Gaboto Library - User Manual

Arno Mittelbach

i

Preface

The Gaboto Library - User Manual contains information and tutorials on how to use the library and extend its functionality. Gaboto is using RDF and the named graph extension to RDF as its underlying data model. The two main libraries that implement the RDF standard and that Gaboto is built upon are:

Jena Jena is an open source RDF triple store written entirely in Java. More information on Jena can be found at its website.

NG4J NG4J is an extension for Jena implementing the concept of named graphs. More information on NG4J can be found at its website.

Besides the manual, I've tried to thoroughly document the source code in the JavaDoc format so that it can be used as a reference when working with the library. Looking at Jena's documentation will not be necessary for the first steps, but if you want to seriously use Gaboto you should at least work through the main tutorials.

Have fun with Gaboto,
Arno Mittelbach

Contents

Chapter 1

Introduction to Gaboto

This chapter will give a brief introduction into the Gaboto system. I will explain the basic concepts and will give a short introduction to RDF, since understanding Gaboto's underlying data model is crucial for being able to properly work with it. I will start with describing the project, that led to the creation of Gaboto called **OxPoints** which I will use as a running example throughout this manual. Parts of the OxPoints data will also be shipped with Gaboto as an example project for you to be able to directly start playing with a completely configured system.

1.1 OxPoints - a short History

The old OxPoints system consisted of a large XML file mainly containing information about Colleges, their main buildings and where these are located. Using XSL this information could be transformed into various output formats such as JSON, HTML or KML. For example, the map of all colleges on the university's main website was dynamically created by the OxPoints system.

After more and more data was added and with the plan to add much more, it became clear that a system based on one hierarchical XML file would not scale with the new requirements:

Time based data A time dimension should be introduced to be able to create historical maps.

Rich Relationships It should be able to express many different kind of relationships between entities in the system.

More Entities Besides colleges and buildings many more entities should be added to the system (e.g. drain covers).

Extendable Since nobody could pin down where OxPoints was to be going it should be as easy as possible to extend the system in any way imaginable.

If we only partly wanted to achieve these requirements we had to be very careful during the design of the underlying data model. We conducted research for several months and finally decided to go with an RDF (a standard maintained by the W3C, meant to express relationships between entities) based solution. We hope that chosing this very generic data model will address all the above requirements:

Time based data We have found a way to introduce a time dimension (see *1.4. Further Reading* for a description on the research).

Rich Relationships RDF is meant to express rich relationsships.

More Entities RDF has no restrictions on the elements it talks about. As long as you can make up a URI for the element, you can use RDF to describe it.

Extendable RDF is very generic and meant to be extended.

The platform choice fell on Java and Jena (an RDF triple store written in Java) since this seemed to give us one of the most sophisticated open source triple stores and the flexibility of Java.

1.2 RDF - a very brief introduction

To understand Gaboto you have to understand RDF. In this section I will only very briefly describe RDF and its named graph extension. If any of this does not sound familiar, you should have a look at the more detailed introductions in *1.4. Further Reading.*

RDF describes the idea that any kind of resource can be identified by a URI and relationships between two resources can be described in the form of triples:

```
subject    predicate    object .
```

Subjects and predicates are always a resource, while objects can be either a resource or a primitive literal (e.g. a string). To describe this manual we could, for example, say:

```
example:GabotoManual    dc:title     "Gaboto Library - User Manual" ;
dc:author    "Arno Mittelbach" .
```

A set of these triples that belong together are called **Graph**.

The RDF specification assumes that all RDF triples in a system are stored in one Graph. An extension to RDF called **Named Graphs** allows for the grouping of statements into various RDF Graphs, each graph described by a URI. This makes it possible to describe a graph, like any other resource, with the help of RDF.

We use this idea of describing graphs in RDF to introduce a time dimension to RDF. Each graph in Gaboto contains statements that describe the state of resources during a given time span. So if, for example, a college existed from 1500 to 1678 then we would put a triple describing the existance of the college (we use the rdf:type predicate for this purpose) to the graph that contains all triples valid during that particular time span:

```
<http://graphURIs/1500-1678>
{
oxpdata:someCollege    rdf:type    oxp:College .
}
```

To describe the graphs in the system and define the time span they cover we use a special graph: the **context description graph**. To actually describe the time information we are using the time ontology proposed by the W3C. We use another special graph, the **global knowledge graph** to store information that is not time bound.

1.3 So what exactly is Gaboto Library

Gaboto Library has nothing to do with Oxford University (except that it was developed there), buildings, colleges or geo-data. It is a RDF storage engine, that allows for automatic mapping from RDF to Java objects and is able to cope with time in RDF. It gives you RDF's flexibility of storing objects, their properties and relationships between objects while mapping these structures to first class java objects. This means that instead of handling abstractions of RDF triples in Java (which is what Jena is providing) you handle objects like (in the case of OxPoints) Building, College and can run operations.

With this Gaboto is Object-relational mapping, like Hibernate or Propel. However, Gaboto is built upon RDF instead of relational databases, which gives you a greater flexibility in defining relationships and changing your datamodel.

The mapping rules are all configured in an XML configuration file (see *3. Part II - Configuring Gaboto*) from which the necessary classes are automatically generated. So the basic steps in using Gaboto are:

1. Write configuration file, define objects, their properties and relationships.
2. Execute configuration script, that basically generates a jar file.
3. Plug in the jar file and start writing your system.

1.3.1 Gaboto versus Relational Database Approaches

At this point you might ask: If I have to explicitely define my entities why shouldn't I go for a relational database approach? The answer to this is simple: Although it is true that you have to define the objects you are planning on using it is very easy to change these objects and add new objects and relationships. It is a simple change in your configuration file and you are set to go. On the other hand, trying to change the datamodel in a system based on a relational database is usually a big hazard, resulting in altering and introducing new tables etc.

1.4 Further Reading

Erewhon Blog [http://oxforderewhon.wordpress.com/] The project's blog provides various information about OxPoints and Gaboto. Especially the following articles might be of interest:

OxPoints - Providing geodata for the University of Oxford An introduction to OxPoints - Part 1

OxPoints and the Semantic Web An introduction to OxPoints - Part 2

RDF - an Introduction An introduction to RDF

RDF and the Time Dimension - Part 1 An article about RDF, focusing on the problem of using RDF to store dimensional data.

RDF and the Time Dimension - Part 2 An article about RDF, focusing on the problem of using RDF to store dimensional data. The solution (using named graphs) described here forms the basis of the Gaboto data model.

The RDF Specification [http://www.w3.org/RDF/] Provides several links and documents about the Resource Description Framework.

RDF Primer [http://www.w3.org/TR/REC-rdf-syntax/] The RDF Primer provides a general introduction into the world of RDF.

Named Graphs [http://www.w3.org/2004/03/trix/] The website of a working group on named graphs.

Chapter 2

Part I - Using Gaboto

Gaboto was developed to be used as the underlying datamodel for the new OxPoints system. As this provides us with a fully configured Gaboto system, I will explain Gaboto through the example of OxPoints.

2.1 Setting up a Test-Environment in Eclipse

In order to try out the presented examples without setting up a full Gaboto system (which would include configuring the persistent storage, defining database connections etc.) I will show you how to set up a simple environment with which you can try out all the examples (and of course your own examples) presented in this manual. While just using Gaboto, you are free to choose any development environment you feel comfortable with. However, as Eclipse is my favorite development environment and we will need Eclipse when we start looking into how Gaboto can be extended, I will present these instructions for Eclipse.

2.1.1 Install Eclipse

If you do not yet have Eclipse installed, go to http://www.eclipse.org/downloads/ and grab the latest copy (currently Ganymede 3.4.1). Eclipse comes in many different packages for various tasks but the standard **Eclipse IDE for Java Developers** should be your package of choice. After downloading the package, the installation usually just consists of unzipping the package. Check the instructions for your OS for more information.

2.1.2 Creating the Project

Once you are in Eclipse, first get rid of the Welcome screen and then switch to the Java Perspective (Window/Open Perspective/Java, if Java is not directly visible go for Other and select Java from the following menu.). You can create a new Java project by right clicking in the **Package Explorer** (panel on the left-hand side) and then choosing new/Java Project. Type in a project name (e.g., Gaboto), make sure that you select Java 1.6 and click on Finish.

2.1.3 Preparations for Running the Examples

copy the folders lib and example-data (from the Gaboto.zip) into the directory (Gaboto) for your newly created project (i.e., just drag the files from the explorer/file browser into eclipse and drop them onto your object). Copy the Gaboto.jar into the new lib folder under your project.

To set the classpath you right-click your project and select **Properties**. You go to **Java Build Path**, then **Libraries** and click on **Add JARs...**. Select all the jars from the lib directory and use Open to add them to the classpath. When you've added all the jars, click on OK.

We can now create a new package under the source folder (right-click on **src**, select New/Package). Give it a name (e.g. example) and hit Finish. Create a new Class underneath it (right-click on the package and select New/Class). Give it the name Test1 and click Finish.

