OpenFurther Reference Documentation

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 About

The following documentation applies to **OpenFurther version 1.4.0-SNAPSHOT**

1.1 Conventions

Note
A note

Important
An important point

Tip
A tip

Warning
A warning

Caution
A point of caution

2 Introduction

OpenFurther is an informatics platform that supports federation and integration of data from heterogeneous and disparate data sources.

It has been deployed at the University of Utah (UU) as the Federated Utah Research and Translational Health e-Repository (FURTHeR) since August 2011 and is available for use by all U of U employees and students. OpenFurther links heterogeneous data types, including clinical, public health, biospecimen and patient-generated data; empowering researchers with the ability to assess feasibility of particular clinical research studies, export biomedical datasets for analysis, and create aggregate databases for comparative effectiveness research. With the ability to link unique individuals from these sources, OpenFurther is able to identify cohorts for clinical research.

It provides semantic and syntactic interoperability as it federates health information on-the-fly and in real-time and requires neither data extraction nor homogenization by data source partners, facilitating integration by retaining data in their native format and in their originating systems.

OpenFurther is built upon Maven, Spring, Hibernate, ServiceMix, and other open source frameworks that promote OpenFurther's code reusability and interoperability.

3 Architecture

Loosely, OpenFurther runs as a multi-tier application. The presentation layer or front end/user-interface is served (currently) through the i2b2 web client. The logic layer is served through the ServiceMix ESB, and the database layer is served using Oracle 11g, although it can be configured for other databases as well.

3.1 User Interface

OpenFurther utilizes the i2b2 web client as a front-end for querying data. The user interface has been modified to support federated querying.

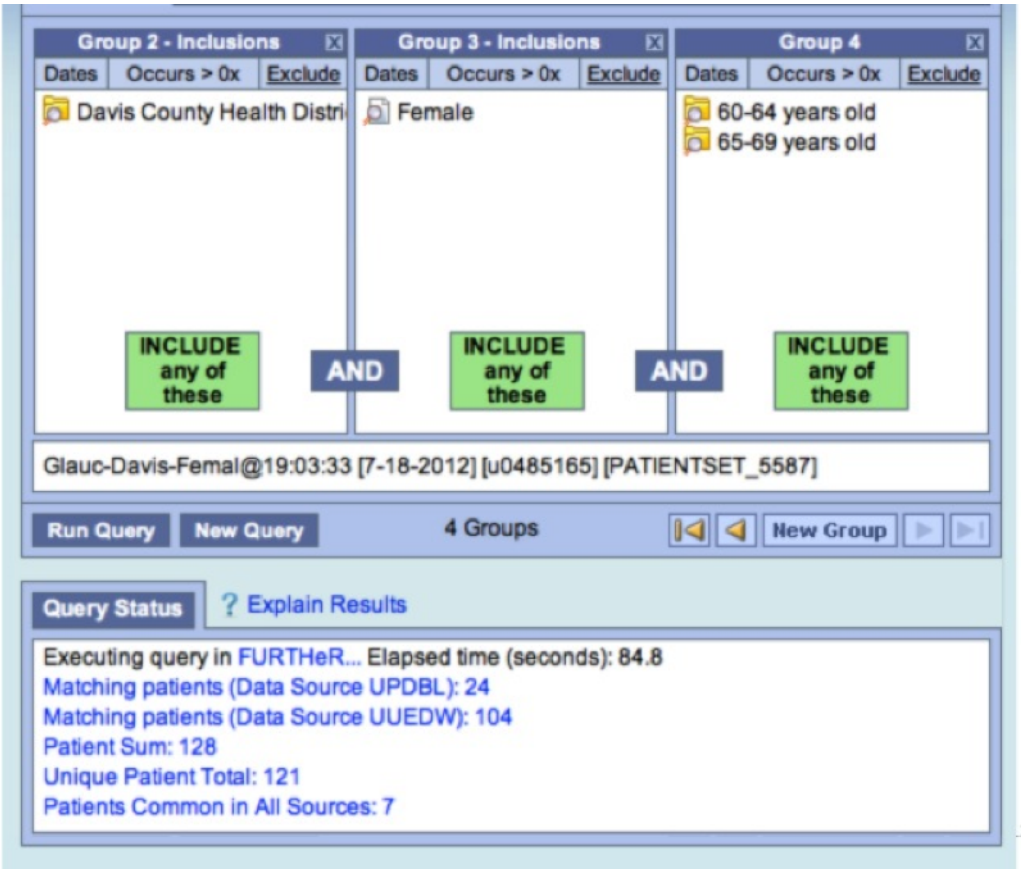

Figure 1: Customized i2b2 User Interface

3.2 Hooking OpenFurther into i2b2

OpenFurther utilizes a Java Servlet Filter to divert query requests to the OpenFurther backend system. The Servlet filter looks for XML messages from i2b2 that indicate a query is being run. Those XML messages are then diverted to OpenFurther where OpenFurther converts them into a OpenFurther query and runs them. All other XML messages are ignored and i2b2 is allowed to run as normal.

Note
No data is stored within i2b2, all data resides within its original location

---

3.3 The Federated Query Engine (FQE)

In OpenFurther, the term "FQE" (Federated Query Engine) is broadly referred to as the set of software modules involved in the execution of a federated query.

