Java VAMDC-TAP Node software

Document Information

Editors:	M. Doronin
Authors:	M. Doronin
Contributors:	
Type of document:	standards documentation
Status:	draft
Distribution:	public
Work package:	WP6
Version:	12.07
Date:	29/08/2012
Document code:	
Document URL:	http://www.vamdc.org/documents/software/JavaNodeSwDoc_12.07.pdf

Abstract: This document is a guide for deployment and use of the Java VAMDC-TAP Node software implementation.

Version History

Version	Date	Modified By	Description of Change
V0.1	31/10/2011	M.Doronin	first draft
V12.07	29/08/2012	M.Doronin	Release documentation for 12.07 Java node software

Acknowledgements

VAMDC is funded under the "Combination of Collaborative Projects and Coordination and Support Actions" Funding Scheme of The Seventh Framework Program. Call topic: INFRA-2008-1.2.2 Scientific Data Infrastructure. Grant Agreement number: 239108.

CONTENTS

1 Introduction **1**
- 1.1 Used software . . . 1
- 1.2 VAMDC common components . . . 1
- 1.3 Node-specific components . . . 2
- 1.4 Comparison with the python/django node software . . . 2
- 1.5 Node implementation . . . 3

2 Database plugin **4**
- 2.1 Request processing . . . 4
- 2.2 DatabasePlugin interface . . . 5
- 2.3 RequestInterface interface . . . 5

3 Database plugin testing **7**
- 3.1 Screenshots . . . 7
- 3.2 Maven Integration . . . 10

4 Database Object Model **12**
- 4.1 Step-by-step guide . . . 12
- 4.2 Notable Cayenne features used . . . 19

5 XSAMS tree building **20**
- 5.1 Example constructor class . . . 20
- 5.2 Attaching objects to XSAMS Document tree . . . 22
- 5.3 Identifiers generation . . . 22
- 5.4 XSAMS JAXB convenience extensions . . . 23
- 5.5 Case-By-Case generic builders . . . 24

6 VSS query recognition and mapping **25**
- 6.1 Query keywords tree . . . 25
- 6.2 Tree objects . . . 25
- 6.3 Query mapping scenarios . . . 27
- 6.4 Query Mapping Library . . . 27

7 Query metrics support **29**
- 7.1 getMetrics(...) method . . . 29
- 7.2 Sample implementation . . . 30

8 VAMDC-TAP node deployment **31**
- 8.1 Install Java application server . . . 31
- 8.2 Deploy node software . . . 31
- 8.3 VAMDC-TAP service configuration file . . . 31
- 8.4 Cayenne configuration using DBCP . . . 32
- 8.5 Database updates and cache . . . 33
- 8.6 Node mirroring . . . 33

Bibliography 35

CHAPTER

ONE

INTRODUCTION

Creation of a separate implementation of the VAMDC-TAP node software in Java pursuied several goals:

- Multiple implementations insure that standards are complete and contain no implementation-specific elements
- Give end-users a choice of the implementation language
- Architectural design of the XML generator based on the schema rather than on a non-hierarchical keywords dictionary (as in Python/Django node software - see below the *Differencies* section)

1.1 Used software

The following open-source libraries were used as Java node software components:

- JAXB RI for XML schema mapping and document output
- Apache Cayenne ORM framework for database access
- MySQL database (any relational database can be used)
- ANTLR generated query parser with slightly modified SQLite syntax
- Oracle Jersey JAX-RS implementation
- Apache Tomcat application server

1.2 VAMDC common components

Following components, developed within the VAMDC project are part of Java VAMDC node software implementation.

- **Dictionaries of standard keywords,** used in a query;
- **Query parsing library,** providing object-oriented view on a query string;
- **Query mapping library,** providing basic support for queries mapping to node database queries.
- **XSAMS helper library,** providing convenience methods for output XML generator implementation;
- Web application, .war archive integrating all libraries
- VAMDC-TAP service validation tool, may be used for node software testing

1.3 Node-specific components

In Java node software each node installation requires creating two code blocks:

1. **Database mapping classes with Apache Cayenne.** This task is well described in Cayenne documentation [CAYDOC] . Fancy graphical tool is provided.
2. **Node plugin,** responsible for query translation into internal queries and building appropriate XSAMS tree from the database access objects, fetched using mapped queries.

This document is dedicated mostly to describe how to implement and deploy a node plugin.

For the rough estimation of the required amount of node-specific code, following numbers should be considered:

- BASECOL node plugin - total of ~1600 lines of code;
- KIDA node plugin - total of ~1500 lines of code;

For all nodes those numbers do not include the autogenerated database model size.

1.4 Comparison with the python/django node software

The paragraph provides a comparison between the Java-Implementation and the Python/Django node software

1.4.1 Common features

- Work as a web application behind a web server
- Use object-relational mapping for database access
- Try to minimize the amount of node-specific code
- Node-specific part works as a plugin

1.4.2 Differencies

- **The main architectural difference** between the the Java implementation and the Python/Django one is the XML generator.

  Java version uses Document Object Model (DOM) mapping of the XML, node plugin needs to build XSAMS blocks as trees of related objects.

  Python version provides a generator with the defined and limited set of loops and anchors("returnables"). Node developer needs to study not only the XSAMS documentation, but also to look through a huge and not well documented list of *returnables* keywords to understand where and how to put his data.

  The use of DOM XML mapping has some advantages and disadvantages:

  – On a good side, it gives much more flexibility in document generation.

  – Additional benefit is that it helps to keep the output document error-free, thanks to compile-time type checks.

  – XML DOM mapping provided is complete: even if node wishes to put the data in a rarely used element of XSAMS, it can without the need to output XML blocks as the plain text.

  – On a bad side, task of building a document tree results in a slightly bigger amount of node-specific code.

  For the task of implementing XSAMS blocks builder, existing builders of KIDA, BASECOL and VALD are useful as good examples.

- **Java implementation doesn't (yet) support document streaming,** it requires to have a whole document tree to be built in memory and then streamed in a response.

  Thus increasing memory footprint, it allows to not strictly follow the document generation order, i.e. export some species and states, then export processes, while exporting some more species and states.

  Also, this way it is easier to verify the document integrity.

- Java implementation doesn't provide an import tool from ASCII files into a relational database

  Well, there was no need for such a tool. You may use Python one.

- Java implementation provides more sophisticated query parsing and mapping support

1.5 Node implementation

So, implementing a node using the Java Node Software would require the following steps:

- Create database model and classes, as described in the *Database Object Model* section.

  After completing this step you will be able to access your database in a convenient way from any Java software you develop. For the details, see the Apache Cayenne documentation. [CAYDOC]

- Set up the project for your plugin, understand the query process and interaction of the node and the plugin. See the *Database plugin* section.

- Create XSAMS tree blocks constructors and builders, as described in the *XSAMS tree building* section

  Here you might need help from the person responsible for database to figure out what XSAMS elements are appropriate for your database content.

  During this step you will be able to test your node plugin: *Database plugin testing*. Try to eliminate any validation errors. The result would be the same for all queries, but it is normal.

- Define the supported restrictables and create mapping classes as described in the *VSS query recognition and mapping* section.

  When this step will be accomplished, you are more than half way through the implementation process. You can test different queries and check if you are getting relevant XSAMS documents as the result.

- The last development step would be to implement the query metrics to be fully compliant with the VAMDC-TAP standard. See the *Query metrics support* section for the implementation details.

- After the node plugin is working, ask your servers manager to deploy the Java Node software on the application server, as it is described in the *VAMDC-TAP node deployment* section. Test again using the VAMDC-TAP Validator in the network mode.

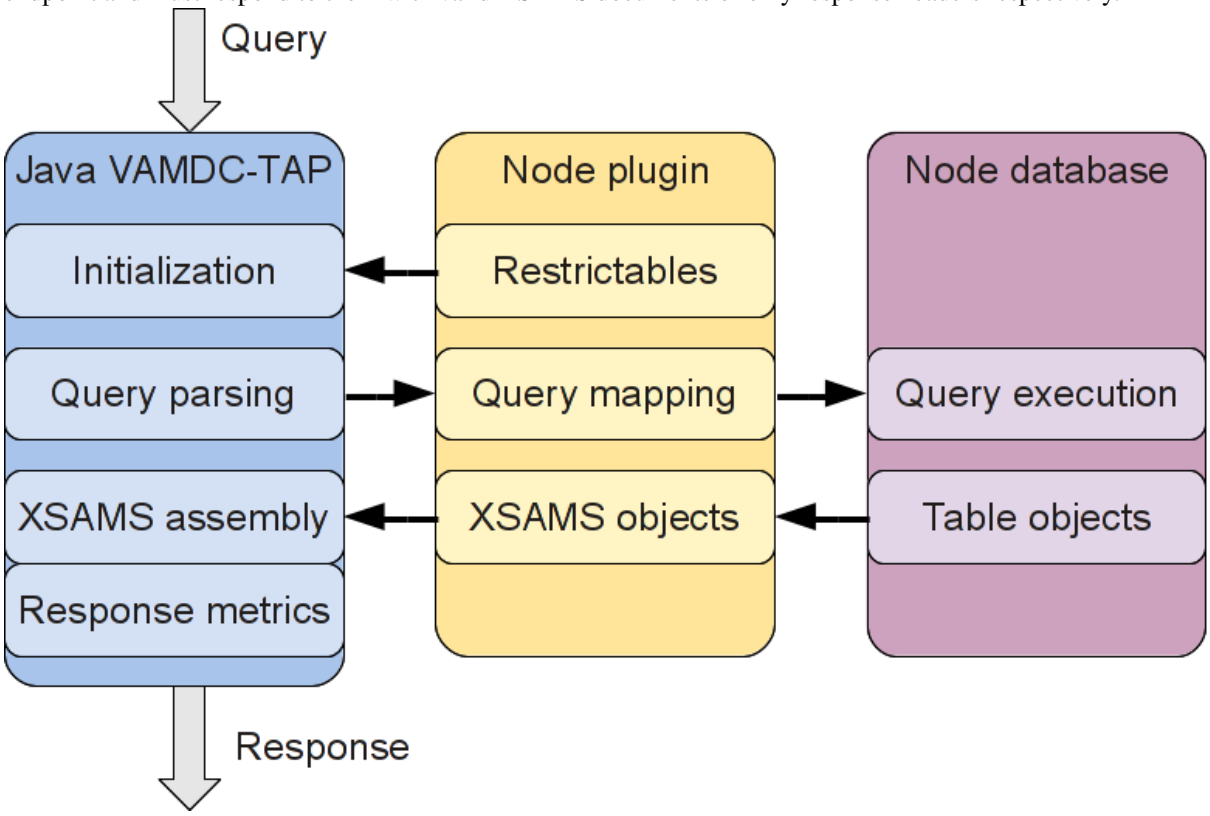
CHAPTER

TWO

DATABASE PLUGIN

2.1 Request processing

In the course of normal operation, node software receives *HTTP GET* and *HTTP HEAD* requests to the */TAP/sync* endpoint and must respond to them with valid XSAMS documents or only response headers respectively.



The following steps are performed to achieve that, Java VAMDC-TAP implementation (**Framework**) is responsible for some of them. Node **Plugin** is responsible for the others. The Apache Cayenne object-relational mapping framework is used to access the **Database**.

- On the query reception, **framework** asks the **plugin** for the list of supported **keywords**.
- **Framework** is parsing the incoming query, checking it's validity and converting it into a group of easily accessible *Query* objects.
- **Framework** asks **Plugin** to construct the document by calling the *DatabasePlugin interface* buildXSAMS() method.
- **Plugin** maps the incoming query to one or more **Database** queries, as described in the *Query mapping scenarios*.

- Before executing or mapping the queries, **Plugin** *should* check if the user actually requested the certain branch of XSAMS document to be built. See the *RequestInterface interface* checkBranch() method for the details.
- **Plugin** receives objects from the **Database**, builds **XSAMS** blocks from them and gives those blocks to the **Framework**. See the *XSAMS tree building* section for the details.
- When the document tree is built, **Plugin** returns the control to the **Framework**.
- **Framework** does the final checks on the document tree, calculates accurate metrics for the document.
- **Framework** converts the document tree into XML stream and sends it to the user.

Interaction between the database plugin and the Java node software is performed through two compact interfaces.

2.2 DatabasePlugin interface

Each and every node plugin must implement the **org.vamdc.tapservice.api.DatabasePlug** interface, defining the following methods:

- **public abstract Collection<Restrictable> getRestrictables();**

  get restrictables supported by this node. Must return a collection of **org.vamdc.dictionary.Restrictable** dictionary elements. This method is called once per each request to the */TAP/sync* and */VOSI/capabilities* endpoints.

- **public abstract void buildXSAMS (RequestInterface userRequest);**

  Build XSAMS document tree from the user request. Object implementing *RequestInterface interface* is passed as a parameter. No return is expected. This method is called every time the node software is receiving an *HTTP GET* request to the */TAP/sync?* endpoint.

  WARNING! Node plugin object is instantiated only once when the node is started, all calls to buildXSAMS should be thread-safe to handle concurrent requests correctly.