Inside the curly brackets for the public class Test1, add the following two methods:

```
/**
* Main entrance Point
*
* @param args
* @throws IOException
* @throws SAXException
* @throws ParserConfigurationException
* @throws GabotoException
*/
public static void main(String[] args) throws ParserConfigurationException, SAXException,
IOException, GabotoException {
// initialize Gaboto system
GabotoLibrary.init(GabotoConfiguration.fromConfigFile());
// load gaboto
Gaboto gaboto = GabotoFactory.getEmptyInMemoryGaboto();
File graphs = new File("graphs.rdf");
FileInputStream fos = new FileInputStream(graphs);
File cdg = new File("cdg.rdf");
FileInputStream cdg_fos = new FileInputStream(cdg);
gaboto.read(fos, cdg_fos);
gaboto.recreateTimeDimensionIndex();
processExample(gaboto);
}
/**
* Here we can test the examples.
*
* @param gaboto
* @throws GabotoException
* @throws FileNotFoundException
*/
private static void processExample(Gaboto gaboto) throws GabotoException, FileNotFoundException {
// the example goes here
System.out.println("Gaboto is up and running.");
}
```

You will need to add some import statements. Eclipse will automatically add these, if you use the QuickFixes (little light bulbs left of the code). Or you can use the autocompletion (CTRL+Space):

Go to the end of each term that is not recognized and click (CTRL+Space). If you chose the same names as suggested, your final class should now look like:

```
package example;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import javax.xml.parsers.ParserConfigurationException;
import org.oucs.gaboto.GabotoConfiguration;
import org.oucs.gaboto.GabotoLibrary;
import org.oucs.gaboto.entities.pool.GabotoEntityPool;
import org.oucs.gaboto.exceptions.GabotoException;
import org.oucs.gaboto.model.Gaboto;
import org.oucs.gaboto.model.GabotoFactory;
import org.oucs.gaboto.model.GabotoSnapshot;
import org.oucs.gaboto.timedim.TimeInstant;
import org.oucs.gaboto.transformation.json.GeoJSONPoolTransfomer;
import org.oucs.gaboto.vocabulary.OxPointsVocab;
import org.xml.sax.SAXException;
public class Test1 {
/**
* Main entrance Point
*
* @param args
* @throws IOException
* @throws SAXException
* @throws ParserConfigurationException
* @throws GabotoException
*/
public static void main(String[] args) throws ParserConfigurationException, SAXException,
IOException, GabotoException {
// initialize Gaboto system
GabotoLibrary.init(GabotoConfiguration.fromConfigFile());
// load gaboto
Gaboto gaboto = GabotoFactory.getEmptyInMemoryGaboto();
File graphs = new File("graphs.rdf");
FileInputStream fos = new FileInputStream(graphs);
File cdg = new File("cdg.rdf");
FileInputStream cdg_fos = new FileInputStream(cdg);
gaboto.read(fos, cdg_fos);
gaboto.recreateTimeDimensionIndex();
processExample(gaboto);
}
/**
* Here we can test the examples.
*
* @param gaboto
* @throws GabotoException
* @throws FileNotFoundException
*/
private static void processExample(Gaboto gaboto) throws GabotoException, FileNotFoundException {
// the example goes here
System.out.println("Gaboto is up and running.");
}
}
```

Running the example (right-click on the class Test1 and choose Run As/Java Application) should result in in printing *Gaboto is up and running* onto the console.
Note: the Console pane has a Maximise button at the right end of its tool bar. You can now use the method **processExample** to try out the examples.

2.1.4 The Gaboto Object

I will later tell you, that you should be careful ablut which Gaboto object you use for what purpose. For trying out the examples you can simply use the created Gaboto object (an in-memory object, that is not connected to any persistent storage).

2.2 The Gaboto Object Model

This chapter describes the most important objects in the Gaboto object model and how to use them. I will not describe objects in detail (that is describe all their methods and fields) but try to concentrate on describing the relationship between objects and their purpose. For an in depth description for each individual object please refer to the JavaDoc.

2.2.1 Time

Although Gaboto does not force you to model time, it is one of its key features. Before we can start to describe any of the ”Gaboto classes” we have to get to know the two main classes that are used througout Gaboto to work with time. [1]

2.2.1.1 TimeInstant

TimeInstant is used to describe a specific point in time. It is able to work with a resolution upto days[2], although, if uncertain, Gaboto is also able to handle time instants that are not exactly specified. We need this functionality in order to be able to record events that we knew took place in a certain year, but we neither know the month nor the day.

2.2.1.1.1 Examples

Creating new time instants:

[1] For more examples on the usage of TimeInstant and TimeSpan, have a look at the unit tests: TestTimeSpan.java and TestTimeInstant.java in org.oucs.gaboto.test.classes.

[2] The resolution might be refined in future versions.

```
TimeInstant ti1 = new TimeInstant(200,8,10);
//September 12th, 567
TimeInstant ti2 = new TimeInstant(567,8,11);
// March 2nd, 200
TimeInstant ti3 = new TimeInstant(200,2,1);
// April, 200
TimeInstant ti4 = new TimeInstant(200,3,null);
// year 200
TimeInstant ti5 = new TimeInstant(200,null,null);
```

Comparing time instants:

```
int i = ti1.compareTo(ti2);
System.out.println(i);
// Result: This prints out -1, since t1 is earlier than t2.
boolean eq = t4.equals(t5);
System.out.println(eq ? "true" : "false");
// Result: Prints out false. The two instants are not the same.
boolean eq2 = t4.aboutTheSame(t5);
System.out.println(eq2 ? "true" : "false");
// Result: Prints out true. The two instants are roughly the same.
```

2.2.1.2 TimeSpan

TimeSpan is used to describe a time span. Since time-owl, the ontology we use to describe the time dimension uses a start date and a duration to describe a time span rather than a start date and an end date, TimeSpan follows the same concept. However, it offers methods to convert one form to another, i.e. you can create a TimeSpan object from a start and an end date or ask a TimeSpan object to return its start or end date. [3][4]

2.2.1.2.1 Examples

Creating new time spans:

```
// begin: 100-04-11, lasts: 200 years, 6 months and 8 days.
TimeSpan ts0 = new TimeSpan(100,3,10,200,6,8);
// begin 1802, April, lasts: 200 years and 6 months
TimeSpan ts1 = new TimeSpan(1802,3,null,200,6,null);
// begin 1850-12-13, lasts: 100 years and 6 months
TimeSpan ts2 = new TimeSpan(1850,11,12,100,6,0);
```

Comparing time spans:

[3] Time spans can, as well as time instants be underspecified. However, if you do not specify a month for its start date you may not specify a month for the duration.

[4] Time spans do not have to have a defined end date. If a TimeSpan object is created without any specification for its duration, Gaboto interprets the TimeSpan as being unbound.

```
System.out.println( ts0.equals(ts1) ? "true" : "false" );
// Result: false
System.out.println( ts1.contains(ts2) ? "true" : "false" );
// Result: true
```

2.2.2 GabotoLibrary

The GabotoLibrary is used to initialize Gaboto. It also offers methods to generate the specific Gaboto java classes from a configuration file.

2.2.3 Gaboto

The Gaboto object is your gateway to the underlying data. It provides low-level methods to directly add or remove RDF triples from the graphs but also offers methods to work with high-level Java objects such as GabotoEntities (see *2.2.5. GabotoEntity*) or GabotoTimeBasedEntities (see *2.2.7. GabotoTimeBasedEntity*).

Besides adding and changing data, the Gaboto object allows you to perform simple queries (e.g. does entity x exist) and is able to create snapshots (see *2.2.4. GabotoSnapshot*). Snapshots are a projection of parts of the data to one flat RDF Graph which is much easier to query than data distributed over various named graphs. The snapshot, containing all the data that was valid at some point in 1900 could, for example be generated like this:

```
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
GabotoSnapshot snap = gaboto.getSnapshot(new TimeInstant(1900, null, null));
```

2.2.3.1 Instantiating Gaboto Objects

Gaboto objects cannot be instantiated directly. The **GabotoFactory** object has to be used instead. It provides methods to create three different types of Gaboto objects:

Persistent Gaboto The resulting Gaboto object uses a database backend to store its data. However, this comes at the cost of performance, which is why this object should not be used to query the data, but only for updates.

In-Memory Gaboto The in-memory Gaboto object keeps all data in an in-memory store. It is linked to the persistent Gaboto object and informed about any changes in the dataset. It should therefore not be used to change data but only for query operations.

Empty In-Memory Gaboto The empty in-memory Gaboto can be used for testing. Since it is not linked to the persistent object any changes to this Gaboto model will not be persistent (unless of course you use one of the provided serialization methods).

2.2.4 GabotoSnapshot

The GabotoSnapshot is a wrapper around a projection of parts of the data in the Gaboto system from multiple RDF Graphs into one. GabotoSnapshots can then be used to more easily query the now flat RDF Graph or to create an internal model (see *2.2.5. GabotoEntity* and *2.2.6. GabotoEntityPool*).

GabotoSnapshots can be created using an Gaboto object.

2.2.5 GabotoEntity

Gaboto objects are the representation of the objects in your system in Java at a certain point in time. Gaboto will generate the necessary classes automatically from the configuration file. The GabotoEntity class is the common base class. In the case of OxPoints we are talking about Colleges, Units and Buildings and Gaboto provides the representation of these objects in Java.