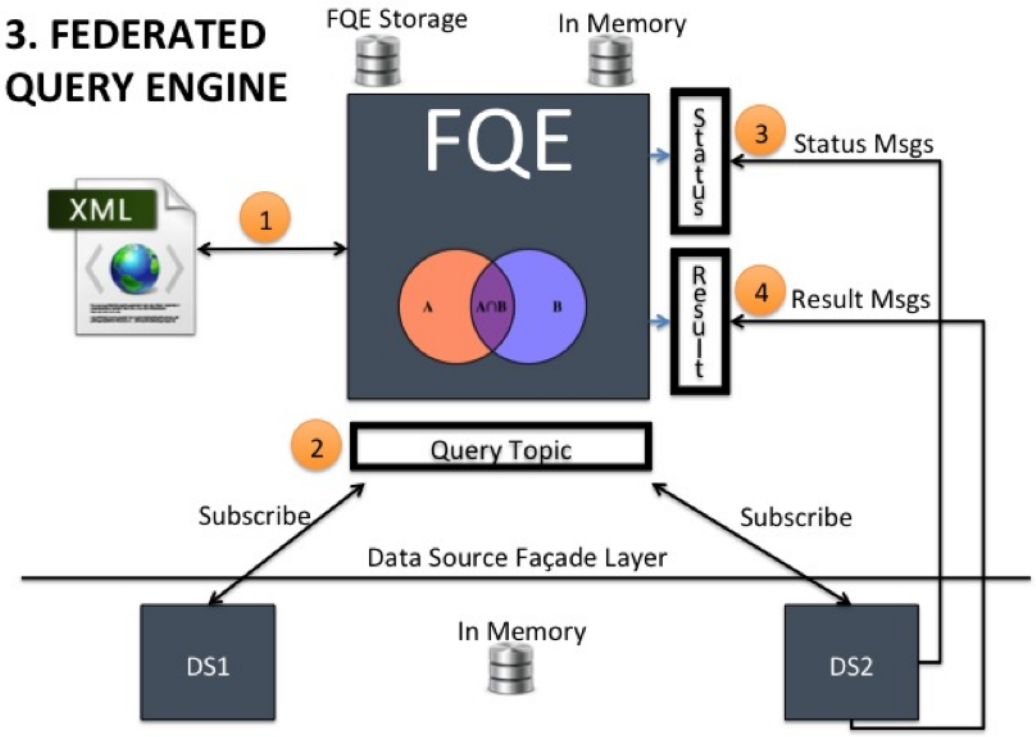

Figure 2: Federated Query Engine

- A federated query written in FQL (an XML based query language) or an i2b2 query is submitted at **1**.
- Utilizing the publish-subscribe pattern, one or more data source adapters are subscribed to the Query Topic at **2**.
- If the query is an i2b2 query, the FQE converts the i2b2 query to a federated query.
- The FQE then posts the query to the Query Topic (**2**) and each listening data source adapter receives a copy of the query.
- Each data source adapter runs through a number of steps to initialize, process, and translate a query for a given data source (Explained below).
- Throughout the processing, status messages are sent to a Status Queue at **3**.
- Once results are translated to a common model, they are persisted to the In-Memory database and result count is sent to **4**.

3.4 Data Source Adapters

Data source adapters are facades around an existing data source. Data source adapters can be entirely custom for any given implementation or they can use a pre-written adapter if their data source is already in a well-known format such as OMOP, i2b2, OpenEMR, etc

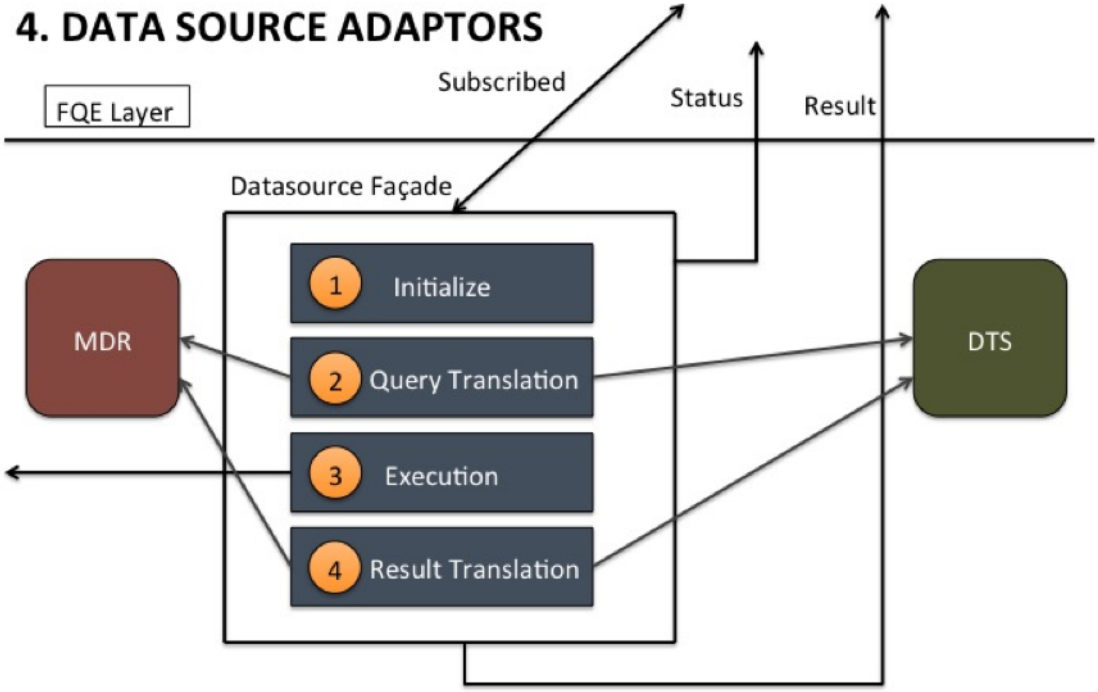

Figure 3: Data Source Adapters

- Data source adapters follow the chain-of-responsibility pattern. The process of adapting a query is broken down into several small steps and each output is passed on to the next step. Data source adapters typically have 4 commons steps.

  1. They are given an initialization step which allows them to determine whether or not the given data source can answer the given query. It also provides for any other initialization required throughout the process.
  2. Query translation translates the logical FQL that is not specific to any data source into data source specific language. This will vary with data sources. Some data sources will utilize SQL, other's might be a web service. It utilizes the Metadata Repository (MDR) for translating attributes and values (e.g. logical query uses Gender but actual data source uses Sex as the attribute). It also utilizes DTS (Terminology Server) to translate from a given code (e.g. ICD9 250) to the data source's code (e.g. 12345)
  3. The query is executed against the data source and results are returned in their native format (SQL ResultSet, XML, etc).
  4. Result translations translates the results into a common model with standardized vocabulary/terminology utilizing the Metadata Repository (MDR) and DTS (Terminology Server).

3.5 Terminology Server

OpenFurther utilizes Apelon's Distributed Terminology System version 3.5.2.203 (aka. DTS) for terminology related functionality. The OpenFurther instance of DTS contains concepts from the standard terminologies SNOMED-CT, ICD-9, RxNorm, and UCUM. There are also non-standard terminologies (aka Local) for each of the data sources as well as associated mappings. The use of standard terminologies and mappings make it possible for the software to resolve differences between concepts in various data sources and achieve a degree of semantic interoperability. Use of Apelon DTS is an assumption of agreement to the Apache Version 2 standard open source license agreement http://www.apache.org/licenses/LICENSE-2.0.html. For more information about Apelon DTS please see their website http://www.apelon.com.