  Implementation details are covered in the *XSAMS tree building* section.

- **public abstract Map<Dictionary.HeaderMetrics,Integer> getMetrics(RequestInterface userRequest);**

  Get query metrics. This method is called every time the node receives the HEAD request to the */TAP/sync?* endpoint. *RequestInterface userRequest* parameter is identical to the one passed to buildXSAMS method. This method should return a map of VAMDC-COUNT-* HTTP header names and their estimate values. For the header names and meaning, see [VAMDC-TAP] documentation

- **public abstract boolean isAvailable();**

  Do some really node-specific availability checks. This method is called periodically from the availability monitor. First call is initiated after the first request to the */VOSI/availability* service endpoint. Method may be used to temporary shutdown the node during the database maintenance, or to do some integrity checks on the database. Availability check interval may be set in the *VAMDC-TAP service configuration file* option **selfcheck_interval**.

  WARNING! this method should not be used for doing periodic maintenance since it is never called before the first request to the */VOSI/availability* service endpoint.

2.3 RequestInterface interface

Calls to the node database plugin through *DatabasePlugin interface* get as a parameter an object implementing the **org.vamdc.tapservice.api.RequestInterface**, providing access to the request information and node software facilities.

Following methods are part of that interface:

- **public abstract boolean isValid();** this method returns **true** if the incoming request is valid and should be processed.

  In case of the **false** return, node plugin should not do any processing. Query string may be saved for logging purposes.

- **public abstract Query getQuery();** This method returns the base object of the QueryParser library. Query interface is described in the *Query* section of this document. A few shortcut methods are provided.

- **public abstract LogicNode getRestrictsTree();** The shortcut method to get the logic tree of the incoming query.

- **public abstract Collection<RestrictExpression> getRestricts();** The shortcut method to get all the keywords of the query, omitting the keywords relation logic.

  WARNING! This method should not be used as the main source of data for the query mapping since it completely looses the query relation logic. Imagine the query:

  ```
  SELECT * WHERE AtomSymbol='Ca' OR AtomSymbol='Fe'
  ```

  If this method is used for the query mapping, this query would produce the same result as the query:

  ```
  SELECT * WHERE AtomSymbol='Ca' AND AtomSymbol='Fe'
  ```

  which is obviously incorrect.

- **public abstract String getQueryString();** The shortcut method to get the incoming query string.

- **public abstract boolean checkBranch(Requestable branch);** The shortcut method for the Query.checkBranch(), returns true if the result document is requested to contain a certain branch of XSAMS, specified by the **org.vamdc.dictionary.Requestable** name.

  This method should be called in all builders to verify if a certain branch should be built, before even executing or mapping the queries.

  The behaviour of the keywords is described in the VAMDC Dictionary documentation [VAMDCDict], the section **Requestables**

- **public abstract ObjectContext getCayenneContext();** Get Apache Cayenne object context. That is the main endpoint of the Cayenne ORM library. For more information on using the Apache Cayenne look in the sections *Database Object Model* and *Query mapping scenarios*.

- **public abstract XSAMSManager getXsamsManager();** Get XSAMS tree manager, containing several helper methods. All XSAMS branches built by the node plugin should be attached to it.

- **public abstract Logger getLogger(Class<?> classname);**

  Get the **org.slf4j.Logger** object. All messages/errors reporting should be done with it.

- **public abstract void setLastModified(Date date);**

  Set the last-modified header of the response. May be called anywhere during request processing for any number of times. If called more than once, the last modification date is updated only if the subsequent date is newer than communicated before.

CHAPTER

THREE

DATABASE PLUGIN TESTING

VAMDC-TAP Validator software may be used to test the database plugin operation, so there is no need to install the web server and deploy web service framework on the development machine. It comes with all the needed libraries bundled, and from the plugin point of view the operation with the VAMDC-TAP Validator is undistinguishable from the real-world operation.

To use VAMDC-TAP Validator for the plugin development, simply add the most recent VAMDC-TAP Validator jar file to the library path and create the new run configuration, indicating the **org.vamdc.validator.ValidatorMain** as the main class.

During the first run, open the Settings dialog, switch to the plugin mode and configure the Plugin class name to contain the fully-qualified name of your class implementing the *DatabasePlugin interface* interface.

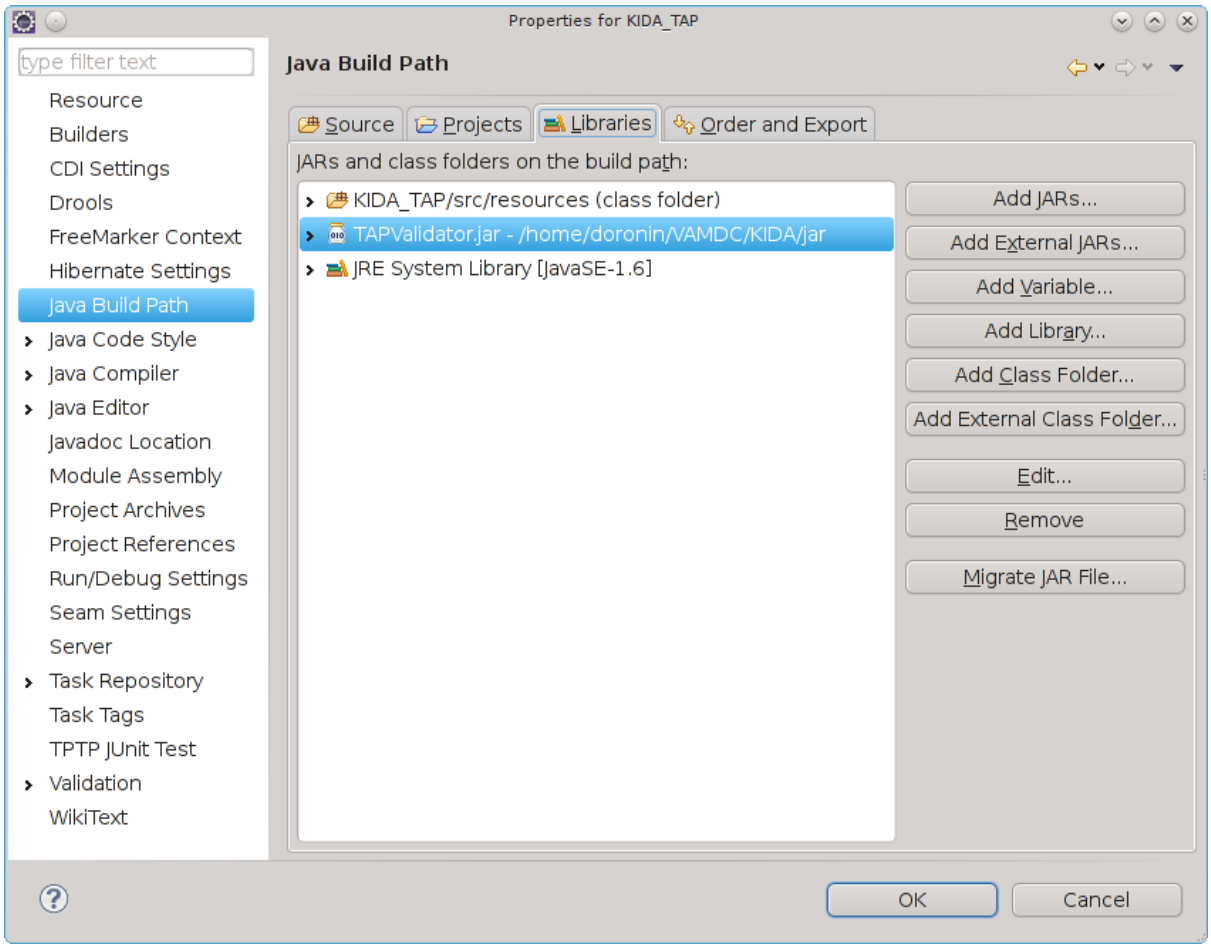
If everything is set up correctly, you should be able to see the list of supported restrictables in the right-top text area and be able to do the queries.

For more information on the VAMDC-TAP Validator user interface and features, consult the [TAPValidator] documentation.

3.1 Screenshots

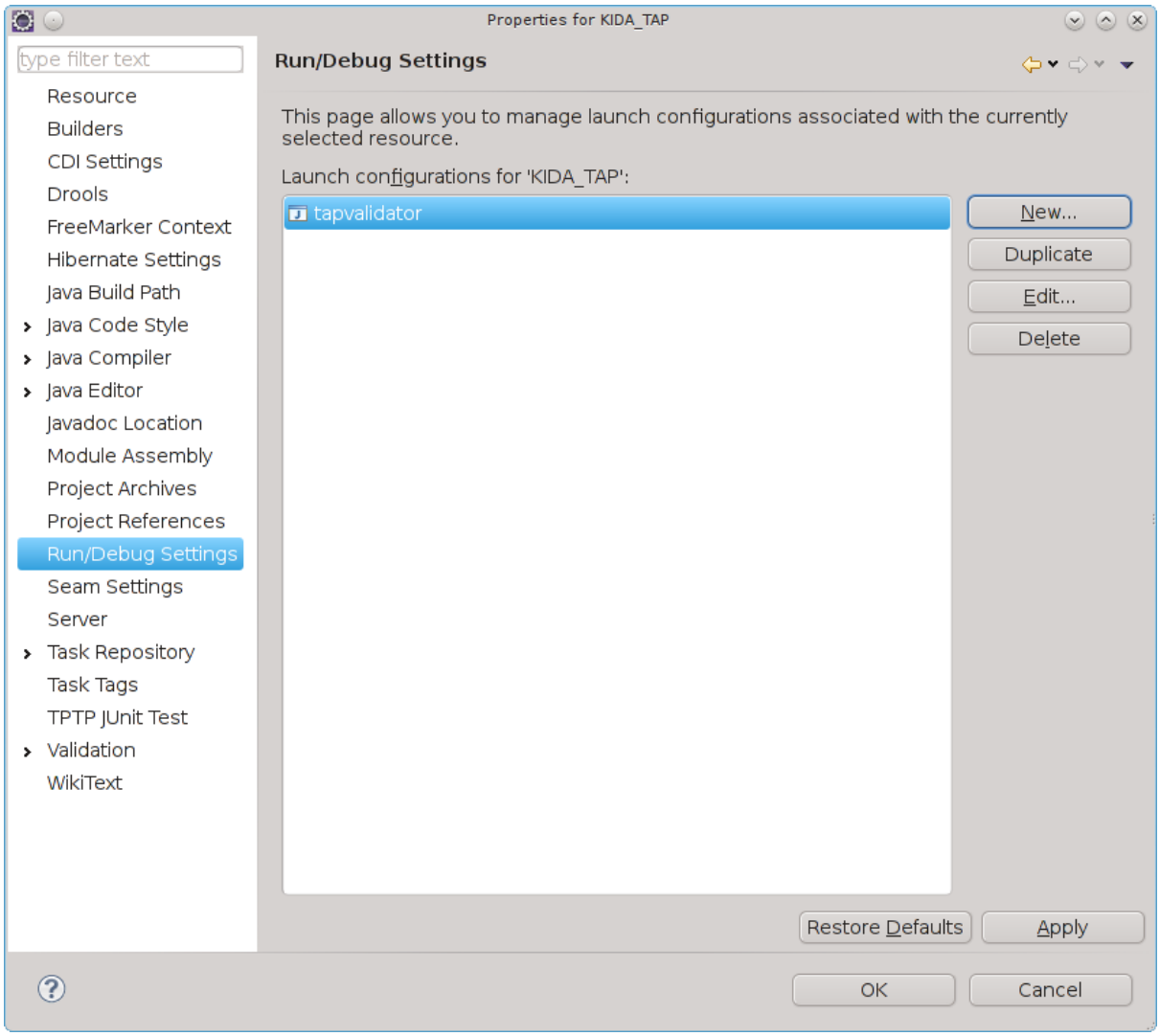
In case you are using the Eclipse for development, the following screenshots might help. Open the project properties of your database plugin.