GabotoEntities represent the state of an entity at a given point in time (or time span), i.e. all its properties are set to the values that hold at that particular point in time (or time span).

For now, we will concentrate on using existing objects. We will learn more about the implementation details in `part_configuring`.

2.2.5.1 Examples

GabotoEntities are first-class Java objects. In this case we are creating a new Building (Building is one of the defined objects in OxPoints):

```
Building b = new Building();
b.setURI("someURI");
b.setTimeSpan(new TimeSpan(200,0,0));
b.setName("foo");
assert(b.getName().equals("foo"));
assert(b.getTimeSpan.contains(new TimeSpan(300,0,0,100,0,0)));
```

You can store entities in Gaboto using the Gaboto object:

```
Gaboto gaboto = GabotoFactory.getPersistentGaboto();
gaboto.add(b);
```

You can load entities from Gaboto using the Gaboto object: [5]

```
Gaboto gaboto_m = GabotoFactory.getInMemoryGaboto();
Building b1 = (Building) gaboto_m.getEntity(uri, new TimeInstant(600,0,0));
```

Besides the type safe get and set mehtods, GabotoEntities offer a range of generic methods for querying for properties:

```
// lets assume we have an entity called entity
// GabotoEntity entity = this entity came from somewhere
// let's try to find the title property described by DC:title
String title = entity.getPropertyValue(DC.title_URI);
System.out.println(title);
```

[5] For any query operation an in-memory Gaboto should be used.

2.2.6 GabotoEntityPool

The GabotoEntityPool is a collection of GabotoEntities that you want to group together. You can either create an entity pool by adding entities by hand, or by parsing an GabotoSnapshot (or a Jena Model) and automatically extracting entities from the specified source. Simple rules and filters can be specified to guide the creation process.

Entity pools can then be used to do more work with the extracted entities or they can be automatically transformed into different output formats (such as RDF or KML). For most queries it will be sufficient to create an entity pool, containing all the entities that fulfil the query's requirements and then to transform the resulting pool into whatever output format the user requested.

2.2.6.1 Examples

Creating an entity pool containing all Colleges that currently exist:

```
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
// create snapshot
GabotoSnapshot snap = gaboto.getSnapshot(TimeInstant.now());
// configure creation
GabotoEntityPoolConfiguration config = new GabotoEntityPoolConfiguration(snap);
config.addAcceptedType(OxPointsVocab.College_URI);
// create entity pool
GabotoEntityPool pool = GabotoEntityPool.createFrom(config);
// transform to KML
String kml = new KMLPoolTransformer().transform(pool);
```

In the above example, we create an entity pool through a **GabotoEntityPoolConfiguration** object. This configuration object offers various properties to guide the pool's creation process. For a detailed reference refer to the GabotoEntityPoolConfiguration's JavaDoc. We will also tackle more examples in *2.3. Querying Gaboto.*

Using **EntityFilters** to find colleges near OUCS:

```
// define some constants
final double max_distance = 0.002;
final double lat_oucs = 51.760010;
final double long_oucs = -1.260350;
// create gaboto object
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
// create snapshot
GabotoSnapshot snap = gaboto.getSnapshot(TimeInstant.now());
// configure creation
GabotoEntityPoolConfiguration config = new GabotoEntityPoolConfiguration(snap);
config.addAcceptedType(OxPointsVocab.College_URI);
// add filter
config.addEntityFilter(new EntityFilter(){
```

```
@Override
public boolean filterEntity(GabotoEntity entity) {
College col = (College) entity;
// reject if no primary place is set
if(null == col.getPrimaryPlace())
return false;
// load location
Location loc = col.getPrimaryPlace().getLocation();
// reject if no location is set
if(null == loc)
return false;
double lat_diff = loc.getLatitude() - lat_oucs;
double long_diff = loc.getLongitude() - long_oucs;
double distance = Math.sqrt(lat_diff*lat_diff + long_diff*long_diff);
// if distance is small enough, allow entity to pass.
if(distance < max_distance)
return true;
// reject by default
return false;
}
});
// create entity pool
GabotoEntityPool pool = GabotoEntityPool.createFrom(config);
// transform to KML
String kml = new KMLPoolTransformer().transform(pool);
// output kml to console
System.out.println(kml);
```

EntityFilters are a powerful tool to select the entities that are added to the entity pool. However, since the filters can only be executed after an entity was instantiated, they do not provide a very good performance. They should therefore be used with care. We will learn more about this in *2.3. Querying Gaboto.*

2.2.7 GabotoTimeBasedEntity

GabotoEntities are the Java representation of an Gaboto object with fixed attributes as it existed at a given point in time (or time span). **GabotoTimeBasedEntity** represents an object over its entire lifespan containing all the properties and how they changed over time. The GabotoTimeBasedEntity can be used to iterate over the entity as it changed over time and it provides an easy to use interface for adding and changing time bound data.

2.2.7.1 Examples

Loading an entity and traversing over all changed versions:

```
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
GabotoTimeBasedEntity entityTB = GabotoTimeBasedEntity.loadEntity("someBuildingsURI", gaboto);
System.out.println(entityTB.getType());
// prints Building, if the URI was indeed a Building's URI
Iterator<GabotoEntity> it = entityTB.iterator();
while(it.hasNext()){
Building b = (Building) it.next();
System.out.println("During " + b.getTimeSpan() + " " + b.getUri() + "'s name was: " + b.getName());
}
```

Creating an GabotoTimeBasedEntity:

```
TimeSpan ts = new TimeSpan(500,0,0,500,10,10);
GabotoTimeBasedEntity       entityTB       =       new       GabotoTimeBasedEntity(Building.class,
TestUtils.generateRandomURI(), ts);
entityTB.addProperty(DC.title, name1);
```

2.3 Querying Gaboto

This chapter describes different approaches to query the Gaboto system presenting their advantages and disadvantages. Since Gaboto is still in a very early phase of development and we don't have any reliable experiences with the performance of the different query types described in this chapter, we can only present assumptions.

2.3.1 Different Query Types

There are many different ways in which you can query data from the Gaboto system. You can use high-level objects such as the GabotoEntityPool or use low-level methods like SPARQL.

2.3.1.1 The Gaboto Object Model

Using the Gaboto object model, you are presented with a rich high-level easy to use interface to the data. The downside to this approach is that, high-level approaches always perform (slightly) worse than optimized low-level approaches. However, if you follow certain guidelines when working with the Gaboto object model, the performance disadvantages should not be too severe and the advantages (maintainability, time for development, etc.) should outweigh the disadvantages. [6]

2.3.1.1.1 Point Queries

If we want to get information about a specific entity we can simply ask the Gaboto object for it. We may want to get the details of the entity at a specific point in time or we may be interested in getting the details of how it has changed over time.

```
// get Gaboto object
```

[6] While it is rather difficult to write bad SQL, that cannot be properly optimized by today's query optimizers, it is very easy to write bad SPARQL queries. The algorithms used in Jena to optimize queries are currently still far from being able to produce the results their SQL counterparts are able of.

```
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
// get entity at a specific point in time
GabotoEntity entity = gaboto.getEntity("entityURI", TimeInstant.now());
// or instead get details about how the entity has changed over time
GabotoTimeBasedEntity entityTB = gaboto.getEntityOverTime("entityURI");
```

These objects can then be used to ascertain the information you are interested in.

Finding Entities

To run point queries you first need to find the URIs of the entities in question. Gaboto and GabotoSnapshot provide helper methods for this task.

Finding Entities in Snapshots

If you already have a snapshot that you are working with, you can create entities by calling the method **loadEntitiesWithProperty**. This allows you to easily create entity pools that contain only those entities that have a specific property or entities that have that property with a specific value.

```
// load snapshot
GabotoSnapshot snap = gaboto.getSnapshot(TimeInstant.now());
// get all entities that have the title "Somerville College"
GabotoEntityPool pool = snap.loadEntitiesWithProperty(DC.title, "Somerville College");
```

In RDF properties are described using URIs. The presented methods therefore take a URI or a Jena abstraction called *Property* as input for defining the property you are looking for.

Finding Entities in the Gaboto Object

The Gaboto object provides two mechanisms to find entities. You can either ask for a set of URIs (*getEntityURIsFor*) or directly create GabotoTimeBasedEntities (loadEntitiesOverTimeWithProperty).

This example lists all URIs of entities that have a title:

```
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
// get list of URIs
Collection<String> uris = gaboto.getEntityURIsFor(DC.title);
for(String uri : uris)
System.out.println(uri);
```

Get time based entities that have the title ”Somerville College”:

```
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
// get entities
```

```
Collection<GabotoTimeBasedEntity>  tbEntities  =  gaboto.loadEntitiesOverTimeWithProperty(DC.title,
"Somerville College");
```

Note: although there is likely to be only one element in the collection when the title is ”Somerville College”, the collection will have many elements if the title is ”Lodge”.