3.5.1 Features of Apelon DTS

The Apelon DTS (Distributed Terminology System) is an integrated set of open source components that provides comprehensive terminology services in distributed application environments.

DTS Supports national and international data standards, which are a necessary foundation for comparable and interoperable health information, as well as local vocabularies.

DTS consists of

- DTS Core - the core system, database, api, etc
- DTS Editor - a GUI interface for viewing, adding, and editing concepts
- DTS Browser - a web interface for viewing concepts
- Modular Classifier - allows for extending standard ontologies

3.6 The Metadata Repository (MDR)

The MDR is responsible for storing information (artifacts) about varying data sources. This includes things like data models, attributes, attribute types, etc. It is accessed using web services.

- Home grown but follows standards
  - XMI, Dublin Core
  - HL7 datatypes, CDA, DDI
- Stores artifacts
  - Logical models (UML), local models (UML), model mappings
  - Administrative information
  - Descriptive information
- Models supported
  - OMOP, i2b2, local models

4 Terminology

4.1 Getting Started

In order to utilize the OpenFurther software, it is necessary to have terminology mappings from your desired data sources to standard terminologies. These standard codes are then translated via the software, terminology server, and associated mappings to be able to resolve to a local data source's codes/terms.

Important
It is important to note that the content distributed with OpenFurther is for demonstration purposes only. The standard terminologies have been provided with permission via Apelon's distribution of free subscription content available on their open source website http://apelon-dts.sourceforge.net/. This standard content is several years out of date and would not be the most suitable for a real world instance.

Tip
It is recommended for organizations that desire to use the OpenFurther software to consider resourcing a dedicated terminologist or someone that has experience with controlled vocabularies and ontologies to work on managing/mapping local vocabularies/codes to their specific implementation of OpenFurther.

Apelon provides a content delivery subscription service at a reasonable cost. Standard terminologies can also be downloaded from the U.S. National Library of Medicine Unified Medical Language System (link: UMLS) after meeting and accepting their requirements and license agreements.

Note
The local vocabularies have been mapped to the best possible matches to the available standard terminologies. However, in some cases such as OpenMRS, local concepts had to be created to fit the OpenFurther demonstration scenario. Any creation of local concepts was done in best accordance of the specifications provided by the source.

OpenFurther's i2b2 front end user interface contains an ontology based off of the recommendations of the Healthcare Information Technology Standards Panel (HITSP). For instance, HITSP recommends the use of ICD-9 codes for diagnosis and LOINC for laboratory data. Please note that because of licensing agreements, not all of the HITSP recommendations could be followed for OpenFurther. For example, HITSP recommends the use of CPT for procedures. In OpenFurther, procedures will be based off the SNOMED CT hierarchy for procedures.

4.1.1 Why are mappings needed?

Mappings are needed because of the variations in terminology used between disparate data sources. Mappings equate concepts that are intended to mean the same thing.

Tip
Mapping can be a very human labor intensive task. Mappings must be verified and tested to ensure quality of results. Involving subject matter experts and collaborating effectively across datasources will be paramount to achieving a successful implementation of terminology.