3.1.1 Adding VAMDC-TAP Validator to the build path



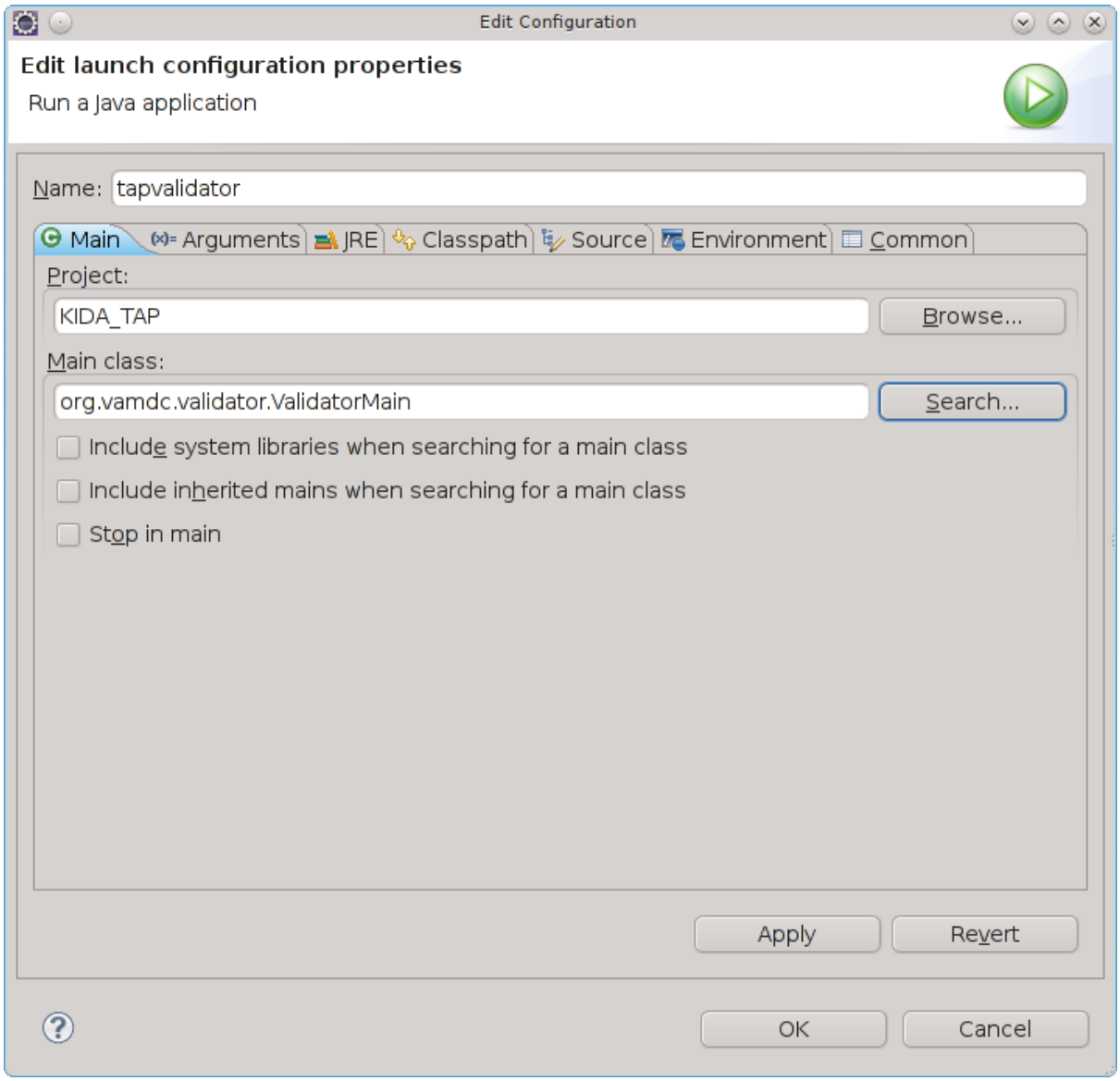
Add the latest VAMDC-TAP Validator JAR to the build path, clicking on the “Add external JARs” button

3.1.2 Managing run configurations



Setup a new run configuration by clicking the “New...” button

3.1.3 Creating run configuration



Create the new run configuration with the following Main class path

3.2 Maven Integration

All Java software developed as a part of VAMDC is available at VAMDC Maven repository

http://dev.vamdc.org/nexus/content/repositories/releases/

To use Maven for dependency management of your plugin, a following sample POM.xml may be used:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>org.vamdc.%databasename%</groupId>
        <artifactId>plugin</artifactId>
        <name>databasename plugin for java node software</name>

        <parent>
```

```
                <groupId>org.vamdc.tap</groupId>
                <artifactId>vamdctap-plugin</artifactId>
                <version>12.07</version>
        </parent>

        <distributionManagement>
                <repository>
                        <id>releases</id>
                        <url>http://dev.vamdc.org/nexus/content/repositories/releases</url>
                </repository>
        </distributionManagement>

        <dependencies>
                <dependency>
                        <groupId>org.vamdc.%databasename%</groupId>
                        <artifactId>database_dao</artifactId>
                        <version>0.0.1-SNAPSHOT</version>
                </dependency>
                <dependency>
                        <groupId>commons-collections</groupId>
                        <artifactId>commons-collections</artifactId>
                        <version>3.2.1</version>
                </dependency>
        </dependencies>
</project>
```

All the required common dependencies are described within *parent* pom, **vamdctap-plugin**

CHAPTER

FOUR

DATABASE OBJECT MODEL

Java VAMDC-TAP node software implementation suggests and supports the use of Apache Cayenne Object-Relational Mapping (ORM) library.

Apache Cayenne supports various database engines, notably, MySQL, PostgreSQL, SQLite, Oracle, DB2, Microsoft SQL Server.

Creation of database objects is made simple thanks to the graphical modeler application, provided as a part of Cayenne.

Process of creating and using a database model is well described in the project official documentation [CAYDOC] and there is no need to repeat it in this document. Reading the Cayenne documentation is a **MUST** for the understanding and creating efficient query mapper routines and high performance database access classes.

4.1 Step-by-step guide

Here is a small illustrated guide on creating database mappings. We will need Cayenne modeler application, it can be downloaded from http://cayenne.apache.org/download.html as a part of binary distribution.

4.1.1 Create maven project

First we need to generate a Maven project:

```
mvn archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DgroupId=org.vamdc.database \
  -DartifactId=daoclasses
```

and create folder src/main/resources where we will put cayenne modeler files.

Maven pom.xml can be replaced with the following contents:

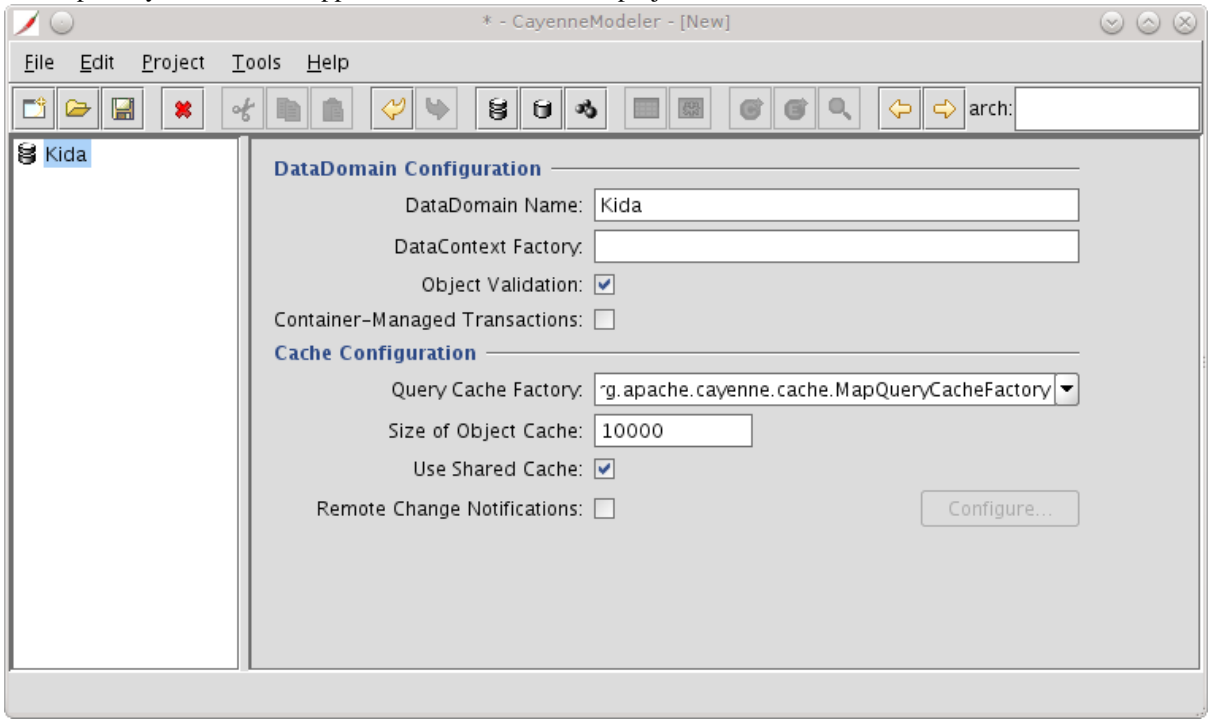
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>org.vamdc.databaseName</groupId>
        <artifactId>node_dao</artifactId>
        <version>12.07</version>
        <name>databaseName database objects</name>

        <parent>
                <groupId>org.vamdc.tap</groupId>
                <artifactId>cayenne_dao</artifactId>
                <version>12.07</version>
        </parent>
```

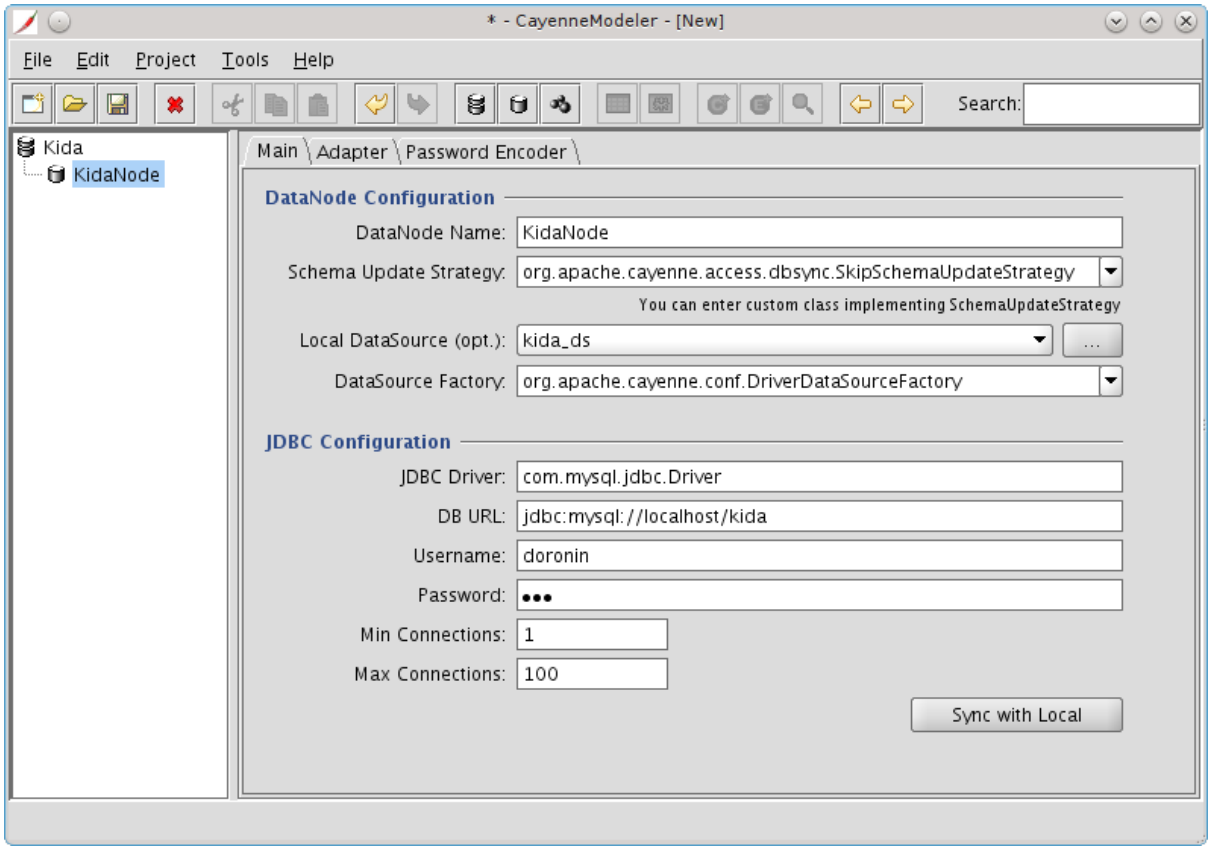
```
        <repositories>
                <repository>
                        <id>vamdc repository</id>
                        <name>VAMDC stuff for Maven</name>
                        <url>http://dev.vamdc.org/nexus/content/repositories/releases</url>
                        <layout>default</layout>
                </repository>
        </repositories>
</project>
```