2.3.1.1.2 GabotoEntityPool - Complex Queries

The GabotoEntityPool offers many configuration properties to guide its creation process. These are outsourced to the GabotoEntityPoolConfiguration object that is then used as input for GabotoEntityPool’s **createFrom** method. For a complete description of all configuration options, refer to the JavaDoc of GabotoEntityPoolConfiguration.

Types

With the methods **addAcceptedType**, **setAcceptedTypes**, **addUnacceptedType** and **setUnacceptedTypes** you can control which types of entity are to be added to the pool. By default all types are accepted.

Create pool that only contains colleges:

```
GabotoEntityPoolConfiguration config = new GabotoEntityPoolConfiguration(snapshot);
config.addAcceptedType(OxPointsVocab.College_URI);
GabotoEntityPool pool = GabotoEntityPool.createFrom(config);
```

Create pool that contains everything but colleges:

```
GabotoEntityPoolConfiguration config = new GabotoEntityPoolConfiguration(snapshot);
config.addUnacceptedType(OxPointsVocab.College_URI);
GabotoEntityPool pool = GabotoEntityPool.createFrom(config);
```

Resources

If you only want to transform some specific RDF resources into GabotoEntities, then you can use the methods **addResource** and **setResources** to define which resources you want transformed. By default, entities are created for all resources that can be found.

A list of resources can be obtained by either using Jena’s API (use GabotoSnapshot.getModel() to obtain the underlying Jena Model of a snapshot) or by running SPARQL queries (see *2.3.1.2. SPARQL Queries*).

Dereferencing of Objects

Objects in an Gaboto system usually have relationships with other objects. In OxPoints a College has, for example, a primary building. These relationships have to be somehow dereferenced when creating GabotoEntities during a pool’s creation. The default method is called **lazy dereferencing**, which means, that relationships are only dereferenced if a relationship is actually accessed. If you, for example, created a pool containing College objects, the referenced Building objects (the college's primary buildings) will only be created if your program accesses the college's primary building property.

If you know that you will need full derefercning (this also includes references of references of references ...), you can set the parameter *enableLazyDereferencing(false)*.

Add Referenced Entities to Pool

Defines whether referenced entities are added to the pool directly. By default referenced objects are not added directly, which means that if you ask the pool to give you a collection of all its entities these will not be present.

Entity Filters

Entity Filters are called after an entity was loaded and before it was added to the pool. It will only be added to the pool if it passes all entity filters. Entity filters are an easy way to guide the pool creation, but since entities are first loaded they do not provide a very good performance.

Simulate accepted types parameter with entity filters (only accept colleges):

```
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
GabotoSnapshot snap = gaboto.getSnapshot(TimeInstant.now());
// configure pool
GabotoEntityPoolConfiguration config = new GabotoEntityPoolConfiguration(snap);
// add entity filter
config.addEntityFilter(new EntityFilter(){
@Override
public boolean filterEntity(GabotoEntity entity) {
if(!(entity instanceof College))
return false; // reject, if entity is not a college
return true; // pass
}
});
```

Simulate unaccepted types (accept everything but colleges):

```
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
GabotoSnapshot snap = gaboto.getSnapshot(TimeInstant.now());
// configure pool
GabotoEntityPoolConfiguration config = new GabotoEntityPoolConfiguration(snap);
// add entity filter
config.addEntityFilter(new EntityFilter(){
// this filter only applies to entities of type College
@Override
public Class<? extends GabotoEntity> appliesTo(){
return College.class;
}
@Override
public boolean filterEntity(GabotoEntity entity) {
// reject everything
return false;
}
});
```

Resource Filters

Resource Filters are called before an entity is loaded and work with Jena's **Resource** object. Since they are called a bit earlier in the creation process, they provide a better performance than Entity Filters.

The implementation is similar to Entity Filters. A method **appliesTo** specifies to which entity types the filter will be applied. The filter method is called **filterResource**.

Example: Find all colleges that do not have a title property with the value ”Somerville College”:

```
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
GabotoEntityPoolConfiguration config = new GabotoEntityPoolConfiguration(snap);
config.addAcceptedType(OxPointsVocab.College_URI);
config.addResourceFilter(new ResourceFilter(){
@Override
public Class<? extends GabotoEntity> appliesTo(){
return College.class; // unnecessary because of accepted types.
}
public boolean filterResource(Resource res){
// everything that is not somerville
return ! res.hasProperty(DC.title, "Somerville College");
}
});
GabotoEntityPool pool = GabotoEntityPool.createFrom(config);
```

Note: the full name of the Resource class is com.hp.hpl.jena.rdf.model.Resource.

Predefined Resource Filters

PropertyExistsFilter Tests whether resources that apply have a specific property.

PropertyNotExistsFilter Tests whether resources that apply do not have a specific property.

PropertyEqualsFilter Tests whether resources that apply have a specific property with a given value.

PropertyNotEqualsFilter Tests whether resources that apply have a specific property with a value unequal to a given value.

2.3.1.2 SPARQL Queries

SPARQL is the W3C query language for querying RDF data. It is a read-only language and uses a syntax somewhere in between SQL and Datalog. Basically there are 4 different types of

SPARQL queries. SELECT queries are similar to SQL SELECT. You specify a set of variables and restrictions and the SPARQL engine goes off and tries to find bindings for these variables that match the defined restrictions. CONSTRUCT queries enable you to create a new RDF Graph, that can then be used for further processing. DESCRIBE queries describe entire objects and ASK queries return a boolean, indicating whether or not your query statement can be matched or not.

For information about how to write SPARQL queries, have a look at the Further Reading section.

The GabotoSnapshot object offers methods to easily run SPARQL queries. If you want to work directly with Jena's API, aks the snapshot for the underlying Jena Model object (*getModel()*).

2.3.1.2.1 Prefixes

To make query writing a bit easier, you can use the static method **GabotoPredefined-Queries.getStandardPrefixes()** to create a set of prefixes for you, that can then be used in your query.

```
String query = GabotoPredefinedQueries.getStandardPrefixes();
System.out.println(query);
/* prints:
PREFIX dc:<http://purl.org/dc/elements/1.1/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gaboto:<http://ns.ox.ac.uk/namespace/gaboto/2009/03/owl#>
PREFIX data:<http://ns.ox.ac.uk/namespaces/oxpoints/data/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema>
PREFIX rdfcon:<http://ns.ox.ac.uk/namespace/rdfcon/2009/02/owl#>
PREFIX owl-time:<http://www.w3.org/2006/time#>
PREFIX rdfg:<http://www.w3.org/2004/03/trix/rdfg-1/>
PREFIX oxp:<http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#>
*/
```

2.3.1.2.2 SELECT Queries

Print the name of all resources in a snapshot.

```
// load snapshot
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
GabotoSnapshot snap = gaboto.getSnapshot(TimeInstant.now());
// build query
String query = GabotoPredefinedQueries.getStandardPrefixes();
query += "SELECT ?uri ?name WHERE { ?uri  dc:title  ?name . }";
// execute SSELECT query
snap.execSPARQLSelect(query, new QuerySolutionProcessorImpl(){
public void processSolution(QuerySolution solution) {
System.out.println(solution.getResource("uri").getURI()   +   "   has   the   name:      "   +
solution.getLiteral("name").getValue());
}
});
```

2.3.1.2.3 CONSTRUCT Queries

Create a new snapshot that contains all colleges that start with **b**:

```
// load snapshot
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
GabotoSnapshot snap = gaboto.getSnapshot(TimeInstant.now());
// build query
String query = GabotoPredefinedQueries.getStandardPrefixes();
query += "CONSTRUCT { ?a ?b ?c. } WHERE {\n" +
"?a rdf:type oxp:College . \n" +
"?a dc:title ?title . \n" +
"FILTER regex(?title, \"^b\", \"i\") . \n" +
"?a ?b ?c . \n" +
"}";
// execute query
GabotoSnapshot newSnapShot = snap.execSPARQLConstruct(query);
```

7

2.3.1.2.4 DESCRIBE Queries

Create a snaphsot that only contains colleges:

```
// load snapshot
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
GabotoSnapshot snap = gaboto.getSnapshot(TimeInstant.now());
// build query
String query = GabotoPredefinedQueries.getStandardPrefixes();
query += "DESCRIBE ?x WHERE { ?x rdf:type oxp:College }";
// create new snapshot
GabotoSnapshot describedSnap = snap.execSPARQLDescribe(query);
```

2.3.1.2.5 ASK Queries

Ask a simple question:

[7] If you use the newly created snapshot to create an entity pool, the primary buildings will cannot be dereferenced in the resulting College objects since they are not part of the snapshot. If you want to achieve this change the pool's snapshot:

```
  // create resultPool from model GabotoEntityPool resultPool = newSnapShot.buildEntityPool();
  // change snapshot resultPool.setSnapshot(snap)
```

```
// load snapshot
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
GabotoSnapshot snap = gaboto.getSnapshot(TimeInstant.now());
// build query
String query = GabotoPredefinedQueries.getStandardPrefixes();
query += "ASK { data:abc123  dc:title  \"Classics Lending Library\" . }";
// execute query
System.out.println( snap.execSPARQLAsk(query) ? "true" : "false" );
```

2.3.2 The GabotoQuery Interface

The **GabotoQuery** interface and especially its abstract implementation **GabotoQueryImpl** are meant to provide a starting point for writing reusable queries that can be used to generate different output formats. They provide an interface to the user to specify an output format and will handle the transformation to that format automatically. The only thing the creator has to worry about is how to get the data s/he is interested in.