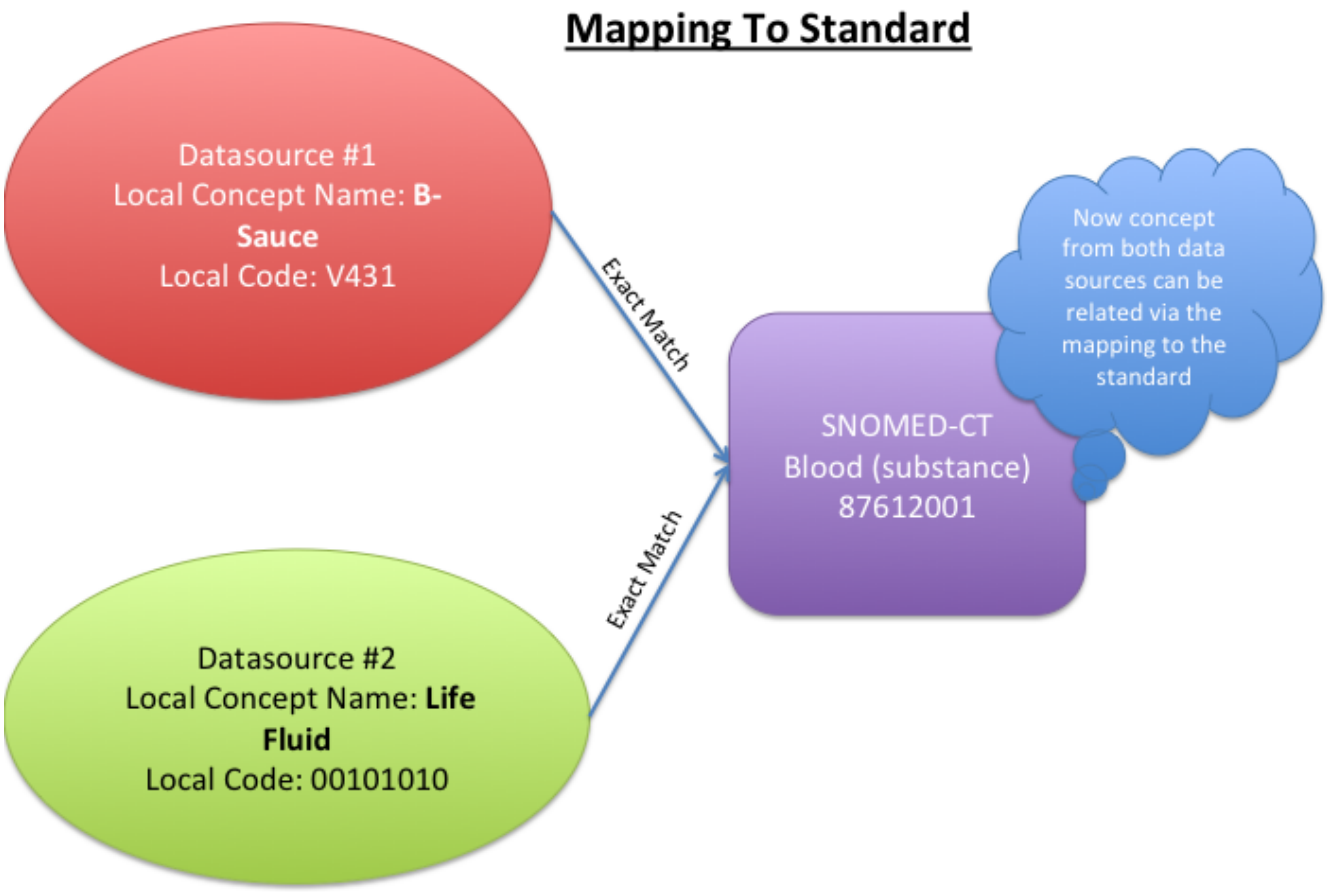

Figure 4: Mapping Terminology

4.1.2 Initial Steps

Apelon DTS provides excellent documentation and examples of how to use their terminology server software. All Apelon documentation can be found at: http://apelon-dts.sourceforge.net/documents.html

Important
It is highly recommended that you familiarize yourself with the basic use of the Apelon DTS software. The instance included in OpenFurther can serve as an example of how the OpenFurther team has used Apelon DTS but the best instruction on how to use Apelon DTS is provided directly from Apelon.

Local Namespaces

Refer to page 62 of the Apelon DTS Editor documentation.

Authorities

Refer to page 72 of the Apelon DTS Editor documentation.

Association Types

Refer to pages 75-77 of the Apelon DTS Editor documentation.

Association Qualifier Types

Refer to pages 80-84 of the Apelon DTS Editor documentation.

Property Types

Refer to pages 94-96 of the Apelon DTS Editor documentation.

Property Qualifier Types

Refer to pages 99-101 of the Apelon DTS Editor documentation.

Adding new concepts/terms, assign properties, assosciations/mappings

Refer to pages 119-141 of the Apelon DTS Editor documentation.

Bulk loading and working with spreadsheets

Refer to the import wizard plugin user guide

5 Metadata Repository (MDR)

5.1 Getting Started

Two important parts of the metadata repository are Query Translation and Result Translation. Data stored within the MDR is used to drive each of these processes.

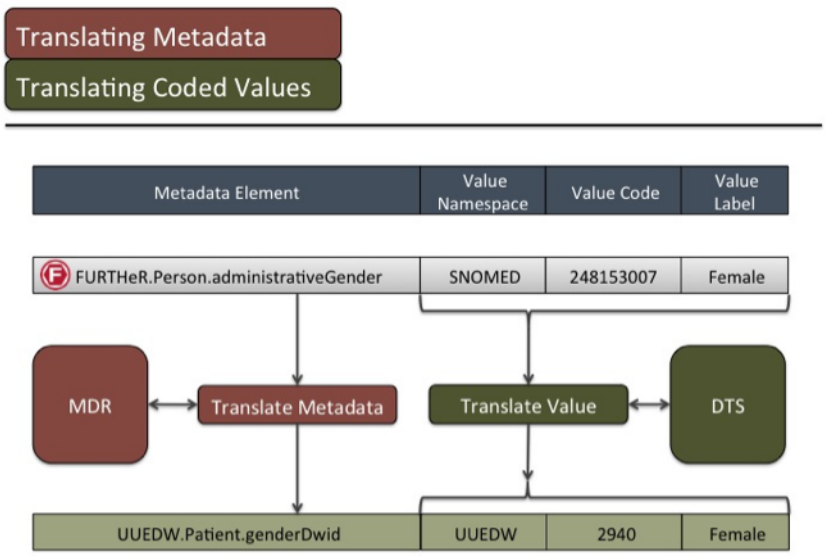

Figure 5: Translating Metadata

5.1.1 Query Translation

The objective of a query translation is to convert the OpenFurther Query Language (FQL) query (OpenFurther's classes, attributes, and attribute values) into the physical data source's data classes, attributes, and attribute values while maintaining the integrity of the query logic.