4.1.2 Create Cayenne classes

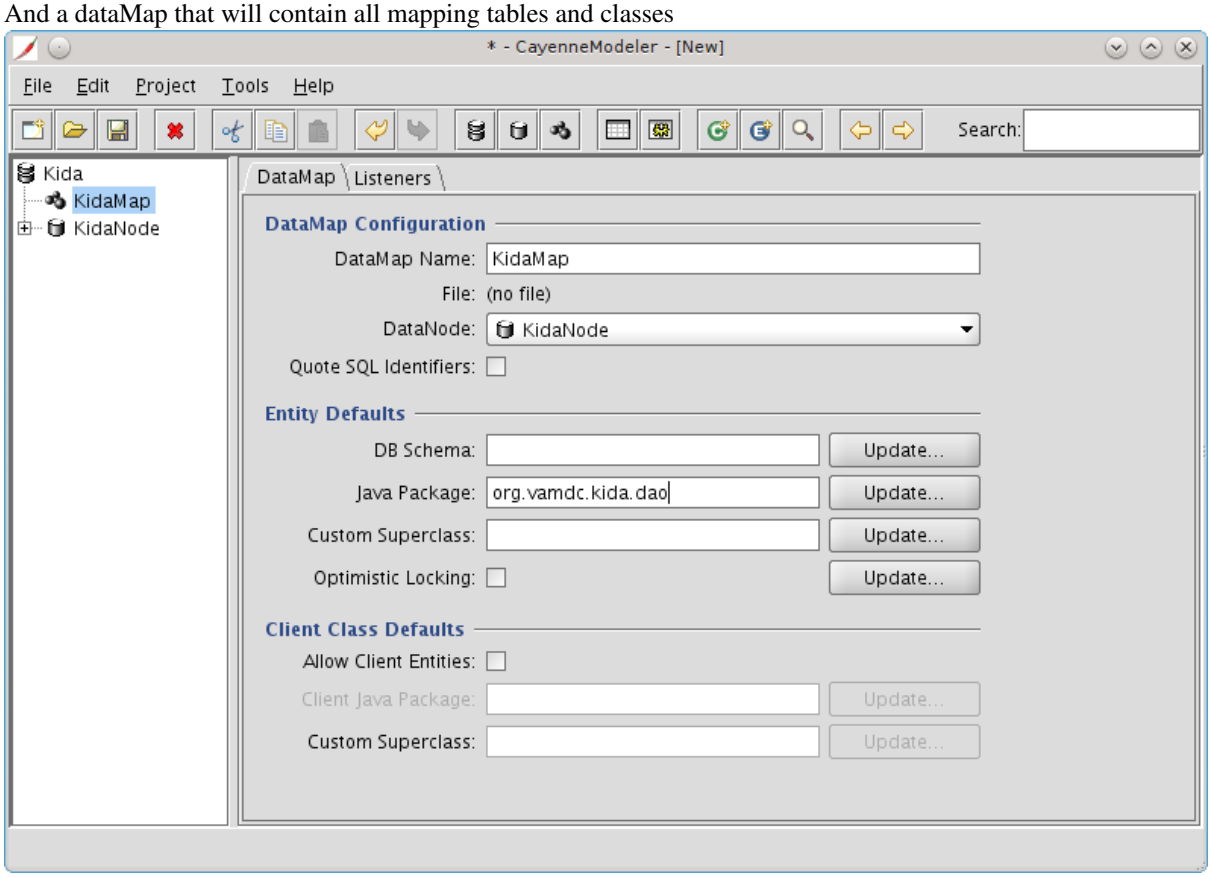
Let's open cayenne modeler application and create a new project:



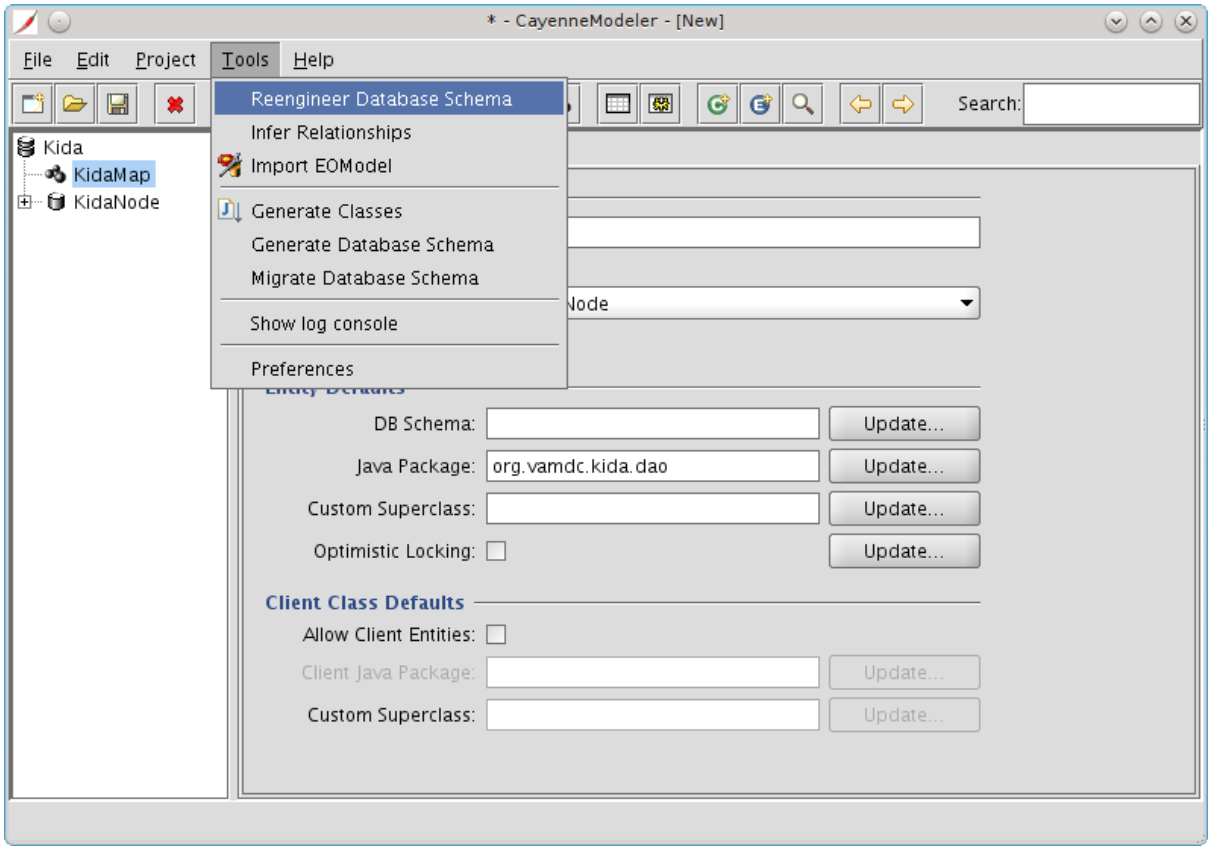
Now we need a dataNode describing database connection

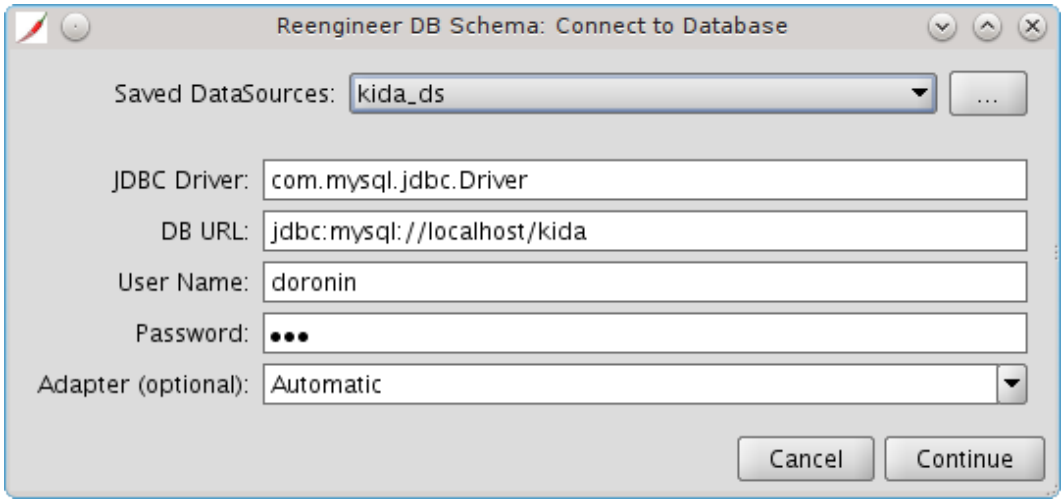


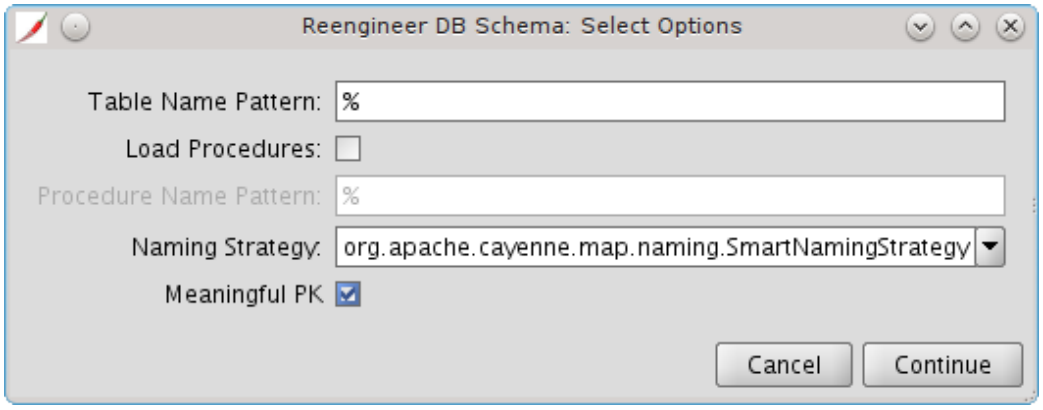
And a dataMap that will contain all mapping tables and classes



After the DataMap is created, we need to import the database schema:

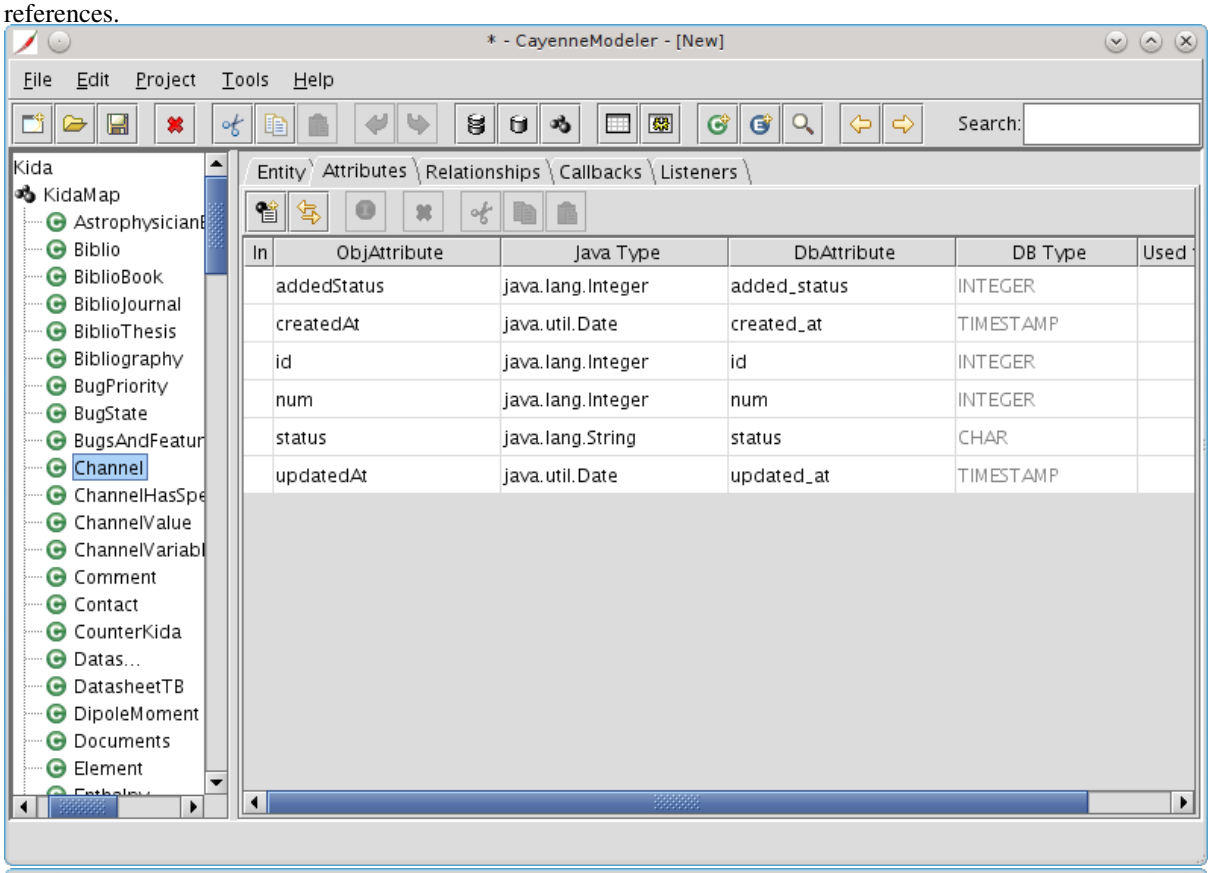


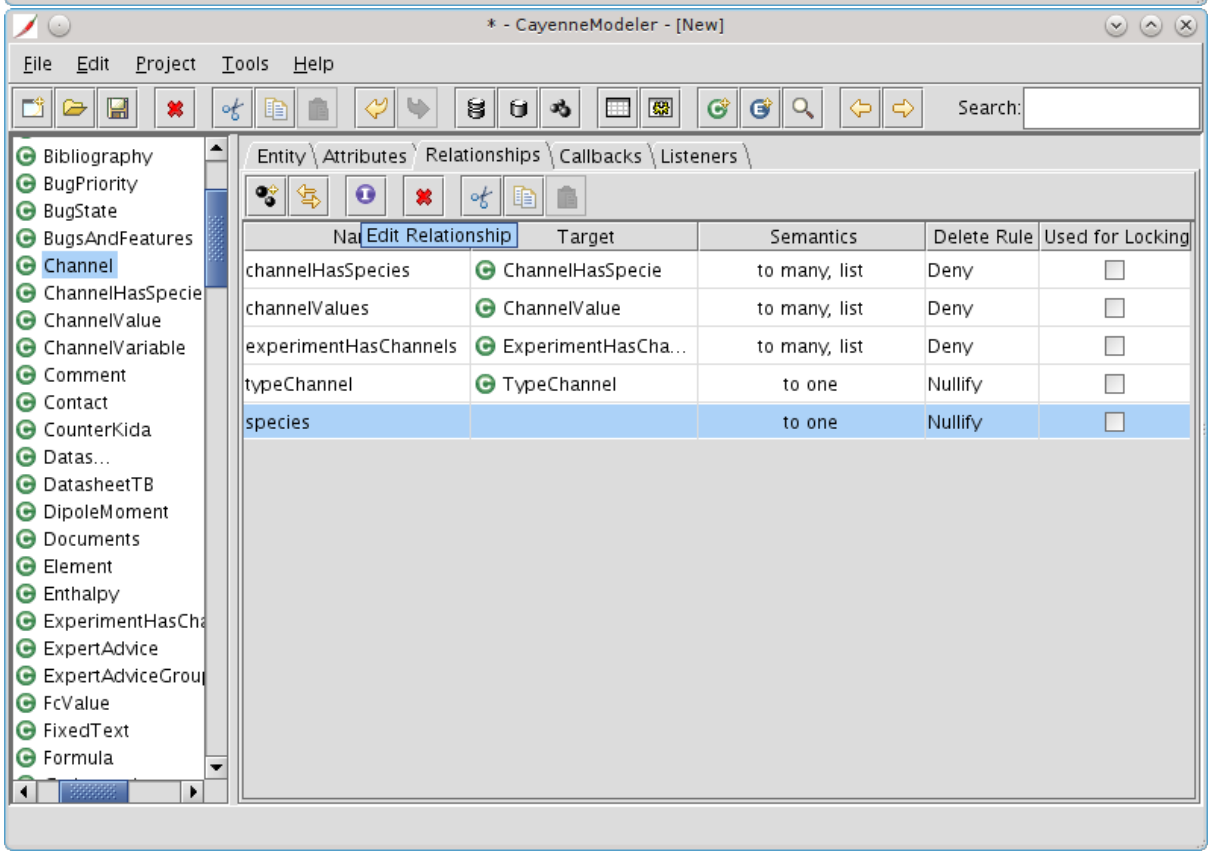




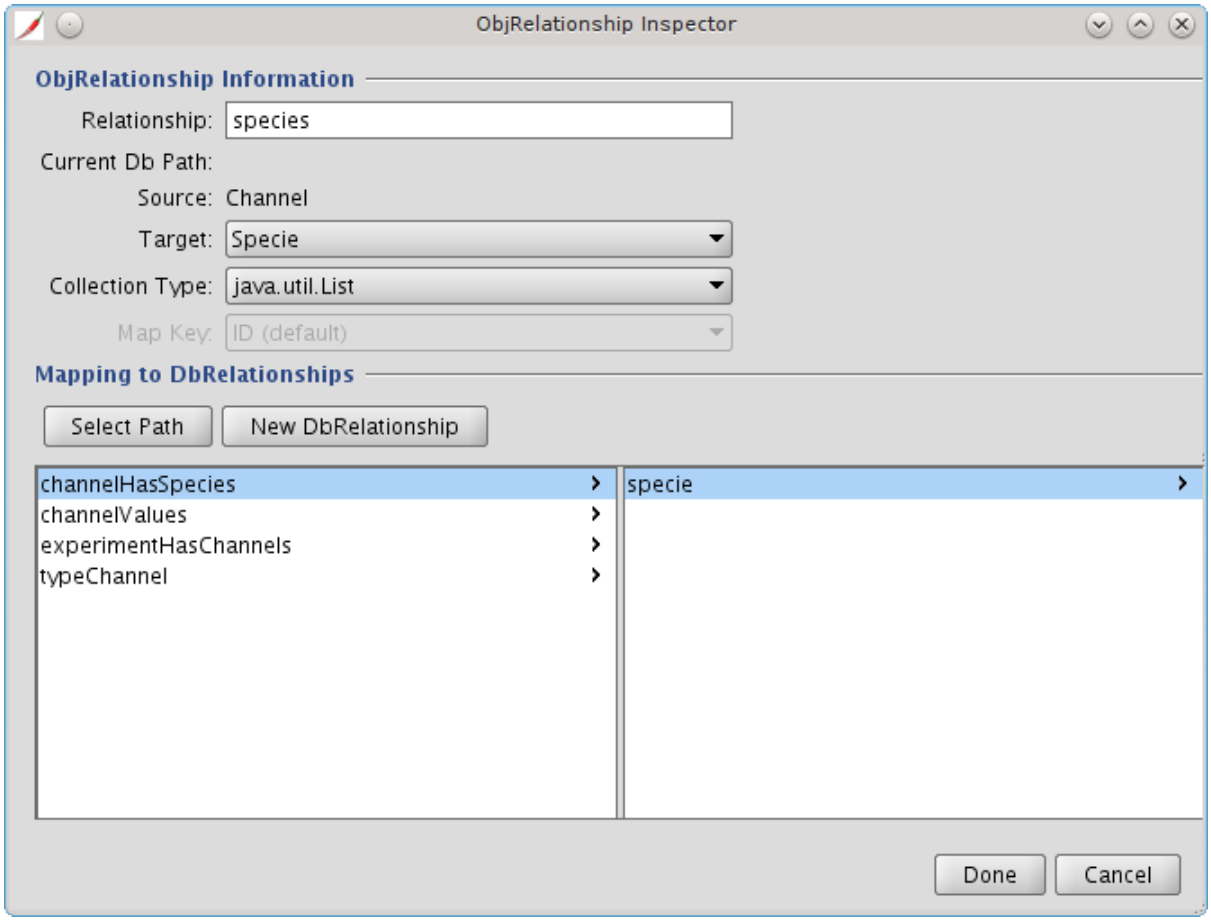
!Note the "Meaningful PK" flag is set for the sake of the generated classes to have getters for the table primary key field.

As a result for each database table a separate Java class is generated, containing attribute fields and relationship references.