2.3.2.1 Using Existing Queries

Using an GabotoQuery object is really simple. Instantiate the object, configure it, call **execute** and tell it what output format it should generate. If we wanted to have the RDF for all carparks in the system in XML abbreviated form, we could use the **ListOfTypedEntities** query:

```
// get Gaboto object
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
// instantiate query
GabotoQuery    loteQuery    =    new    ListOfTypedEntities(gaboto,     OxPointsVocab.Carpark_URI,
TimeInstant.now());
// perform query and get output
String output = (String) loteQuery.execute(GabotoQuery.FORMAT_RDF_XML_ABBREV);
// print output
System.out.println(output);
/**
Result:
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:j.0="http://www.opengis.net/gml/"
xmlns:j.1="http://ns.ox.ac.uk/namespace/Gaboto/2009/02/owl#">
<j.1:Carpark rdf:about="http://ns.ox.ac.uk/namespaces/Gaboto/data/carpark/6ed58a8a">
<j.1:hasLocation>
<j.0:Point>
<j.0:pos>51.75966716041378 -1.25807142865529</j.0:pos>
</j.0:Point>
</j.1:hasLocation>
<dc:title>Keble Triangle</dc:title>
```

```
</j.1:Carpark>
<j.1:Carpark rdf:about="http://ns.ox.ac.uk/namespaces/Gaboto/data/carpark/88f2cd48">
<j.1:hasLocation>
<j.0:Point>
<j.0:pos>51.76312928396877 -1.259968959743287</j.0:pos>
</j.0:Point>
</j.1:hasLocation>
<dc:title>Bevington Road</dc:title>
</j.1:Carpark>
.....
</RDF>
*/
```

For a list of all supported output formats you can call the **getSupportedFormats** method. Currently there are generic transformations (we will have a look at these in detail in *4.2. Output Transformation*) available for the following formats:

RDF Various RDF formats are supported.

KML Google's geospatial XML markup.

JSON JavaScript Object Notation

Jena Model A Jena Model object (see «chap_jenaAPI»).

Entity Pool An GabotoEntityPool object.

2.3.2.2 Writing new custom queries

Let us now try writing a query that, given a number **n** and a title finds us the nearest **n** Colleges to an entity with this title (at a given point in time). The sample implementation I will show you here is not in any case optimized for speed but uses easy to understand highlevel methods to achieve its goal.

The first step is to create a subclass to GabotoQueryImpl and make it configurable in the constructor. We also provide for set methods to set the two parameters title and number.

```
public class CollegesNearEntity extends GabotoQueryImpl {
private String title;
private TimeInstant timeInstant;
private int number;
private GabotoSnapshot snapshot;
private List<College> listOfColleges;
public CollegesNearEntity(String title, int number, TimeInstant ti) throws GabotoException {
super();
this.title = title;
this.timeInstant = ti;
this.number = number;
}
public CollegesNearEntity(Gaboto gaboto, String title, int number, TimeInstant ti) throws
GabotoException {
super(gaboto);
this.title = title;
this.timeInstant = ti;
this.number = number;
}
public void setTitle(String title) {
this.title = title;
}
public void setNumber(int number) {
this.number = number;
}
@Override
protected void doPrepare() throws GabotoException {
}
@Override
public int getResultType() {
}
@Override
protected Object execute() throws GabotoException {
}
}
```

You should always provide a constructor with a Gaboto object and one without it (in this case Gaboto tries to instantiate an in-memory Gaboto object).

GabotoQueryImpl defines three abstract methods that we have to override:

doPrepare **doPrepare** should be used to perform tasks that only have to be done once, before the query can be executed. We will use this to create a snapshot and produce a list of all colleges.

getResultType Basically, there are two types of queries that you can write. You can either create a Jena Model (for example using SPARQL) or low-level methods or you can create an GabotoEntityPool that contains all the entities you are interested in. **getResultType** defines which kind of query you will write. The two possible return values are:

- GabotoQueryImpl.RESULT_TYPE_ENTITY_POOL
- GabotoQueryImpl.RESULT_TYPE_MODEL

In this case we will create an entity pool with the resulting **n** colleges.

execute In this method you perform the actual query. The return value is either an GabotoEntityPool or a Jena Model (depending on the return value of **getResultType**).

We can now implement the three methods.

2.3.2.2.1 The resulting class

```
package org.oucs.gaboto.sample;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;
import org.oucs.gaboto.beans.Location;
import org.oucs.gaboto.entities.College;
import org.oucs.gaboto.entities.GabotoEntity;
import org.oucs.gaboto.entities.pool.GabotoEntityPool;
import org.oucs.gaboto.entities.pool.GabotoEntityPoolConfiguration;
import org.oucs.gaboto.exceptions.GabotoException;
import org.oucs.gaboto.model.Gaboto;
import org.oucs.gaboto.model.GabotoSnapshot;
import org.oucs.gaboto.model.query.GabotoQueryImpl;
import org.oucs.gaboto.timedim.TimeInstant;
import org.oucs.gaboto.vocabulary.OxPointsVocab;
import com.hp.hpl.jena.vocabulary.DC;
import com.hp.hpl.jena.vocabulary.DC_11;
public class CollegesNearEntity extends GabotoQueryImpl {
private String title;
private TimeInstant timeInstant;
private int number;
private GabotoSnapshot snapshot;
private List<College> listOfColleges;
public CollegesNearEntity(String title, int number, TimeInstant ti) throws GabotoException {
super();
this.title = title;
this.timeInstant = ti;
this.number = number;
}
public CollegesNearEntity(Gaboto gaboto, String title, int number, TimeInstant ti) throws
GabotoException {
super(gaboto);
this.title = title;
this.timeInstant = ti;
this.number = number;
}
public void setTitle(String title) {
this.title = title;
}
public void setNumber(int number) {
this.number = number;
```

```
}
@Override
protected void doPrepare() throws GabotoException {
// create snapshot
snapshot = getGaboto().getSnapshot(timeInstant);
// get all colleges
GabotoEntityPoolConfiguration config = new GabotoEntityPoolConfiguration(snapshot);
config.addAcceptedType(OxPointsVocab.College_URI);
GabotoEntityPool colleges = GabotoEntityPool.createFrom(config);
// find colleges we are interested in
listOfColleges = new ArrayList<College>();
for(GabotoEntity en : colleges.getEntities()){
// cast to college
College col = (College) en;
// get its location
Location colLoc = (Location) col.getPropertyValue(OxPointsVocab.hasLocation);
if(null == colLoc)
continue;
// add to results if not enough results yet
listOfColleges.add(col);
}
}
@Override
public int getResultType() {
return GabotoQueryImpl.RESULT_TYPE_ENTITY_POOL;
}
@Override
protected Object execute() throws GabotoException {
// find entity with name
GabotoEntityPool entities = snapshot.loadEntitiesWithProperty(DC_11.title, title);
Iterator<GabotoEntity> it = entities.getEntities().iterator();
if(! it.hasNext())
throw new IllegalArgumentException("There is no entity with that title.");
GabotoEntity entity = it.next();
// we do not know what kind of entity we have
// but we are interested in its location. We
// can get to this property, by calling its getPropertyValue
// method.
Location loc = (Location) entity.getPropertyValue(OxPointsVocab.hasLocation);
if(null == loc)
throw new IllegalArgumentException("The entity " + entity.getUri() + " does not have a location.");
// store latitude and longitude
final double lat = loc.getLatitude();
final double _long = loc.getLongitude();
// sort results
```

```
Collections.sort(listOfColleges, new Comparator<College>(){
public int compare(College c1, College c2) {
// distance of college 1
Location loc = (Location) c1.getPropertyValue(OxPointsVocab.hasLocation);
double distX = Math.abs(lat - loc.getLatitude());
double distY = Math.abs(_long - loc.getLongitude());
double dis1 = Math.sqrt(distX*distX + distY*distY);
// distance of college 2
loc = (Location) c2.getPropertyValue(OxPointsVocab.hasLocation);
distX = Math.abs(lat - loc.getLatitude());
distY = Math.abs(_long - loc.getLongitude());
double dis2 = Math.sqrt(distX*distX + distY*distY);
if(dis1 < dis2)
return -1;
else if(dis1 > dis2)
return 1;
return 0;
}
});
//create result pool
GabotoEntityPool resultPool = new GabotoEntityPool(getGaboto(), snapshot);
for(int i = 0; i < number && i < listOfColleges.size(); i++)
resultPool.addEntity(listOfColleges.get(i));
return resultPool;
}
}
```

This query can now be used to provide results in many different formats:

```
// create query object that uses in memory Gaboto
GabotoQuery query = new CollegesNearEntity("Somerville College", 5, TimeInstant.now());
String output = (String) query.execute(GabotoQuery.FORMAT_KML);
System.out.println(output);
```