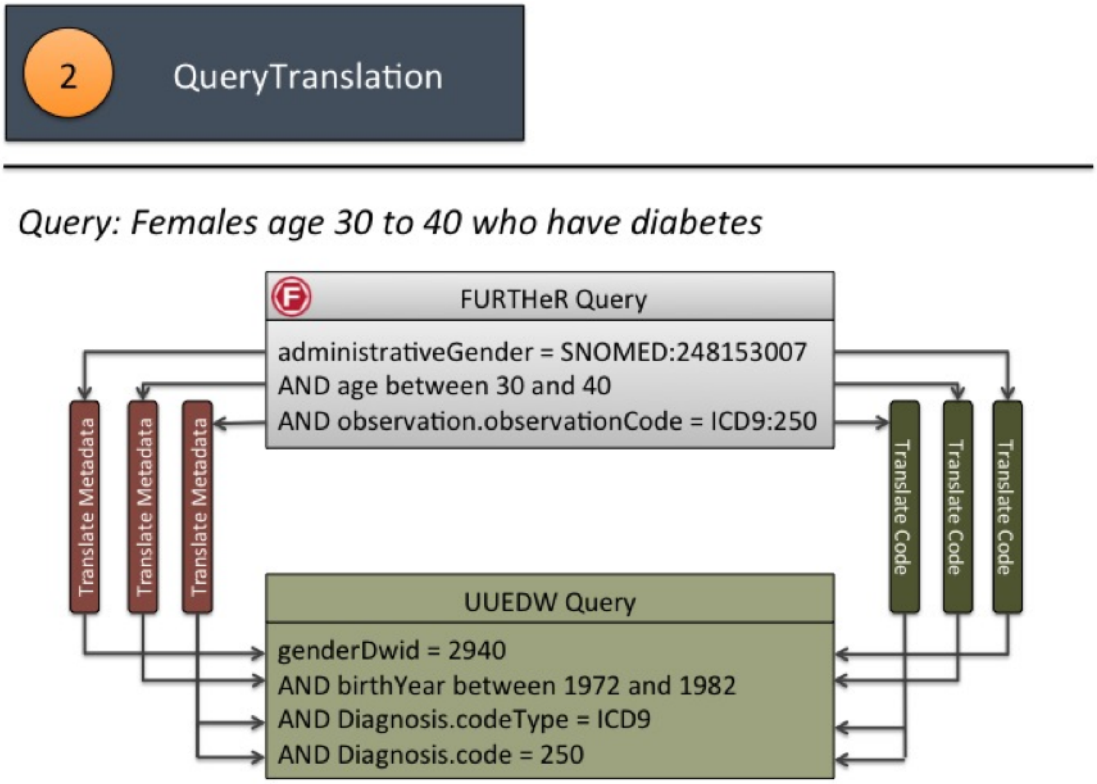

Figure 6: Query Translation

The user interface, currently i2b2, is responsible for building a query. When a query is submitted to the FQE, the FQE converts i2b2's query into the FQL, an XML representation of the query (see the FQL XML Schema) that consists of logical expressions using OpenFurther's data model classes and attributes. Class and class attribute names used in FQL are based on OpenFurther classes and attributes and can be found in the OpenFurther's Java code located here: https://github.com/openfurther/further-open-core/tree/master/ds/ds-further/src/main/java/edu/utah/further/ds/further/model/impl/domain

Coded class attribute value domains within the OpenFurther model are all based on standard terminology where demographics are SNOMED CT codes, diagnosis are ICD-9 codes, and labs are LOINC codes. All attributes that have coded values sets also have an associated attribute that ends with the term *NamespaceId* (namespaces are also called coding systems). This NamespaceId attribute is used to signify what coding system a particular attribute will use. For instance, raceCode=413773004 and raceCodeNamespaceId=30 would signify the SNOMED CT code for the Caucasian race.

By default, Apelon DTS reserves certain identifiers for use with standard terminologys.

Table 1: Apelon DTS Namespace Identifiers

Namespace	Identifier
SNOMED CT	30
ICD-9	10
LOINC	5102
RxNorm	1552

Example input and output

Example query translation input

```
<query xmlns="http://further.utah.edu/core/query"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" rootObject="Person">
        <rootCriterion>
                <searchType>CONJUNCTION</searchType>
                <criteria>
                        <searchType>SIMPLE</searchType>
                        <parameters>
                                <parameter xsi:type="RelationType">EQ</parameter>
                                <parameter xsi:type="xs:string">
                                        raceNamespaceId
                                </parameter>
                                <parameter xsi:type="xs:long">30</parameter>
                        </parameters>
                </criteria>
                <criteria>
                        <searchType>SIMPLE</searchType>
                        <parameters>
                                <parameter xsi:type="RelationType">EQ</parameter>
                                <parameter xsi:type="xs:string">race</parameter>
                                <parameter xsi:type="xs:string">
                                        413773004
                                </parameter>
                        </parameters>
                </criteria>
        </rootCriterion>
        <sortCriteria />
        <aliases />
</query>
```

Given the above input, query translation would generate the following output

Example query translation output

```
<query xmlns="http://further.utah.edu/core/query"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" rootObject="Person">
        <rootCriterion>
                <searchType>CONJUNCTION</searchType>
                <criteria>
                        <searchType>SIMPLE</searchType>
                        <parameters>
                                <parameter xsi:type="RelationType">EQ</parameter>
                                <parameter xsi:type="xs:string">
                                        raceConceptId
                                </parameter>
                                <parameter xsi:type="xs:decimal">
                                        4185154
                                </parameter>
                        </parameters>
                </criteria>
        </rootCriterion>
        <sortCriteria />
        <aliases />
</query>
```

5.1.2 Result Translation

Each data source queried by OpenFurther will respond with a result set in the platform/database specific format and need to be converted into OpenFurther's data model for final analysis and reconciliation of the returned data from each data source, ie. all the pears, oranges, and pineapples need to be converted the same kind of apples. This is the job of the query result set translations, to translate all the query results back to a common/canonical/platform-independent model, or the OpenFurther model in this case. OpenFurther uses XQuery code to translate platform-specific result sets to the OpenFurther model implying all data is/must be converted to XML. Converting to XML is not an extra cost since OpenFurther is a web service-centric infrastructure where messages between services are communicated via XML. Query results are no exception. Data within the MDR drives the XQuery code to translate the data source specific data model and values to the OpenFurther data model and values based on standard terminology. After the XML has been translated the data are unmarshaled back to Java objects, the OpenFurther model Java objects, where/when they are persisted to the query results database (typically the in-memory database) using Hibernate.