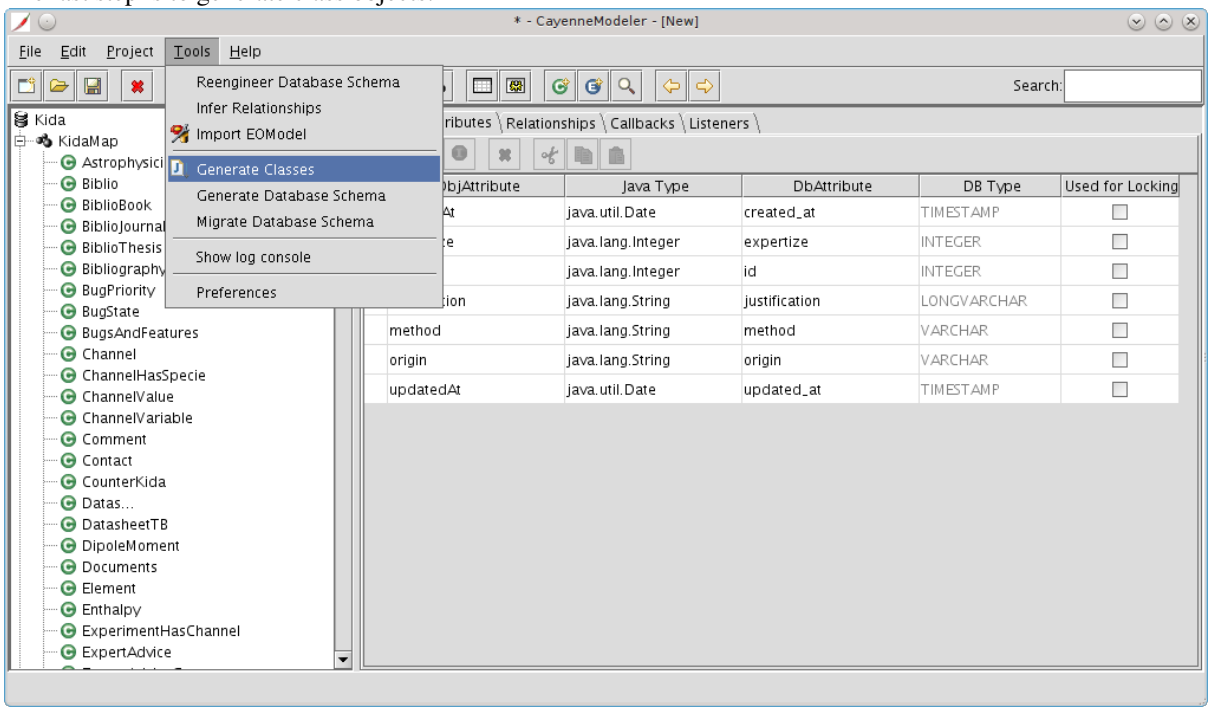


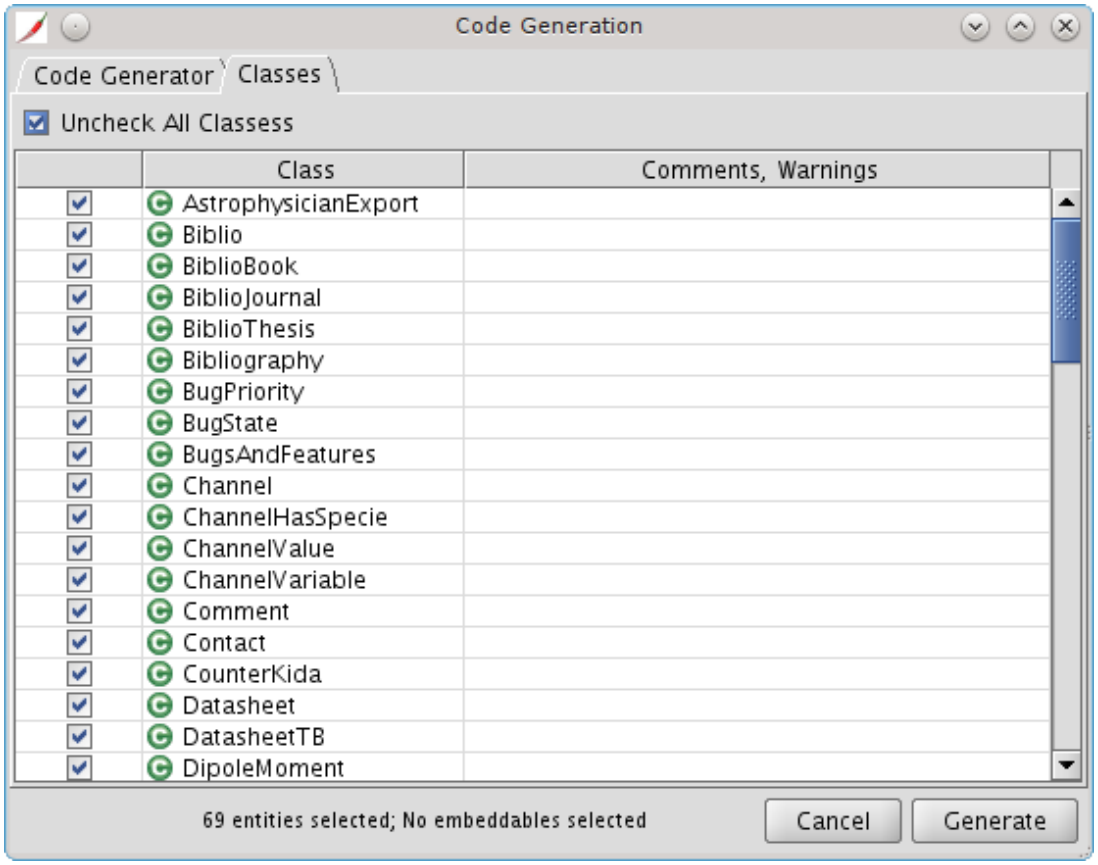


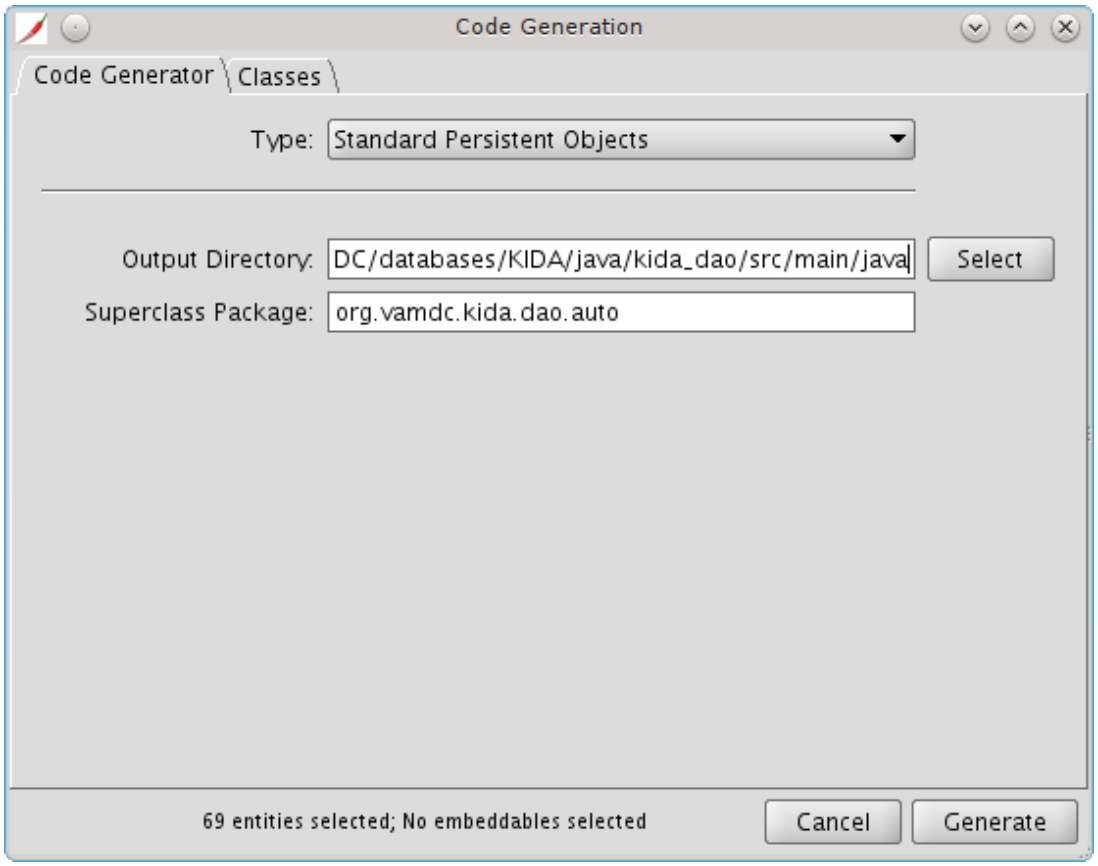
For many-to-many relations relationships need to be created manually: create a new relationship, press on the violet I button, indicate the path in the bottom of the window.



The last step is to generate class objects:







As a destination directory, src/main/java of Maven project needs to be specified. Cayenne project itself needs to be saved next to it, in src/main/resources.

4.2 Notable Cayenne features used

The goal of using ORM framework is not only to get object-oriented view of database, but rather to simplify and automate query translation.

4.2.1 Relations traversing

If properly defined, database model contains information about all tables relations through the foreign keys. While constructing the query, those relations can be automatically traversed to form the correct query with desired selection criterias.

For example, we have a table **'artists'** with fields **'id'** and **'name'** and a table **'albums'** with fields **'id'**, **'artistId'**, **'name'** and **'year'** For the table **'albums'** we have one-to-one relation with the **'artists'** table, called **'albumArtist'** and for artists the reverse one-to-many relationship **'artistAlbums'**

So, if we want to get all artists that released albums in 1980, we would create an **Expression** containing the path from the **'artists'** table to the **'year'** field of **'albums'** table and the expression type **'match'**

```
Expression exp = ExpressionFactory.matchExp("artistAlbums.year", 1980);
SelectQuery query = new SelectQuery(Artists.class,exp);
List<Artists> artists = context.performQuery(query);
```

To add another constraint on a query, we may redefine the Expression:

```
exp = exp.andExp(ExpressionFactory.likeExp("name", "Thomas%"));
```

Here we are not traversing the relationship, but using the table field as a constraint directly.

Normally, none of the expressions would require 'manual' construction, they will be translated from the incoming queries. Query translation is described in a separate chapter *Query mapping scenarios*

4.2.2 Path aliases

Imagine that we have the scenario of many-to-many relation through a separate table. For the previous example, let's add a table **'artistAlbums'** with three columns, **'id'**, **'artistId'** and **'albumId'** Table **'albums'** doesn't any more contain the 'artistID' column, but both forward and reverse relations are still called **'albumArtists'** and **'artistAlbums'**

If we need to select artists that released albums both in 1980 and 1990, joining expressions neither with exp.andExp nor exp.orExp would give us appropriate queries.

exp.andExp() would return no results,

exp.orExp() would return all artists that released albums either in 1980 or 1990.

For such a case, Cayenne provides aliases mechanism:

```
Expression e1 = ExpressionFactory.match("artistAlbumsAlias1.year", 1980);
Expression e2 = ExpressionFactory.match("artistAlbumsAlias2.year", 1990);
Expression e = e1.andExp(e2);
q = new SelectQuery(Artists.class, e);
q.aliasPathSplits("artistAlbums", "artistAlbumsAlias1", "artistAlbumsAlias2");
```

That last command tells the select query how to interpret the alias. Because the aliases are different, the SQL generated will have two completely separate set of joins. This is called a "split path".

CHAPTER

FIVE

XSAMS TREE BUILDING

XSAMS is an XML schema, adopted within VAMDC for data exchange.

Java node software implementation uses JAXB library for mapping between objects and XML elements.

Contrary to the Python/Django node software, Java version doesn't provide limited keyword-based XML generator.

Each node plugin is responsible by itself for building object trees corresponding to the document branches and for attaching them to the main tree, managed by the node software. Node software than outputs the built tree as XML XSAMS document.

XSAMS objects can be constructed by extension of Jaxb mapping classes with convenience methods, provided within **xsams-extra** library. Constructors can receive Cayenne mapping objects as argument and initialize appropriate mapping XML fields. Such an approach allows to instantly apply an arbitrary processing to any field/element value or their combinations.

5.1 Example constructor class

As an example let's look at the BASECOL Source element constructor:

```
package org.vamdc.basecol.xsams;

import java.util.ArrayList;
import java.util.List;

import org.vamdc.basecol.dao.RefsArticles;
import org.vamdc.basecol.dao.RefsGroups;
import org.vamdc.xsams.XSAMSManager;
import org.vamdc.xsams.schema.SourceCategoryType;
import org.vamdc.xsams.schema.SourceType;
import org.vamdc.xsams.util.IDs;

public class Source extends SourceType{

        public Source(RefsArticles article){

                setSourceID(IDs.getSourceID(article.getArticleID().intValue()));
                setCategory(SourceCategoryType.fromValue(article.getJournalRel().getCategory()));

                setSourceName(article.getJournalRel().getSmallName());

                //Year
                setYear(article.getYear().intValue());

                //Authors
                setAuthors(new Authors(article.getFlatAuthorRel()));
                //Title
```

```
                setTitle(article.getTitle());
                //URL
                setUniformResourceIdentifier(article.getUrl());
                //Volume
                setVolume(article.getVolume());
                //Pages
                String pagesbe = article.getPage();
                if (pagesbe!=null){

                        if (pagesbe.contains("-")){
                                fillPages(pagesbe,"-");
                        }else if (pagesbe.contains("\\+")){
                                fillPages(pagesbe,"+");
                        }
                };

        }
        //...
}
```

The full source is available in **org.vamdc.basecol.xsams.Source** class.

Here, RefsArticles is a BASECOL Cayenne mapping object identifying one source record, and SourceType is a root element of XSAMS Sources branch. SourceType is defined by the class **org.vamdc.xsams.schema.SourceType**.

Collection of RefsArticles objects is retrieved automatically through the Cayenne model relation. For each reference element we need to check if it is already attached to the XSAMS Document tree. If not, then the mentioned above builder is called, and finally, after the SourceType object is built, it needs to be attached to the document tree:

```
public static List<SourceType> getSources(
                List<RefsGroups> referenceRel,
                XSAMSManager document,
                boolean filterSource) {

        //Here sources will be added
        ArrayList<SourceType> result = new ArrayList<SourceType>();

        /*always add database self-reference*/
        result.add(document.getSource(IDs.getSourceID(0)));

        if (referenceRel==null)
                return result;

        /*Add all sources that are stated as 'isSource'*/
        for (RefsGroups myref:referenceRel){
                RefsArticles article = myref.getArticleRel();
                if (article!=null && (myref.getIsSource() || !filterSource)){
                        //Check if source with this ID was already referenced:
                        SourceType source = document.getSource(
                                        IDs.getSourceID(article.getArticleID().intValue()));
                        if (source == null){//If not, create and add it:
                                source = new Source(article);
                                document.addSource(source);
                        }
                        //Now, add source record to the list of source references
                        result.add(source);
                }
        }
        return result;
}
```

Later this list should be added to the element requiring source reference, for example, we create a new DataType

value and have references attached to it:

```
DataType quantity = new DataType(table.value, table.units);
quantity.addSources(Source.getSources(table.sourceRelation,request,true));
```

Here, "table" is an object of your database model, providing value and units fields plus the relation to the sources. First, we need to create a quantity of the DataType, then we construct all related source elements, automatically adding them to the XSAMS document tree if necessary, and attach to the quantity element.

5.2 Attaching objects to XSAMS Document tree

RequestInterface provides access to XSAMS Document tree through **XSAMSManager** interface, implementation of which can be obtained by calling **getXsamsManager()** method of the request.

org.vamdc.xsams.XSAMSManager interface provides a handful of methods to add different branches to the XSAMS tree, getting them by known ID or iterating through all of them. For a full list of methods, consult the JavaDoc of the JAXB XSAMS library [XSAMSJavaDoc].

Notable are:

- public String addSource(SourceType source);
- public String addElement(SpeciesInterface species);
- public int addStates(String speciesID,Collection<? extends StateInterface> states);
- public boolean addProcess(Object process);

for adding correspondingly sources, species, states and processes.

5.3 Identifiers generation

Each major block of XSAMS has it's own unique identifier, which is a string starting with a block-specific character.

To assure VAMDC-wide uniquiness of those identifiers, permitting merging of documents, NodeSoftware (both Python and Java implementations) have a mechanism for adding node-specific prefix.

For Java node software it is a special class, **org.vamdc.xsams.IDs**, providing several constants and methods.

- **public static String getID(char prefix, String suffix)** Most generic method, allowing to generate an arbitrary ID. All allowed prefix values are enumerated as *public final static char* constants:
  - IDs.SOURCE
  - IDs.ENVIRONMENT
  - IDs.SPECIE
  - IDs.FUNCTION
  - IDs.METHOD
  - IDs.STATE
  - IDs.MODE
  - IDs.PROCESS
- public static String getSourceID(int idSource)
- public static String getEnvID(int idEnv)
- public static String getFunctionID(int idFunction)
- public static String getMethodID(int idMethod)

- public static String getStateID(int EnergyTable, int Level)
- public static String getModeID(int molecule, int mode)
- public static String getSpecieID(int idSpecies)
- public static String getProcessID(char group, int idProcess)

All those ID generation methods automatically add the configured node-specific ID prefix.

5.4 XSAMS JAXB convenience extensions

For convenience, all XSAMS object classes were extended and grouped into packages by the schema block they are appearing in:

- **org.vamdc.xsams.common** for elements used all around the schema
- **org.vamdc.xsams.environments** for elements from the Environments branch
- **org.vamdc.xsams.functions** for elements from the Functions branch
- **org.vamdc.xsams.methods** for elements from the Methods branch
- **org.vamdc.xsams.process** for elements from the Processes (collisions,transitions) branch
- **org.vamdc.xsams.sources** for elements from the Sources branch
- **org.vamdc.xsams.species** for elements from the Species (atoms, molecules, particles, solids) branch

Few value constructors were added:

- class **org.vamdc.xsams.species.molecules.ReferencedTextType**:

  ```
  public ReferencedTextType(String value);
  ```

  Creates a ReferencedTextType element with the defined value

- class **org.vamdc.xsams.sources.AuthorsType**:

  ```
  public AuthorsType(Collection<String> authors)
  public AuthorsType(String concatAuthors, String separator)
  ```

  First constructor creates Authors elemen with all authors from the passed collection, second one splits the first argument using the separator from the second one and puts the resulting strings into distinct Author records.

- class **org.vamdc.xsams.sources.AuthorType**:

  ```
  public AuthorType(String name)
  ```

  Creates a single Author element with the name from the argument.