Resulting in:

```
<kml xmlns="http://www.opengis.net/kml/2.2"
>
  <Document xmlns="http://www.opengis.net/kml/2.2"
  >
    <Placemark xmlns="http://www.opengis.net/kml/2.2"
    >
      <name xmlns="http://www.opengis.net/kml/2.2"
      >St Anne's College</name>
      <Point xmlns="http://www.opengis.net/kml/2.2"
      >
```

```
        <coordinates xmlns="http://www.opengis.net/kml/2.2"
        >-1.262354850769043,51.76193031549332</coordinates>
      </Point>
    </Placemark>
    <Placemark xmlns="http://www.opengis.net/kml/2.2"
    >
      <name xmlns="http://www.opengis.net/kml/2.2"
      >Somerville College</name>
      <Point xmlns="http://www.opengis.net/kml/2.2"
      >
        <coordinates xmlns="http://www.opengis.net/kml/2.2"
        >-1.2613785266876,51.75943345494568</coordinates>
      </Point>
    </Placemark>
    <Placemark xmlns="http://www.opengis.net/kml/2.2"
    >
      <name xmlns="http://www.opengis.net/kml/2.2"
      >Green Templeton College</name>
      <Point xmlns="http://www.opengis.net/kml/2.2"
      >
        <coordinates xmlns="http://www.opengis.net/kml/2.2"
        >-1.262467503547,51.761257969688444</coordinates>
      </Point>
    </Placemark>
    <Placemark xmlns="http://www.opengis.net/kml/2.2"
    >
      <name xmlns="http://www.opengis.net/kml/2.2"
      >Regent's Park College</name>
      <Point xmlns="http://www.opengis.net/kml/2.2"
      >
        <coordinates xmlns="http://www.opengis.net/kml/2.2"
        >-1.2606704235076904,51.75731499956544</coordinates>
      </Point>
    </Placemark>
    <Placemark xmlns="http://www.opengis.net/kml/2.2"
    >
      <name xmlns="http://www.opengis.net/kml/2.2"
      >St Benet's Hall</name>
      <Point xmlns="http://www.opengis.net/kml/2.2"
      >
        <coordinates xmlns="http://www.opengis.net/kml/2.2"
        >-1.26084208488464,51.75807207653707</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

And:

```
output = (String) query.execute(OxPointsQuery.FORMAT_RDF_XML_ABBREV);
System.out.println(output);
```

Resulting in:

```
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

   xmlns:j.1="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:j.0="http://nwalsh.com/rdf/vCard#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <College xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

      about="http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/a43adce8">
    <hasHomepage xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

        resource="http://www.st-benets.ox.ac.uk"/>
    <primaryPlace xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

        resource="http://ns.ox.ac.uk/namespaces/oxpoints/data/building/102e814f"/>
    <title xmlns="http://purl.org/dc/elements/1.1/"
    >St Benet's Hall</title>
    <adr xmlns="http://nwalsh.com/rdf/vCard#"
    >
      <Address xmlns="http://nwalsh.com/rdf/vCard#"
      >
        <street-address xmlns="http://nwalsh.com/rdf/vCard#"
        >38 St Giles', Oxford</street-address>
        <postal-code xmlns="http://nwalsh.com/rdf/vCard#"
        >OX1 3LN</postal-code>
      </Address>
    </adr>
  </College>
  <College xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

      about="http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/c16e28a7">
    <hasITHomepage xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

        resource="http://www.st-annes.ox.ac.uk/study/undergraduate/computing_it.html"/>
    <hasHomepage xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

        resource="http://www.st-annes.ox.ac.uk/"/>
    <primaryPlace xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

        resource="http://ns.ox.ac.uk/namespaces/oxpoints/data/building/275ab0d1"/>
    <title xmlns="http://purl.org/dc/elements/1.1/"
    >St Anne's College</title>
    <adr xmlns="http://nwalsh.com/rdf/vCard#"
    >
      <Address xmlns="http://nwalsh.com/rdf/vCard#"
      >
        <street-address xmlns="http://nwalsh.com/rdf/vCard#"
        >Oxford</street-address>
        <postal-code xmlns="http://nwalsh.com/rdf/vCard#"
        >OX2 6HS</postal-code>
      </Address>
    </adr>
  </College>
  <College xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

      about="http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/678b78ec">
    <hasHomepage xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"
    resource="http://www.rpc.ox.ac.uk/rpc/"/>
  <primaryPlace xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

      resource="http://ns.ox.ac.uk/namespaces/oxpoints/data/building/ae870de6"/>
  <title xmlns="http://purl.org/dc/elements/1.1/"
  >Regent's Park College</title>
  <adr xmlns="http://nwalsh.com/rdf/vCard#"
  >
    <Address xmlns="http://nwalsh.com/rdf/vCard#"
    >
      <street-address xmlns="http://nwalsh.com/rdf/vCard#"
      >Oxford</street-address>
      <postal-code xmlns="http://nwalsh.com/rdf/vCard#"
      >OX1 2LB</postal-code>
    </Address>
  </adr>
</College>
<College xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

    about="http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/b039a03c">
  <hasITHomepage xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

      resource="http://www.some.ox.ac.uk/students/it_services/"/>
  <hasHomepage xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"
    resource="http://www.some.ox.ac.uk/"/>
  <primaryPlace xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

      resource="http://ns.ox.ac.uk/namespaces/oxpoints/data/building/34e05e71"/>
  <title xmlns="http://purl.org/dc/elements/1.1/"
  >Somerville College</title>
  <adr xmlns="http://nwalsh.com/rdf/vCard#"
  >
    <Address xmlns="http://nwalsh.com/rdf/vCard#"
    >
      <street-address xmlns="http://nwalsh.com/rdf/vCard#"
      >Oxford</street-address>
      <postal-code xmlns="http://nwalsh.com/rdf/vCard#"
      >OX2 6HD</postal-code>
    </Address>
  </adr>
</College>
<College xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

    about="http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/fe041602">
  <hasHomepage xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"
    resource="http://www.gtc.ox.ac.uk/"/>
  <primaryPlace xmlns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"

      resource="http://ns.ox.ac.uk/namespaces/oxpoints/data/building/8b9b83f2"/>
  <title xmlns="http://purl.org/dc/elements/1.1/"
  >Green Templeton College</title>
  <adr xmlns="http://nwalsh.com/rdf/vCard#"
  >
    <Address xmlns="http://nwalsh.com/rdf/vCard#"
    >
      <street-address xmlns="http://nwalsh.com/rdf/vCard#"
      >Oxford</street-address>
```

```
        <postal-code xmlns="http://nwalsh.com/rdf/vCard#"
        >OX2 6HG</postal-code>
      </Address>
    </adr>
  </College>
</RDF>
```

2.3.3 Further Reading

SPARQL [http://www.w3.org/TR/rdf-sparql-query/] Description of the W3C Query Language for querying RDF data.

ARQ [http://jena.sourceforge.net/ARQ/] Jena's SPARQL processor

2.4 Managing Data

In this chapter I'll describe in detail how to add, change and delete data in Gaboto and give some insights on how Gaboto will store data that has been added. As with queries, Gaboto provides low and high-level methods for managing data. However, for most cases you will never have to bother about using the low-level methods.

The objects we will be investigating in this chapter are GabotoEntity or rather the specific subclasses, GabotoTimeBasedEntity and the Gaboto object. While GabotoEntities store the data that we want to add, the Gaboto object provides the methods to actually persist the objects in the underlying RDF store.

While we were using an in-memory Gaboto object while querying the system, we will be working with a persistent object when managing it.

2.4.1 Importing and Exporting Data

The Gaboto object offers the methods **read**, **write** and **writeCDG** which allow you to export/import data from/into various RDF formats. If there is already data in the system, the imported data will be merged with the existing data.

2.4.1.1 Example: Export to files

```
Gaboto gaboto = GabotoFactory.getInMemoryGaboto();
// store named graphs in file graphs.rdf
File graphs = new File("graphs.rdf");
FileOutputStream fos = new FileOutputStream(graphs);
// store context description graph in cdg.rdf
File cdg = new File("cdg.rdf");
FileOutputStream cdg_fos = new FileOutputStream(cdg);
// perform the actual export
gaboto.write(fos);
gaboto.writeCDG(cdg_fos);
```

2.4.1.2 Example: Import from files

```
// create empty Gaboto object
// this could of course also be a persistent object
Gaboto gaboto = GabotoFactory.getEmptyInMemoryGaboto();
File graphs = new File("graphs.rdf");
FileInputStream fis = new FileInputStream(graphs);
File cdg = new File("cdg.rdf");
FileInputStream cdg_fis = new FileInputStream(cdg);
gaboto.read(fis, cdg_fis);
// after the import we have to recreate the time index
gaboto.recreateTimeDimensionIndex();
```

2.4.2 Adding Data

We will first have a quick glance at the low level methods, that allow you to directly work with RDF constructs. We will then discuss the high-level methods in greater detail as you will hardly ever have to use anything else.

2.4.2.1 Adding RDF Triples

The Gaboto object offers the method **add** that takes a Jena Triple object and an optional TimeSpan as parameters. The time span describes the validity of the triple, i.e., during the specified time span the triple is valid. If no time span is provided, then the triple is universally valid, meaning, that it is valid at any point in time.