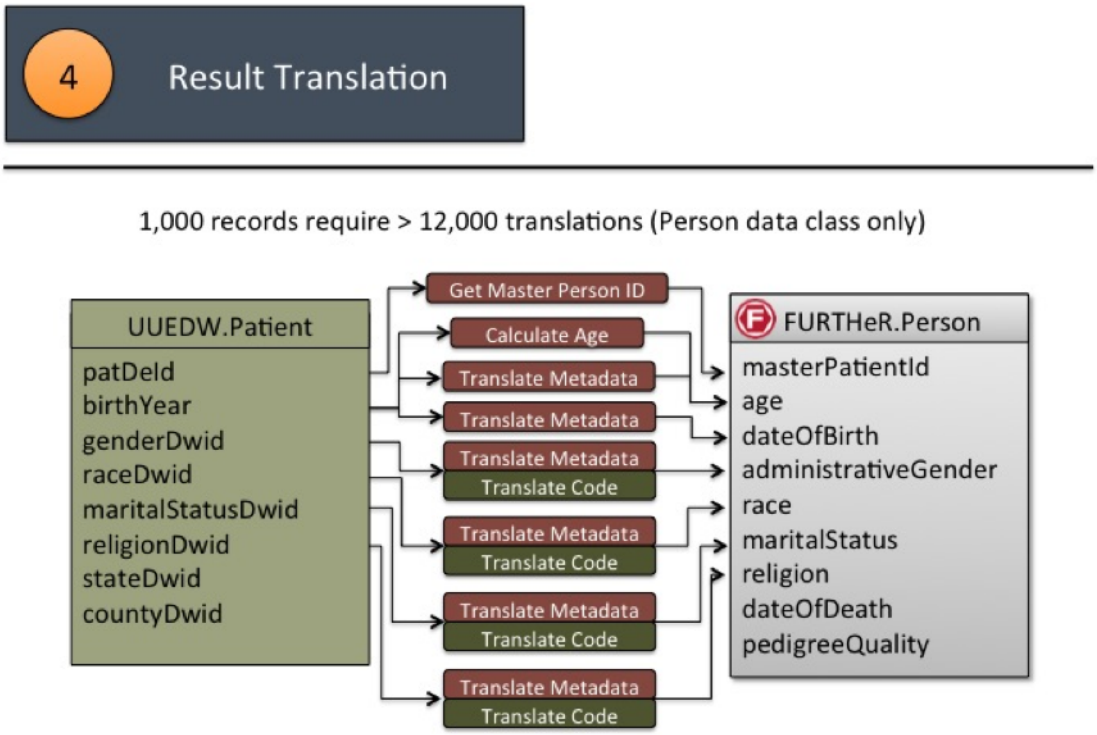

Figure 7: Result Translation

Creating metadata in the MDR

Translations depend on the MDR for attribute-to-attribute translations. The MDR is supported by an abstract data model where metadata "things" are Assets (see the FMDR.ASSET table), including data classes and class attributes. There are other Asset supporting tables ASSET_VERSION and ASSET_RESOURCE that you can ignore for now, for this, as they are not currently used for this purpose. There are, however, two other tables that are critical, ASSET_ASSOC (association) and ASSET_ASSOC_PROP (association properties). ASSET, ASSET_ASSOC, and ASSET_ASSOC_PROP work together to describe attribute-to-attribute translation mappings.

ATTRIBUTE-ATTRIBUTE MAPPING INSTRUCTIONS

1. Create a namespace - a namespace is itself an Asset of Asset type namespace (see other namespace Assets for examples. For demonstration purposes we will call it MyNamespace

2. Create a class in MyNamespace - this is done by creating another Asset that is of type Physical Class.
3. Create the class attributes in MyNamespace - this is done creating Assets that are of type Class Attribute.
4. Associate all of the class attributes with your class by creating an Asset Association (ASSET_ASSOC) to create the facts myPhysicalClass hasAttribute myClassAttribute for each of the attributes previously created.
5. Attribute-to-attribute mappings are mapped together in a similar way, through the Asset Association, by associating attribute Assets with other attribute Assets. In this case we would want to map your newly created attributes with the OpenFurther model attributes, say OpenFurther.Person.dateOfBirth to myPerson.birthDate. And by the way, the direction of this relationship is crucial, LS=left side, RS=right side, so that OpenFurther.Person.dateOfBirth (right side) maps to myPerson.birthDate (left side) using the association "translatesTo". The view ASSET_ASSOC_V illustrates existing mappings.
6. Create attribute-to-attribute translation properties - translation associations (and other associations) can have properties (in ASSET_ASSOC_PROP) that describe the translation mapping requirements. For example, some properties may direct a data type conversion such as int to string, while others may declare a function that needs to be used for a functional conversion, or even an instruction to not change an attribute name. Properties are created via the ASSET_ASSOC_PROP table and are associated to ASSET_ASSOC records.

6 Data Source Adapters

Data source adapters are the pieces of OpenFurther which interact with a data source. Loosely speaking, data source adapters are like plugins. They are simply modules that listen for incoming query requests and act upon them, following a specified protocol. Any programming language that can send and receive messages to a JMS topic, as well as process XML, can be used to program a data source adapter. We do, however, recommend using the existing framework.

Data source adapters follow a standard protocol:

- initialization
- query translation
- execution
- result translation

At the end of each step, status messages are sent to a JMS topic. Statuses include the current state of the query and how many results have been processed at that time.

Likewise, every query can be in one of the following states:

- QUEUED
- STARTED
- EXECUTING
- STOPPED
- FAILED

6.1 Java Data Source Adapter Framework

OpenFurther provides several data source adapters, supported by the community, that run against well known data models. These adapters can be used by downloading the existing adapter, customizing the configuration, compiling them for execution, and installing them into the system.

Additionally, OpenFurther is flexible and also provides the ability to implement your own custom adapter. Reasons for doing this include but are not limited to:

- A custom data model
- A custom interface for accessing the data, such as a web service.
- Custom processing required beyond the standard processing steps within an adapter.

6.2 Implementing a custom data source adapter

We recommend downloading the source code of an existing data source adapter to use as a reference and starting point for your custom data source adapter. Existing data source adapters can found here https://github.com/openfurther/further-open-datasources

6.2.1 Query Processors

Data source adapters follow a chain-of-responsibility pattern. The query is passed through several processors and each processor is given an opportunity to interact or ignore the data given to it by the processing of previous processors.

There are several default query processors for each step within data source adaption.

Each Query Processor has a Delegate implementation that contains the business logic to implement each processor.

- QueryTranslatorQp
  - Delegate: QueryTranslatorXQueryImpl - implements query translator by utilizing an XQuery program which in turns utilizes metadata within the MDR. Xquery files are stored within the MDR and can be referenced by path. The path to the MDR file is given as part of initialization.
- QueryExecutorQp
  - Delegate: ExecutorQuestImpl – implements query execution based on the data source type specified by DS_TYPE within initialization. Currently, only database data sources are well supported, however, web services data sources can be implemented with additional effort.
- ResultTranslatorQp
  - Delegate: ResultTranslatorXqueryImpl – implements results translation by applying an xquery file to the marshaled XML results. Xquery files are stored within the MDR and can be referenced by path. The path to the MDR file is given as part of initialization.
- FinalizerQp
  - Delegate: FinalizerMock – does nothing but finish the query

7 Federated Query Engine (FQE)

7.1 Federated Query Language (FQL)

All queries sent to OpenFurther are constructed using FQL. FQL is an object oriented query language expressed in XML that is largely based off of the Hibernate Criteria API.

7.1.1 Root Object

FQL queries are constructed against a given data model, for instance, the OpenFurther model. Every query is centered around a given object. This is called the root object. For instance, when querying for persons with a particular diagnosis, the root object would be Person.

You declare the root object as an attribute of the <query> tag

Declaring a root object

```
<query xmlns="http://further.utah.edu/core/query"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        rootObject="Person">
        ....
</query>
```

7.1.2 Query Attributes

FQL query attributes are simply the field names (instance variables) of the root object you're querying against. If you're familiar with SQL, you can think of query attributes like database columns.

