easy to map to SQL at all. The reason I emphasize ease of a relational implementation is simply because this is the chosen implementation in all registry implementations I am familiar with.

A.1.1.1 Mapping of Predicates Involving Primitive Attributes

Most of the RIM interfaces methods are simple get/set methods that map directly to primitive attributes. For example the getName() and setName() methods on Object map to a name attribute of type String. In the OQL option such predicates may be passed as is to the database as a SQL query (e.g. name LIKE '%Acme%').

A.1.1.2 Mapping of Predicates Involving References

A few of the RIM interface methods return refernces to objects (e.g. Object#getAccessControlPolicy()). In such cases the references map to the ID attribute for the referenced object. This is again a special case of a primitive attribute mapping. In this case the ID attribute is used as a foreign key to reference a row in another table.

A.1.1.3 Mapping of Predicates Involving Complex Attributes

A few of the RIM interface methods return objects that are complex types (e.g. Organization#getPostalAddress()). In such cases the complex type may be flattened in the relational schema. For example the address attribute could be mapped to several primitive columns such as address_city. Again this becomes a special case of the primitive type predicate case. The OQL processor would simply do the appropriate flattening and then pass the resulting simple SQL query to the database.

A.1.1.4 Mapping of Predicates Involving Relationship Methods

Many of the RIM interface methods are relationship methods involving many-to-many relationships (e.g. ManagedObject#getPackages()). In such cases the OQL processor would define an implementation method that maps the relationship method to a corresponding SQL query. The resulting SQL query would be passed to the database.

A.1.1.5 Mapping of Predicates Involving Joins

Since the OQL processor would have the ability to converts individual OQL predicates to SQL it would be able to compose a complex SQL query from the results of the mapping. Such complex queries would be a subset of standard SQL92 and will have all the expressive power of SQL. This includes the ability to do simple joins involving more than one table (RIM interface). The result is a very powerful capability to do queries involving simple joins.

Appendix B Sample Constrained XPATH Queries

Following is a cut-and-paste from an unreleased version of the [RS] document. Note a more thorough analysis with working sample data and queries has been posted to the list by me earlier.

5.1.8 Simple Meta Data Based Queries

The simplest form of ad hoc queries are based upon metadata attributes specified for Registry content as specified in [RIM]. This section gives some examples of simple metadata based queries.

For example, to get the collection of ExtrinsicObjects whose name contains the word ‘Acme’ and have version greater than 1.3, the following query predicates must be supported:

```
/ExtrinsicObject[contains(@name,'Acme') and
        (@majorVersion = 1 and @minorVersion > 3) or
        (@majorVersion > 1) ]
```

Note that the query syntax allows for conjugation of simpler predicates into more complex queries as shown in the simple example above.

5.1.9 Classification Queries

This section describes the various classification related queries that must be supported. Classification queries operate on virtual documents where the root is a ClassificationNode.

5.1.9.1 Getting Root Classification Nodes

To get the collection of root ClassificationNodes the following query predicate must be supported:

```
/ClassificationNode[@parent='']
```

The above query returns all ClassificationNodes that have their parent attribute set to null. Note that the above query may also specify a predicate on the name if a specific root ClassificationNode is desired.

5.1.9.2 Getting Children of Specified ClassificationNode

To get the children of a ClassificationNode given the ID of that node the following query predicate must be supported:

```
//ClassificationNode[@parent='/ClassificationNode/Geography']
```

The above query returns all ClassificationNodes that have the URI for the Geography node as their parent attribute.

5.1.9.3 Getting Objects Classified By a ClassificationNode

To get the collection of Objects classified by specified ClassificationNodes the following query must be supported:

```
/ExtrinsicObject[

./ClassificationNode[@name='Industry']/ClassificationNode[@name='Automotive'] and
./ClassificationNode[@name='Geography']/ClassificationNode[@name='Asia']]
```

The above query gets the collection of ExtrinsicObjects that are classified by the Automotive Industry and the Japan Geography. Note that the query will also contain any objects that are classified by descendents of the specified ClassificationNodes.

5.1.9.4 Getting Associated Objects Based On Association Attributes

To get the collection of ExtrinsicObjects that are associated with a specified objects based on an Association with specified, the following queries must be supported:

```
/ExtrinsicObject[./AssociationList/Association[

        @associationType='CONTAINED_BY' and

        @name='foo'

        @sourceObjectRef='/partyProfiles/Acme_PartyProfile']]
```

The above query selects ExtrinsicObjects that are associated with the object specified by the sourceObjectRef with an Association that has an assoociationType of CONTAINED_BY and a name of ‘foo’.