```
// get Gaboto object
Gaboto gaboto = GabotoFactory.getPersistentGaboto();
// create a triple
Triple t = new Triple(
Node.createURI( "someSubjectURI" ),
Node.createURI( "somePredicateURI" ),
Node.createLiteral( "some literal" )
);
// add the triple with a time span from 1900-2000
gaboto.add(new TimeSpan(1900,null,null,100,null,null),t);
```

In the above example, we added the triple

```
<someSubjectURI>  <somePredicateURI>  "some literal" .
```

to the graph that contains all triples that are valid from 1900-2000. If the graph did not yet exist in the system, **add** would have automatically created the new graph first. If we had not specified a time span, then the triple would have been added to the the special **generalKnowledgeGraph**, the graph that contains all universally valid triples.

2.4.2.2 Adding GabotoEntities and GabotoTimeBoundEntities

As discussed in *2.2. The Gaboto Object Model* GabotoEntity describes an entity as it exists at a given point in time or time span, where it has a fixed set of properties. In contrast, GabotoTimeBoundEntity describes an entity over its entire life span. The same applies, when these objects are persisted to the underlying RDF. Persisting an GabotoEntity will put all the triples that describe the entity into one graph. This can either be the GeneralKnowledgeGraph, in case, the entity did not have a life span associated with it. If it has, all the triples will be added to the graph that contains the valid data for this time span. Persisting an GabotoTimeBasedEntity object will usually result in spreading triples across multiple graphs.

Let us now create a new college and add it to the system.

```
// get Gaboto object
Gaboto gaboto = GabotoFactory.getPersistentGaboto();
// create college and set URI
College col = new College();
col.setUri(gaboto.generateID("college"));
// set properties
col.setTimeSpan(new TimeSpan(1900,null,null,100,null,null));
col.setName("AM College");
// persist college in RDF
gaboto.add(col);
```

If we happened to start off with an empty system, the resulting RDF would look like this (Syntax: TRIG):

```
@prefix dc:        <http://purl.org/dc/elements/1.1/> .
@prefix rdf:       <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
<http://ns.ox.ac.uk/namespaces/oxpoints/graphs#gkg.rdf> { }
<http://ns.ox.ac.uk/namespaces/gaboto/graphs#tg-1900~100-0-0> {<http://ns.ox.ac.uk/namespaces/oxpoints/data/coll
a       <http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#College> ;
dc:title "AM College" .
}
```

While this describes the college data, another entry is added to the **contextDescriptionGraph** (a special graph that describes all the graphs in the system), to describe the newly created **http://ns.ox.ac.uk/namespaces/gaboto/graphs#tg-1900~100-0-0**:

```
<http://ns.ox.ac.uk/namespaces/gaboto/graphs#tg-1900~100-0-0>
a       <http://www.w3.org/2004/03/trix/rdfg-1/Graph> ;
<http://ns.ox.ac.uk/namespace/rdfcon/2009/02/owl#hasTemporalDimension>
[ a       <http://www.w3.org/2006/time#Interval> ;
<http://www.w3.org/2006/time#hasBeginning>
[ a       <http://www.w3.org/2006/time#Instant> ;
<http://www.w3.org/2006/time#hasDateTimeDescription>
[ a       <http://www.w3.org/2006/time#DateTimeDescription> ;
<http://www.w3.org/2006/time#unitType>
<http://www.w3.org/2006/time#unitYear> ;
<http://www.w3.org/2006/time#year>
1900
]
] ;
<http://www.w3.org/2006/time#hasDurationDescription>
[ a       <http://www.w3.org/2006/time#DurationDescription> ;
<http://www.w3.org/2006/time#days>
0 ;
<http://www.w3.org/2006/time#months>
0 ;
<http://www.w3.org/2006/time#years>
100
]
] .
<http://ns.ox.ac.uk/namespaces/gaboto/graphs#gkg.rdf>
a       <http://ns.ox.ac.uk/namespace/rdfcon/2009/02/owl#GlobalKnowledgeGraph> .
```

2.4.2.2.1 URIs

When creating a new entity object that you want to add to Gaboto you have to assign it a unique id. For this purpose the Gaboto object provides the method **generateID** which makes sure that the generated ID is in fact unique. As a shortcut, GabotoEntity provides the static method **createNew**:

```
Gaboto gaboto = GabotoFactory.getPersistentGaboto();
College col = GabotoEntity.createNew(oxp, new College());
```

2.4.2.2.2 GabotoTimeBasedEntities

If you have an entity with a defined life span but a fixed set of properties, GabotoEntity is your friend. If you however, have properties that changed over time, you have to work with the slightly more complicated GabotoTimeBasedEntity object.

Let's try and describe parts of the history of **Oxford University Computing Services (OUCS)**. OUCS was founded in 1957 and was known as **Computing Laboratory** from 1957 to 1969. It then changed names (in fact it was split in two, but we ignore this fact for this example) and is since known as **Oxford University Computing Services**:

```
// get persistent Gaboto object
Gaboto gaboto = GabotoFactory.getPersistentGaboto();
// instantiate GabotoTimeBasedEntity of type unit with
// a open ended time span starting in 1957
GabotoTimeBasedEntity oucs = new GabotoTimeBasedEntity(Unit.class, gaboto.generateID("unit"), new
TimeSpan(1957,null,null));
// set the title from 1957 - 1969 and 1969 - dooms day
```

```
oucs.addProperty(new TimeSpan(1957,null,null,12,null,null), DC.title, "Computing Laboratory");
oucs.addProperty(new TimeSpan(1969,null,null), DC.title, "Oxford University Computing Services");
// add it to the system.
gaboto.add(oucs);
```

This produces the following RDF (again assuming that the system did not contain any data before this):

```
@prefix dc:        <http://purl.org/dc/elements/1.1/> .
@prefix rdf:       <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
<http://ns.ox.ac.uk/namespaces/gaboto/graphs#gkg.rdf> { }
<http://ns.ox.ac.uk/namespaces/gaboto/graphs#tg-1957~12-0-0> {<http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/
dc:title "Computing Laboratory" .
}
<http://ns.ox.ac.uk/namespaces/gaboto/graphs#tg-1957~dooms-day> {<http://ns.ox.ac.uk/namespaces/oxpoints/data/un
a       <http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Unit> .
}
<http://ns.ox.ac.uk/namespaces/gaboto/graphs#tg-1969~dooms-day> {<http://ns.ox.ac.uk/namespaces/oxpoints/data/un
dc:title "Oxford University Computing Services" .
}
```

In this case three new graphs were created. One to store the associate rdf:type (Unit), with an open ended time span from 1957 onwards, one from 1957-1969 claiming the unit's name is **Computing Laboratory** and another open ended graph (1969-onwards) claiming the unit's name is **Oxford University Computing Services**.
We can now query the system [8]:

```
Collection<GabotoTimeBasedEntity>    units    =    gaboto.loadEntitiesOverTimeWithProperty(DC.title,
"Computing Laboratory");
for(GabotoTimeBasedEntity entityTB : units){
System.out.println("Entity    "    +    entityTB.getUri()    +    "    is    of    type    "    +
entityTB.getEntityClass().getSimpleName());
Iterator<GabotoEntity> entityIterator = entityTB.iterator();
while(entityIterator.hasNext()){
Unit unit = (Unit)entityIterator.next();
System.out.println("\tDuring " + unit.getTimeSpan() + " the entity's name was " + unit.getName());
}
}
/**
Result:
Entity http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/cc9ff6ae is of type Unit
During 1957~12-0-0 the entity's name was Computing Laboratory
During 1969~dooms-day the entity's name was Oxford University Computing Services
```

[8]In a productive system, you should always use an in-memory Gaboto for any kind of query.