For instance, in the following Person class, you could use *compositeId*, *administrativeGenderNamespaceId*, and *administrativeGender* are all query attributes that can be used in FQL.

Java root object fields

```
public class Person implements PersistentEntity<PersonId>
{

        @Column(name = "FPERSON_COMPOSITE_ID")
        private String compositeId;

        @Column(name = "administrative_gender_nmspc_id")
        private Long administrativeGenderNamespaceId;

        @Column(name = "administrative_gender_cid")
        private String administrativeGender;

        ....

}
```

7.1.3 FQL Reference

Simple Expressions

EQ - Equals

```
<criteria>
    <searchType>SIMPLE</searchType>
    <parameters>
        <parameter>EQ</parameter>
        <parameter>DiagnosisGrouping.codeSequenceNumber</parameter>
        <parameter>766.2</parameter>
    </parameters>
</criteria>
```

NE - Not Equals

```
<criteria>
    <searchType>SIMPLE</searchType>
    <parameters>
        <parameter>NE</parameter>
        <parameter>Lab.value</parameter>
        <parameter>1234</parameter>
    </parameters>
</criteria>
```

GT - Greater Than

```
<criteria>
    <searchType>SIMPLE</searchType>
    <parameters>
        <parameter>GT</parameter>
        <parameter>Lab.reading</parameter>
        <parameter>1234</parameter>
    </parameters>
</criteria>
```

LT - Less Than

```
<criteria>
    <searchType>SIMPLE</searchType>
    <parameters>
        <parameter>LT</parameter>
        <parameter>Lab.reading</parameter>
        <parameter>1234</parameter>
    </parameters>
</criteria>
```

LE - Less Than or Equal

```
<criteria>
    <searchType>SIMPLE</searchType>
    <parameters>
        <parameter>LE</parameter>
        <parameter>Lab.reading</parameter>
        <parameter>1234</parameter>
    </parameters>
</criteria>
```

GE - Greater Than or Equal

```
<criteria>
    <searchType>SIMPLE</searchType>
    <parameters>
        <parameter>GE</parameter>
        <parameter>Lab.reading</parameter>
        <parameter>1234</parameter>
    </parameters>
</criteria>
```

Unary Expressions

NOT - Negation

```
<criteria>
    <searchType>NOT</searchType>
    <parameters/>
```

```
    <criteria>
        ...
    </critera>
</criteria>
```

Multinary Expressions

Conjunction - a conjunction between two or more expressions

```
<criteria>
    <searchType>CONJUNCTION</searchType>
    <parameters/>
    <criteria>
        ...
    </criteria>
    <criteria>
        ...
    </criteria>
    <criteria>
        ...
    </criteria>
</criteria>
```

Disjunction - a disjunction between two or more expressions

```
<criteria>
    <searchType>DISJUNCTION</searchType>
    <parameters/>
    <criteria>
        ...
    </criteria>
    <criteria>
        ...
    </criteria>
    <criteria>
        ...
    </criteria>
</criteria>
```

Interval Expressions

Between

```
<criteria>
    <searchType>BETWEEN</searchType>
    <parameters>
        <parameter>Observation.observationValue</parameter>
        <parameter>1</parameter>
        <parameter>2</parameter>
    </parameters>
</criteria>
```

String Expressions

Like - Contains the value

```
<criteria>
    <searchType>LIKE</searchType>
    <parameters>
        <parameter xsi:type="xs:string">Observation.observation</parameter>
        <parameter xsi:type="xs:string">250</parameter>
    </parameters>
    <options>
        <matchType>CONTAINS</matchType>
        <ignoreCase>false</ignoreCase>
    </options>
</criteria>
```

Like - Exact match of the value

```
<criteria>
    <searchType>LIKE</searchType>
    <parameters>
        <parameter xsi:type="xs:string">Observation.observation</parameter>
        <parameter xsi:type="xs:string">250</parameter>
    </parameters>
    <options>
        <matchType>EXACT</matchType>
        <ignoreCase>false</ignoreCase>
    </options>
</criteria>
```

Like - Value starts with

```
<criteria>
    <searchType>LIKE</searchType>
    <parameters>
        <parameter xsi:type="xs:string">Observation.observation</parameter>
        <parameter xsi:type="xs:string">250</parameter>
    </parameters>
    <options>
        <matchType>STARTS_WITH</matchType>
        <ignoreCase>false</ignoreCase>
    </options>
</criteria>
```

Like - Value ends with

```
<criteria>
    <searchType>LIKE</searchType>
    <parameters>
        <parameter xsi:type="xs:string">Observation.observation</parameter>
        <parameter xsi:type="xs:string">250</parameter>
    </parameters>
    <options>
        <matchType>ENDS_WITH</matchType>
        <ignoreCase>false</ignoreCase>
    </options>
</criteria>
```

Collection Expressions

In - Value(s) is within set

```
<criteria>
    <searchType>IN</searchType>
```

```
    <parameters>
        <parameter xsi:type="xs:string">Observation.observation</parameter>
        <parameter xsi:type="xs:string">401.1</parameter>
        <parameter xsi:type="xs:string">401.2</parameter>
        <parameter xsi:type="xs:string">401.3</parameter>
    </parameters>
</criteria>
```

7.1.4 FQL to Hibernate Criteria

Since FQL is largely based on Hibernate Criteria objects, it's possible to convert an FQL query into Hibernate Criteria that will then allow Hibernate to convert that into SQL.

Converting an FQL is very simple.