- class **org.vamdc.xsams.common.TabulatedDataType**:

  ```
  public TabulatedDataType(String... CoordsUnits);
  public TabulatedDataType(Collection<String> columns);
  ```

  Constructors, defining multi-dimensional tables. Parameters passed define the units of axes, the last element of the collection or the last string define the units for Y (values). The *org.vamdc.xsams.common.TabulatedDataType* class contains a full set of methods for the XSAMS tables manipulation, so if you need to use them it is worth reading the XSAMS library JavaDoc [XSAMSJavaDoc]

- class **org.vamdc.xsams.common.DataType**:

  ```
  public DataType(Double value,String units, AccuracyType accuracy, String comments);
  public DataType(Double value,String units);
  ```

You will certainly use DataType objects, since almost any quantity in XSAMS is represented by them. Two constructors are provided, with parameter names speaking for themselves. Source references may be attached to created object later by calling the *addSource()* or *addSources()* methods.

- class **org.vamdc.xsams.common.ValueType**:

  ```
  public ValueType(Double value, String units);
  ```

  ValueType, used as often as the DataType, supports no source reference and is a simple extension of the Double type, providing the *units* attribute. Convenience constructor is also provided for it.

- class **org.vamdc.xsams.common.ChemicalElementType**:

  ```
  public ChemicalElementType(int charge, String symbol);
  ```

  Used in Atoms and Solids branches, ChemicalElementType has a convenience constructor consuming the atom nuclear charge and it's chemical element symbol.

So far, this is the full list of all convenience constructors created for the XSAMS library. If you need more convenience constructors or methods to be added, contact the Java node software authors and those methods would be included in the next software release.

5.5 Case-By-Case generic builders

Molecular state quantum numbers in XSAMS are represented as additional XML sub-schemas, defining an element QNs with ordered child quantum number elements. Each case has it's own separate namespace, that means that Java JAXB mapping of each case would be in a separate package and the user would either require a generic builder using Java Reflection or have a builder for each case.

Since all cases are just combinations of roughly 30 quantum numbers, the decision was taken to create an intermediate structure able to keep all of them plus the case identifier. The class name is **org.vamdc.xsams.util.StateCore**. It is able to contain a collection of quantum numbers and other important state-related information.

Each quantum number is represented by the **org.vamdc.xsams.util.QuantumNumber** object. It contains the value, optional label and mode index plus the mandatory quantum number type, defining the place where in the case-by-case representation the value will go.

Each autogenerated case package is complemented with it's own builder. The general case builder **org.vamdc.xsams.cases.CaseBuilder** accepts **StateCore** as a single parameter and is calling case builders based on the integer case ID, returning the built tree. Case ID is the same as it is defined in the case-by-case documentation. The following code illustrates the use:

```
StateCore statedata = new BasecolStateCore(myetable, level);
MolecularStateType molecularState = new MolecularStateType();
// filling in other MolecularStateType fields is omitted
if (myrequest.checkBranch(Requestable.MoleculeQuantumNumbers))
        molecularState.getCases().add(CaseBuilder.buidCase(statedata));
```

Here, BasecolStateCore is a custom class that extends StateCore to automatically fill in all the fields from the Basecol Cayenne model.

MolecularStateType is the autogenerated XSAMS JAXB mapping class that should be fed direcly to the XSAMS library by calling the:

```
RequestInterface.getXsamsroot().addState(speciesID, molecularState);
```

Obviously, the element corresponding to the speciesID should already be there.

CHAPTER

SIX

VSS QUERY RECOGNITION AND MAPPING

Obviously, each node has it's own database structure. In order to fetch data requested by the NodeSoftware client, incoming query needs to be mapped into one or multiple queries to the node database. Java Node software tries to make the process of query mapping as simple and sophisticated as possible.

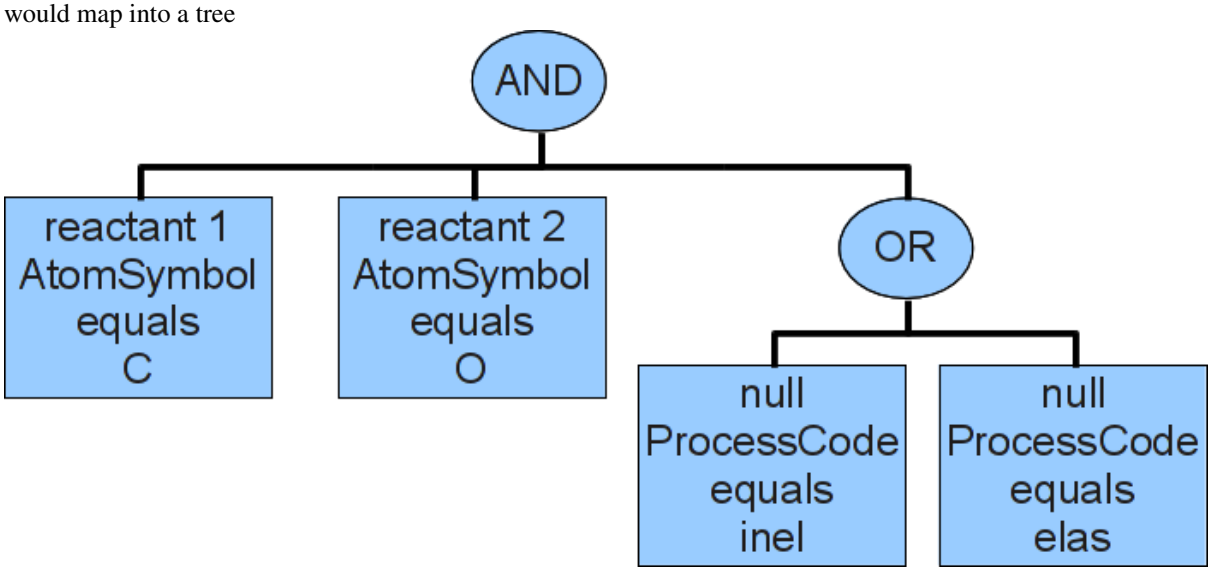
6.1 Query keywords tree

First, as Java node software receives a query, it validates it and parses into a tree of objects. Intermediate nodes of that tree are representing boolean relations and leafs keep the information about the query keywords, comparison operator and values.

For example, query:

```
SELECT * WHERE reactant1.AtomSymbol = 'C' AND reactant2.AtomSymbol = 'O'
AND (CollisionCode='inel' or CollisionCode='elas')
```

would map into a tree



6.2 Tree objects

Following objects are representing the incoming query tree

6.2.1 Query

Main interface of the query parser library, provides access to the query tree and few utility methods.

- **public LogicNode getRestrictsTree()** is the main method, returning the root of the query tree.

  Accompanying are two methods, **getFilteredTree** and **getPrefixedTree**, returning subsets of tree.

- **public LogicNode getFilteredTree(Collection<Restrictable> allowedKeywords)** returns a subtree containing only keywords listed in collection passed as a parameter.
- **public LogicNode getPrefixedTree(VSSPrefix prefix, int index)** returns a subtree containing only keywords having the defined prefix and index. If *null* is passed as a prefix, returned tree would only contain nodes without any prefix.
- **public Collection<Prefix> getPrefixes()** returns a collection of prefixes present in the query
- **public List<RestrictExpression> getRestrictsList()** is the most dummy method, returning a list of all keywords specified in the query. Using that list as a main source for query mapping is discourages since it leads to the loss of logic.

In **getFilteredTree()** and **getPrefixedTree()** the filtering algorithm removes the irrelevant RestrictExpression objects from LogicNodes, then removes logicNodes that has no children.

For example, in case of filtering by the prefix:

- Original query:

```
select ALL where reactant1.AtomSymbol='C' and reactant1.AtomIonCharge=1
and reactant2.AtomSymbol='H' and reactant2.AtomIonCharge=-1 and EnvironmentTemperature > 100
```

- Effective query for getPrefixedTree(VSSPrefix.REACTANT, 1):

```
select ALL where reactant1.AtomSymbol='C' and reactant1.AtomIonCharge=1
```

- Effective query for getPrefixedTree(VSSPrefix.REACTANT, 2):

```
select ALL where reactant2.AtomSymbol='H' and reactant2.AtomIonCharge=-1
```

- Effective query for getPrefixedTree(null, 0):

```
select ALL where EnvironmentTemperature > 100
```

- Effective query for getFilteredTree() with a collection containing only AtomSymbol:

```
select ALL where reactant1.AtomSymbol='C' and reactant2.AtomSymbol='H'
```

6.2.2 LogicNode

LogicNode interface represents a node of the query tree. **getOperator()** method provides access to the node operator, **getValues()** returns a collection of child nodes.

For certain operators like **NOT**, **getValue()** method also makes sense, returning a single child element.

6.2.3 RestrictExpression

RestrictExpression elements are the leafs of the logic tree, representing the actual query restriction keywords.

Same as the LogicNode, RestrictExpression provides **getOperator()**, **getValues()** and **getValue()** methods, plus

- **public Prefix getPrefix()** method, returning this expression prefix
- **public Restrictable getColumn()** method, returning a restriction keyword from the dictionary for this expression

6.2.4 Prefix

Prefix is a simple class, keeping **VSSPrefix** from the dictionary and integer index of the prefix.

- **int getIndex()** method provides access to index, and
- **VSSPrefix getPrefix()** gives access to the prefix name.

6.3 Query mapping scenarios

To obtain an Apache Cayenne Expression object, several mapping scenarios are provided, plus plugin developer is free to implement his own one.

6.3.1 Mapping of logic tree

Mapping of logic tree nodes is always trivial and is one-to-one with Cayenne Expression.andExp(), Expression.orExp(), Expression.notExp(), see the Cayenne Javadoc [CAYJAVADOC]

Usable example of such mapper is provided in *org.vamdc.tapservice.query.QueryMapper* class (vamdctap-querymapper library), that is bundled both with the TAPValidator and the node software.

6.3.2 Mapping of RestrictExpression elements

Mapping of RestrictExpression elements may be a bit more tricky, since they contain lots of information:

- prefix
- prefix index
- VAMDC dictionary keyword
- comparison operator
- value/value set

VAMDC keyword itself may map to one or more database columns, for example, **MoleculeInchiKey** keyword, in case of a database that contains all species within one table, says that the field is **InchiKey** and that we must verify that species we are looking at are actually molecules. To correctly handle such a keyword we will need to AND two Cayenne Expressions and add them to the mapped tree.

Prefix and prefix index may also require a check for a certain field, like if element is a reactant or product in chemical reaction. In this case it may make sense to loop over all defined prefixes using **Query.getPrefixes()** method, then filter the incoming query tree by the prefix with the **Query.getPrefixedTree(...)**, map it as usual, add the desired logic to the resulting expression and finally AND the mapped filtered subtree to the resulting query.

6.4 Query Mapping Library

As a part of Java node software, a Query Mapper implementation is provided. It is able to map incoming query trees into cayenne Expression objects. Query Mapper implementation is a part of **vamdctap-querymapper** library, represented by two interfaces and two generic implementations within a package *org.vamdc.tapservice.querymapper*

- **KeywordMapper** interface defining an interface of RestrictExpression mapper;
- **KeywordMapperImpl** generic implementation, providing one-to-one mapping of Restrictable keywords to database fields without value transformation. In many cases node plugin may use extensions of this class, implementing value translation, to-many fields mapping or prefix-conditional mapping.
- **QueryMapper** interface defining the library main interface;

- **QueryMapperImpl** generic implementation, keeping references to KeywordMappers and responsible for mapping parsed query trees to Cayenne Expressions. Boolean logic operations between nodes are translated one-to-one with Cayenne andExp, orExp and notExp, KeywordMappers are called for each RestrictExpression encountered.

6.4.1 Using QueryMapper library

From the plugin side work with the mapper library is performed the following way:

- In some class we initialize a static variable QueryMapper, in constructor adding keyword mappers for each keyword supported by the node:

```
public final static QueryMapper queryMapper= new QueryMapperImpl(){{
        this.addMapper(
                        new KeywordMapperImpl(Restrictable.IonCharge)
                        .addNewPath("symelementRel.elementRel.charge")
                        .addNewPath("partyRel.elementRel.charge")
                        );
}};
```

  Here subsequent calls to **addNewPath** method define cayenne relations path originating from different primary tables, both used for mapping. The first call is for species query, the second for processes.

- Own extensions of KeywordMapperImpl may be implemented to add the possibility to map keywords to multiple fields, translate values from query units to database units, or add any other specific handling.
- QueryMapper automatically keeps a list of Restrictable keywords supported by node, it can be fetched using **public Collection<Restrictable> getRestrictables();** method.
- From XSAMS builder methods **mapAliasedTree(...)** or **mapTree(...)** methods are called to construct Cayenne Expressions from incoming query trees or filtered subtrees.

CHAPTER

SEVEN

QUERY METRICS SUPPORT

To enable the estimation of the amount of data the node will return for any specific query, node software supports HEAD request queries, containing the same set of parameters as normal GET/POST queries.