1. Using JAXB, unmarshal the XML into a SearchQueryTo.
2. Locate the root hibernate entity class (typically Person or Patient) <Root Entity>
3. Call the QueryBuilder class like below

Converting FQL to Hibernate Criteria

```
final GenericCriteria hibernateCriteria =
        QueryBuilderHibernateImpl.convert(CriteriaType.CRITERIA, <Root
                Entity>.class, sessionFactory, searchQuery);
```

7.1.5 FQL Schema

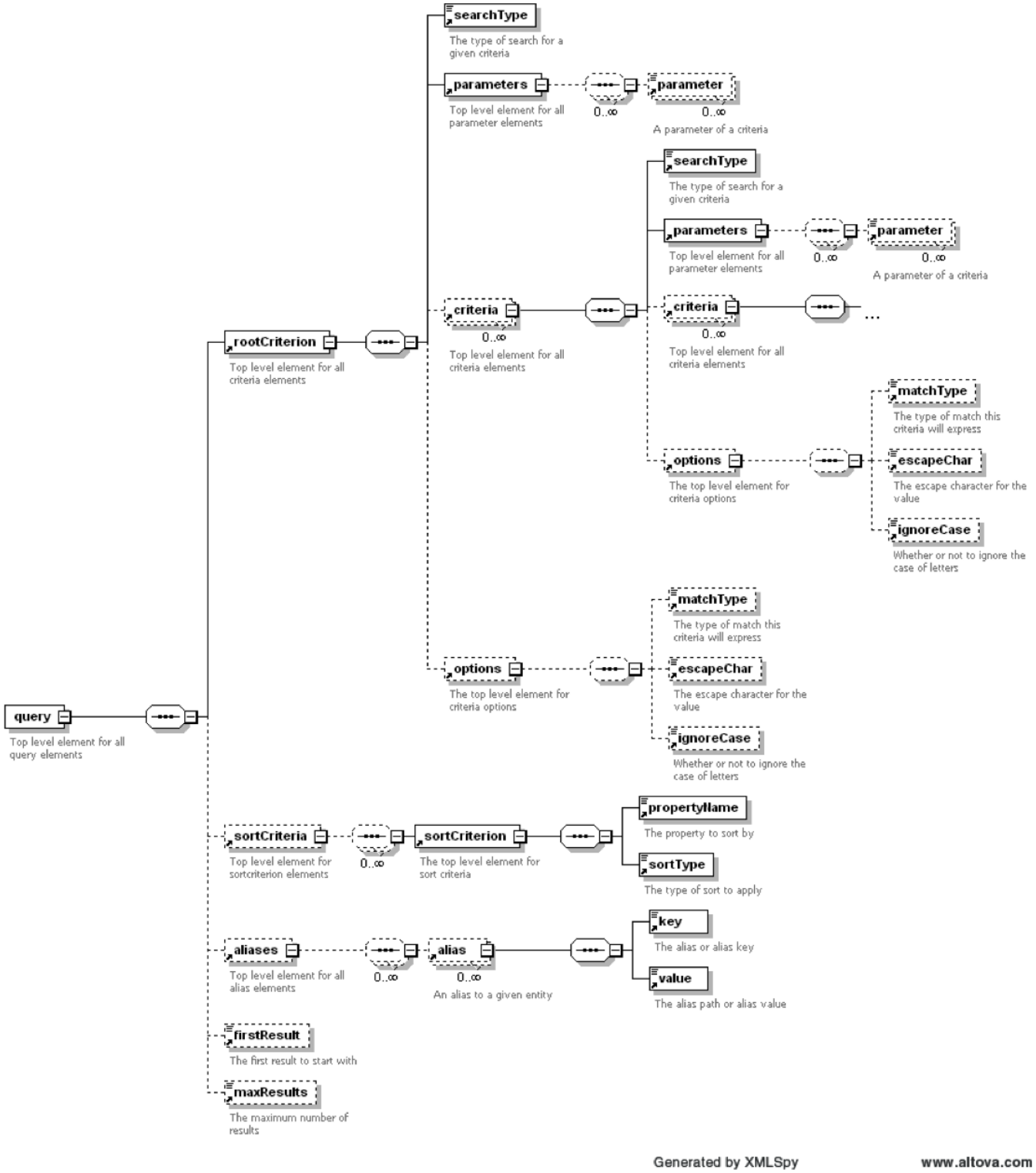

Figure 8: FQL Schema

8 Technologies

OpenFurther is built on a number of Open Source technologies

- Languages
  - Java
  - Groovy
  - Bash
  - Python
- Development Tools
  - Maven 3
  - SonaType Nexus
  - Eclipse
  - Git
  - JIRA
  - Bamboo
- Service Frameworks
  - Spring
  - Apache Commons
  - Apache CXF
  - Apache Camel
- Application Servers
  - Apache ServiceMix
- Testing
  - JUnit
  - Spock

9 Installing

OpenFurther is provided as a VM image for download at this time. The VM can be used as a reference for installation, typically splitting out each Linux user as an individual server.

TODO: Expand this section with detailed instructions for installing on Linux and Windows

10 Demo System Administration

OpenFurther utilizes a number of different servers to run. The following instructions pertain to the demo VM of OpenFurther that is available for download. All scripts used for starting and stopping services are available within the further-open-extras repository on GitHub.

Tip
The demo version contains all of the servers as individual Linux users.

10.1 Apache HTTP Server

The Apache HTTP server runs on port 80 and port 443. As root, run the following

```
service httpd start|stop
```

10.2 In-Memory Database Server

The HSQLDB server runs on port 9001. As root, run the following

```
/etc/init.d/hsqldb start|stop
```

10.3 Core Database Server

Note
While our architecture supports different database, we've currently only tested OpenFurther on Oracle and Oracle XE

```
service oracle-xe start|stop
```

10.4 Terminology Server

The terminology server (Apelon DTS) runs on port 16666 (Requires that the Oracle Database Server has started). As root, run the following

```
su - dtsdemo
dts-auto start|stop
```

10.5 Enterprise Service Bus (ESB)

OpenFurther utilizes an ESB (Apache ServiceMix) to run application code. The ESB requires that the in-memory database, core database, and terminology server are already started. As root, run the following

```
su - esb
start_esb
```

To stop the ESB:

```
su - esb
esbl
further@localhost's password:
further@local> shutdown
Confirm: shutdown instance local (yes/no):
```

10.6 Logging Locations

10.6.1 Apache HTTP Server

The Apache HTTP server logs are located in /var/www/httpd/

10.6.2 In-Memory Database Server

The HSQLDB is currently not configured for logging

10.6.3 Core Database Server

The Oracle XE database server is currently not configured for logging

10.6.4 Terminology Server

The Apelon DTS server logs in /home/demodts/Apelon_DTS/dts/bin/logs

10.6.5 Enterprise Service Bus (ESB)

ServiceMix ESB logs in /home/esb/servicemix/data/log

10.6.6 OpenFurther-i2b2

FURTHeR-i2b2 logs in 2 different locations

- jboss: /home/i2b2/jboss/server/default/logs
- tomcat: /home/i2b2/tomcat/logs