As a response node software provides a set of HTTP headers indicating the estimate count of each XSAMS block elements:

- ***VAMDC-COUNT-SPECIES*** Total count of the atomic **Ion** and **Molecule** records with distinct SpecieID attribute.
- ***VAMDC-COUNT-ATOMS*** Count of the atomic **Ion** records with distinct SpecieID attribute.
- ***VAMDC-COUNT-MOLECULES*** Count of the **Molecule** records with distinct SpecieID attribute.
- ***VAMDC-COUNT-SOURCES*** Count of distinct **Source** records
- ***VAMDC-COUNT-STATES*** Count of distinct **State** records, both **AtomicState** and **MolecularState** combined
- ***VAMDC-COUNT-COLLISIONS*** Count of the **CollisionalTransition** elements of the **Processes** branch of XSAMS.
- ***VAMDC-COUNT-RADIATIVE*** Count of the **RadiativeTransition** elements of the **Processes** branch of XSAMS.
- ***VAMDC-COUNT-NONRADIATIVE*** Count of the **NonRadiativeTransition** elements of the **Processes** branch of XSAMS.
- ***Last-Modified*** **HTTP header can also be sent to client to indicate the time when the extracted** data was modified last time.

7.1 getMetrics(...) method

Producing such a response doesn't require to build a document tree, so a dedicated plugin method is called by the node software for each HEAD request.

```
public abstract Map<Dictionary.HeaderMetrics,Object> getMetrics(RequestInterface userRequest);
```

userRequest has the same interface as the parameter of *buildXSAMS* method, but it doesn't expect to have XSAMS tree objects attached, so *XSAMSManager* object should not be accessed.

Typical logic of the method would be like that:

- Translate the query using the same translation strategy as builders use;
- **Convert the query into Count() using** *org.vamdc.tapservice.query.QueryUtil.countQuery(DataContext,SelectQuery)* static method;
- Add the resulting value to the map of headers;

- **If header value is greater than zero, set the value for *Last-Modified* header** using RequestInterface *setLastModified(...)* method and *org.vamdc.tapservice.query.QueryUtil.lastTimestampQuery(...)* translator to obtain the value from a database last-modified date column.
- iterate if more then one builder is normally used

7.2 Sample implementation

Here follows the sample implementation from BASECOL database plugin

```
@Override
public Map<HeaderMetrics, Integer> getMetrics(RequestInterface request) {
        if (request.isValid() && checkRequest(request)){
                return Metrics.estimate(request);
        }
        return null;
}
```

and Metrics.estimate has the following implementation:

```
public static Map<HeaderMetrics, Object> estimate (RequestInterface request){
        Map<HeaderMetrics, Object> estimates = new HashMap<HeaderMetrics, Object>();

        //Estimate collisions
        Expression colExpression = CollisionalTransitionBuilder.getCayenneExpression(request);
        SelectQuery query=new SelectQuery(Collisions.class,colExpression);
        Long collisions = QueryUtil.countQuery((DataContext) request.getCayenneContext(), query);

        if (collisions>0){
                estimates.put(HeaderMetrics.VAMDC_COUNT_COLLISIONS, collisions.intValue());

                request.setLastModified(QueryUtil.lastTimestampQuery(
                                (DataContext) request.getCayenneContext(),
                                query,
                                "modificationDate"));
        }

        //Estimate species
        Expression spExpression = ElementBuilder.getExpression(request);
        SelectQuery spQuery=new SelectQuery(EnergyTables.class,spExpression);
        Long etables = QueryUtil.countQuery((DataContext) request.getCayenneContext(), spQuery);

        if (etables>0){
                estimates.put(HeaderMetrics.VAMDC_COUNT_SPECIES, etables.intValue());

                request.setLastModified(QueryUtil.lastTimestampQuery(
                                (DataContext) request.getCayenneContext(),
                                spQuery,
                                "modificationDate"));
        }

        return estimates;

}
```

Here, getExpression(...) methods are the same translator methods as used in corresponding XSAMS builders.

CHAPTER

EIGHT

VAMDC-TAP NODE DEPLOYMENT

8.1 Install Java application server

Java implementation of VAMDC-TAP node software is intended to be run as a web application within Java application server like Apache Tomcat. For installation instructions refer to the server documentation.

8.2 Deploy node software

Deploying the Java implementation is a simple process that requires not that many steps. If your database plugin is already throughly tested with the VAMDC-TAP Validator, everything should just work.

1. Deploy a recent **vamdctap-webservice** war using your server default method. For Apache Tomcat it would require just to copy .war file into webapps directory with a desired webservice name.
2. It is unlikely that you would need to modify servlet configuration file, but just in case, it's located at: WEB-INF/web.xml
3. Load the address *http://$BASEURL/config* to see the default configuration, adjust it as necessary and put into tapservice.conf in WEB-INF/config. For the full description of all configuration parameters, consult *VAMDC-TAP service configuration file* section of this document;
4. Copy your Apache Cayenne configuration xml to WEB-INF/config/cayenne/ (path can be adjusted in WEB-INF/web.xml) Database credentials can be changed in WEB-INF/config/cayenne/....driver.xml (filename is specific to your database)
5. Copy your DAO jar and plugin jar (or it can be combined in single jar file) to WEB-INF/lib/ directory of your servlet. **!WARNING!** Don't try to copy it to webserver or java system library directory, it won't work, you'll be getting strange ClassCast exceptions on every request.
6. Restart application server or reload application to update the configuration. If you open the application root URL, you will get an index page with all relevant addresses.
7. Check availability and capabilities if everything is fine. Try some test queries with VAMDC-TAP Validator
8. **!WARNING!** Do a backup copy of all configuration files and .jar files that were put or adjusted somewhere in the deployed application folder, notably the node configuration files. All those files will be erased on every **vamdctap-webservice** war update

8.3 VAMDC-TAP service configuration file

Java VAMDC-TAP service is configured through a single file, containing a set of key-value pairs.

Once service is deployed, you may get configuration info through http://host.name:8080/tapservice/config URL, where host.name:8080/tapservice is the url to your web application deployment. If service is not configured, only the default configuration set is presented.

8.3.1 Default configuration parameters

```
force_sources=true
limits_enable=false
limit_states=-1
selfcheck_interval=60
dao_test_class=org.vamdc.database.dao.ClassName
test_queries=select species where atomsymbol like '%';
xsams_id_prefix=DBNAME
baseurl=http://host.name:8080/tapservice
database_plug_class=org.vamdc.database.builders.ClassName
xml_prettyprint=false
limit_processes=-1
```

8.3.2 Detailed parameters description

- **limits_enable** = *(true|false)* enable output document element count limits
- **limit_states** = N - limit maximum output states count in document to N, "-1" disables the limit.
- **limit_processes** = N - limit maximum output processes count in document to N, "-1" disables the limit.
- **selfcheck_interval** = N interval in seconds between service availability self checks. First check is initiated after the first request to /VOSI/availability endpoint. Background checks allow more accurate tracking of "upSince", "backAt" and "downAt" time attributes.
- **database_plug_class** = *org.vamdc.database.builders.ClassName* Class name for builder implementing org.vamdc.tapservice.api.DatabasePlugin interface It is instantiated on tapservice startup and all communication between framework and builders go through it.
- **dao_test_class** = *fully.qualified.apache.cayenne.dao.Object* class name for self availability checking, it must be the table with more than 10 records. May be omitted, but then availability endpoint won't work properly.
- **xsams_id_prefix** = *DBNAME* Prefix for XSAMS library id generator org.vamdc.xsams.util.IDs, used to produce all XML id/idRef references. Each node needs to have it's own prefix to maintain globall uniquiness of identifiers in XSAMS documents.
- **baseurl** = *http://host.name:8080/tapservice* Base url used in VOSI/capabilities output, must contain globally accessible URL pointing to the VAMDC-TAP service web application root. Multiple addresses may be specified to indicate mirror nodes, separated by # symbol.
- **xml_prettyprint** = *(true|false)* Produce pretty-printed XML documents or output everything in a single line of text with no linefeeds. Defaults to false, enabling it increases the size of output XML document by ~20%
- **test_queries** = *select species where atomsymbol like '%';* semicolon-separated list of valid test queries for the node. It must contain only valid queries that demonstrate the full functionality of the node. On the other hand such queries must produce compact documents, since those queries would be used for periodic node testing.

8.4 Cayenne configuration using DBCP

To avoid database connection time-out errors, Apache *commons-dbcp* library should be used in junction with Apache Cayenne. Configuration change for this case is simple and straight-forward. **vamdctap-webservice** application archive already comes with bundled *commons-dbcp* jar.

- **First, in cayenne.xml node element *factory* attribute have to be changed for org.apache.cayenne.conf.DBCPDataSourc** and *datasource* attribute should indicate a file name that will contain key-value pairs of dbcp configuration;
- **Second, dbcp configuration file should be put in *cayenne* configuration directory** with the name same as specified in *datasource* attribute and the following parameters:

- cayenne.dbcp.driverClassName=com.mysql.jdbc.Driver
- cayenne.dbcp.url=jdbc:mysql://hostname:port/databasename
- cayenne.dbcp.username=databaseUserName
- cayenne.dbcp.password=databasePassword

Other DBCP parameters also may be adjusted, see http://commons.apache.org/dbcp/configuration.html for more information.

8.5 Database updates and cache

Java node software maintains it's own database cache. If database fields are updated, this cache needs to be purged. To force node to purge it's caches, request to /clear_cache resource needs to be sent. The full URL will be http://host.name:8080/tapservice/clear_cache. As a result, single text line will be sent indicating the number of records that were contained in the cache. This URL may be accessed either manually, included as a frame in node database administration panel or accessed by a special script that tracks database modifications in some way.

8.6 Node mirroring

The best way to set up node mirrors is to configure database replication on each mirror and deploy multiple instances of node software on mirror servers, each of them using own mysql installation. Deployment procedure of node software on master server and mirrors is the same.

8.6.1 Mysql master server configuration

On master server we need to enable binary logging in my.cnf:

```
[mysqld]:
server-id = 1
log-bin = /var/lib/mysql/mysql-bin
replicate-do-db = databasename
bind-address = 0.0.0.0
```

and add user with replication privileges in mysql console:

```
mysql@master> GRANT replication slave ON "databasename".* TO "replication"@"mirror.ip.or.hostname"
        IDENTIFIED BY "password";
```

mysql service needs to be restarted after that.

After server restart we need to create a database dump:

mysql@master> FLUSH TABLES WITH READ LOCK; mysql@master> SET GLOBAL read_only = ON; mysql@master> SHOW MASTER STATUSG

#mysqldump -u root -p databasename | bzip2 -9 -c - > database_dump.sql.bz2

mysql@master> SET GLOBAL read_only = OFF;

Here we need to note **File** and **Position** values from the mysql command *Show Master Status*.

8.6.2 Mysql slave configuration

On a slave (replica) server we need to set up the following things:

import mysql dump for database:

```
#bzip2 -d -c database_dump.sql.bz2 | mysql -u root -p databasename
```

In my.cnf:

```
[mysqld]:
server-id = 2
relay-log = /var/lib/mysql/mysql-relay-bin
relay-log-index = /var/lib/mysql/mysql-relay-bin.index
replicate-do-db = testdb
```

restart mysql service and start replication:

```
mysql@replica> CHANGE MASTER TO MASTER_HOST = "master.ip.or.host", MASTER_USER = "replication",
MASTER_PASSWORD = "password", MASTER_LOG_FILE = "mysql-bin.000003", MASTER_LOG_POS = 98;
mysql@replica> start slave;
```

where MASTER_LOG_FILE and MASTER_LOG_POS parameters we take from the *SHOW MASTER STATUS G* mysql command on master server.

Then we may see the slave server status by issuing mysql command:

```
mysql@replica> SHOW SLAVE STATUS\G
```

Following status parameters are important and indicated values show that replication is working properly:

- **Slave_IO_State**: Waiting for master to send event
- **Slave_IO_Running**: Yes
- **Slave_SQL_Running**: Yes
- **Seconds_Behind_Master**: 0

Changing **Read_Master_Log_Pos** parameter value may indicate that cache of node software on the mirror needs to be purged. A script can be set up to track this parameter, if no other mean of cache invalidation is used.

In detail mysql replication is described in the official manual: http://dev.mysql.com/doc/refman/5.5/en/replication.html

8.6.3 Node mirror registration

Procedure for registering a new mirror is very simple: In configuration of main node and all mirrors another url needs to be added to the end of **baseurl** parameter, separated with hash (#) symbol.

After reload of master node web application, registry needs to be updated with new Capabilities including new mirror. For registry update, please contact VAMDC registry maintainer via support@vamdc.org.

BIBLIOGRAPHY

[CAYDOC] http://cayenne.apache.org/doc30/modeler-guide.html

[CAYJAVADOC] http://cayenne.apache.org/doc30/api/index.html

[XSAMSJavaDoc] http://dev.vamdc.org/nexus/content/repositories/releases/org/vamdc/xml/

[MavenRepo] http://dev.vamdc.org/nexus/content/repositories/releases/

[JavaNodeSoftware] http://www.vamdc.org/software/

[VAMDC-TAP] http://www.vamdc.org/documents/standards/dataAccessProtocol/vamdctap.html#http-header-information

[VAMDCDict] http://www.vamdc.org/documents/standards/dictionary/

[TAPValidator] http://www.vamdc.org/software/