```
*/
```

As we have seen from the example, **GabotoTimeBasedEntity** is a generic object applicable for all entities and does therefore not provide specialized methods like, e.g., **setName**. One way of adding data is (as we have done in the example) to use the underlying RDF predicate to describe what kind of property you want to add. We will later see (in part *3. Part II - Configuring Gaboto*) where these are defined.

Another way to add properties to **GabotoTimeBasedEntity** is to use **GabotoEntity** objects. This is a less error-prone approach (but at the cost of more code lines), since you have the advantage of type aware methods. Let's redo the inital example with this approach:

```
GabotoTimeBasedEntity oucs = new GabotoTimeBasedEntity(Unit.class, gaboto.generateID("unit"), new
TimeSpan(1957,null,null));
// set the title from 1957 - 1969
Unit unit = new Unit();
unit.setName("Computing Laboratory");
unit.setTimeSpan(new TimeSpan(1957,null,null,12,null,null));
oucs.addEntity(unit);
// set the title from 1969 - dooms day
unit = new Unit();
unit.setName("Oxford University Computing Services");
unit.setTimeSpan(new TimeSpan(1969,null,null));
oucs.addEntity(unit);
// add entity to system
gaboto.add(oucs);
```

2.4.2.2.3 The role of the **rdf:type** Property

If you go back to the above examples, you can see that Gaboto added an rdf:type property on oucs in the most general graph. For one thing, this property describes the resource's type, telling Gaboto to load it into a **Unit** object. But it is also used by Gaboto to derive the life span of an entity. An entity's life span is the time span of the graph where its rdf:type is defined. You should therefore be careful, not to add any additional type triples, as this would result in a corrupted data store for Gaboto (although it is not technically wrong RDF). However, if you stick to the described high-levle methods, Gaboto should make sure, that this does not happen.

2.4.3 Removing and Changing Data

Technically removing and changing data is closely related, since Gaboto does not directly support changing data. What happens internally if you call one of the **changeEntity** methods of the **Gaboto** object is that Gaboto removes all traces of the entity (calling *Gaboto.purge*) and then to add the new version of the entity to the system. Besides the changeEntity methods, there are various **remove** methods that can be used for removing data. However, since these do not necessarily remove an entire object but just parts (and you should therefore be very careful, if you want to use them), you will usually go for the method **purge**, which deletes all triples that are associated with the entity you want to remove from the system.

Chapter 3

Part II - Configuring Gaboto

In this chapter we will learn how to set up Gaboto. This mainly includes defining the entities that you want to be able to talk about, their relationships to one another and their properties. We will describe the set up process through the example of the OxPOints system.

In order to better understand the configuration process, I will shortly describe what the system actually does with the finished configuration. Gaboto is a system that allows you to map RDF to Java classes and vice versa. In the previous examples we have already worked with GabotoEntities from the OxPoints system, such as **College** or **Building**. These classes are generated automatically by the configuration process. So what we are configuring is in fact the program to generate all the necessary classes that are needed to for a specialized Gaboto.

3.1 Ontologies

When setting up a new system based on Gaboto, the first step should be to think about the ontologies that describe your entities. Though it is not necessary to create ontologies it will help, as we have to refer to our entity types and properties as URIs and ontologies are the way to define these URIs.

In the OxPoints system we use (amongst others) the following ontologies:

OxPoints Ontology The OxPoints ontology defines university entities, places and some general entities, that we want to talk about in the OxPoints system.

Dublin Core The Dublin Core ontology provides a set of general and widely used properties (e.g., title). We use these, wherever applicable.

GEO_2007 GEO_2007 is an ontology produced by the W3C to capture some of GML's features in RDF. We use it to encode locations.

An OWL Ontology for vCards The OWL Ontology for vCards, created by Norman Walsh, allows you to encode vCards in RDF. We use it to encode addresses.

3.1.1 The OxPoints Ontology

Before we start going deeper into the configuration, let's have a closer look at the the OxPoints ontology. The OxPoints ontology defines all entities in the OxPoints system (Unit, College, Building, etc.) and certain properties that we could not find in other, more specialized ontologies.

We will later see, how we use the entities and properties, defined in this ontology in the Gaboto configuration.

```
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

    xml:base="http://ns.ox.ac.uk/namespace/gaboto/2009/03/owl#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!-- Describe the oxpoints ontology -->
  <Ontology xmlns="http://www.w3.org/2002/07/owl#"
    about="">
    <versionInfo xmlns="http://www.w3.org/2002/07/owl#"
    >0.1</versionInfo>
  </Ontology>
</RDF>
```

3.2 Properties

Gaboto is configured from a single XML based configuration file that defines the entities (see *2.2.5. GabotoEntity*), their properties and the relations between the different entities. Before we look at the configuration file, we will look at the different kind of properties that are supported by Gaboto and how they map to RDF.

Properties describe our entities. In RDF a property is represented (like our entities) by a URI and should be defined in an ontology. One of the properties we have seen in the previous examples is the **title** property from the Dublin Core ontology. In RDF a title would look somewhat like:

```
oxpdata:someEntityURI    dc:title    "The entity's title" .
```

3.2.1 Simple Properties

Simple properties are properties that are represented by one RDF triple, where the object is a literal. This can be, e.g., the title property **dc:title** from the previous example.

Gaboto currently supports the following types:

- String
- int
- double
- float
- boolean

3.2.2 Active vs. Passive Properties

When defining relationships between two entities, you have to define, with which entity the actual information is stored. Take the relationship between buildings and university units as an example. If we want to store that a unit occupies a certain building, we have two options[1]:

[1]In the Gaboto system we went for the first solution and store the information together with the unit.

1. We store the information with the unit. In RDF this could be:

```
        data:someUnitURI    oxp:occupies    data:someBuildingURI .
```

2. We store the information with the building. In RDF this could be:

```
        data:someBuildingURI    oxp:occupiedBy    data:someUnitURI .
```

Where you store your properties is really up to you and there is no definite right or wrong. However, it is faster to load properties that are directly stored with an entity, so if you expect to have lots of queries of the type *what buildings belong to a unit* and only very few of the type *what units occupy this building* then you might want to store the relationship with the unit for a better query performance.

3.2.3 Complex Properties

Complex properties are properties that consist of more than one peace of information. An address for example consists of a postal code, a street address and many more fields. In Gaboto, complex properties are called **GabotoBean**[2] and as **GabotoEntity**s they are first class java objects.

```
data:someUnitURI  vCard:adr [
a                  vCard:Address ;
vCard:postal_code    "OX3 7DQ" ;
vCard:street_address "Old Road Campus Research Building, off Roosevelt Drive, Oxford"
] ;
```

In RDF complex properties are stored using a blank node, which then contains all the necessary information.

3.2.4 Indirect Properties

Indirect properties are properties that are not stored explicitly but can be derived from other properties. We could for example define, that the location of a university unit is the location of its primary building (we assume, that units have a primary building and that buildings have a location). We can then derive the information by simply looking at the units primary building and searching for a location there. [3][4]

Indirect properties can also be used as fallback solutions. We could for example define, that the location of a **Place** is either stored directly with it, but that if no location is specified, we should look at its parent Place.

[2]The term bean was chosen because of the close relationship to Java Beans. A bit more on the subject of Java Beans, and their relationship to GabotoBeans and GabotoEntities will be presented in *4. Part III - Extending Gaboto*

[3]The indirect property does not necessarily have to be stored directly at the next entity. This can again point to a new entity, and the search is continued there.

[4]It is possible to create infinite loops through carelessly defining indirect properties. Gaboto currently does not provide any mechanisms to detect this.

3.2.5 Static Properties

Static properties are properties that are not stored in RDF but that are constructed using information from other properties. For example, you might want to add a description to an entity that consists of its title and its main website.

3.2.6 Unstored Properties

Unstored Properties are properties that have the same value as a stored property. In OxPoints we use unstored properties to define a parent relationship that is used by the transformation to KML.

3.2.7 Collections

In many cases you want to store lists of things. For example a building can be occupied by several units. We use RDF collections to achieve this. Currently only **Bags (unordered lists)** are implemented.

```
data:someUnitURI         oxp:occupies
[ a       rdf:Bag ;
rdf:_1  <http://ns.ox.ac.uk/namespaces/oxpoints/data/building/c8048b54> ;
rdf:_2  <http://ns.ox.ac.uk/namespaces/oxpoints/data/building/b11a46a1> ;
rdf:_3  <http://ns.ox.ac.uk/namespaces/oxpoints/data/building/865f58ae> ;
rdf:_4  <http://ns.ox.ac.uk/namespaces/oxpoints/data/building/25212490>
] ;
```

3.3 The Gaboto configuration file

The configuration file contains general information (such as database connection etc.), and the definition of all the entities in the system, that you want to store information about. To define an entity we have to specify what properties (v.s.) the entity can have. The configuration file also allows us to define GabotoBeans that can then be used as complex properties. The syntax for the configuration file is documented with an XSD schema

In the case of the OxPoints system most of our entities fall into two super classes: Units and Places. To be able to add properties to all entities we also introduced a common abstract super entity called **OxpEntity**. Units describe political units, such as colleges, departments, libraries etc. Places describe anything that can be mapped (e.g. buildings, rooms, etc.). All these types (supertypes and subtypes) will become an GabotoEntity. As for complex properties, the Gaboto system currently models a very simple address (consisting of a postal code and a street address) and a location to store latitude and longitude.

Before we go into detail, let's have a look at the configuration file, the Gaboto system uses:

```
<Gaboto  xml:base="../config/Gaboto.xml"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <config>
    <namespaces
        graphs="http://ns.ox.ac.uk/namespaces/oxpoints/graphs#"
        data="http://ns.ox.ac.uk/namespaces/oxpoints/data/"/>
    <namespacePrefixes>
      <namespacePrefix
```

```
        prefix="oxp"
        ns="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#"/>
  </namespacePrefixes>
  <database>
    <engineName>MySQL</engineName>
    <url>jdbc:mysql://localhost:8889/oxp</url>
    <user>gabotoUser</user>
    <password>gabotoPassword</password>
    <driver>com.mysql.jdbc.Driver</driver>
  </database>
</config>
<GabotoBeans>
  <GabotoBean
      name="Address"
      type="http://nwalsh.com/rdf/vCard#Address">
    <properties>
      <property
          name="postCode"
          type="String"
          uri="http://nwalsh.com/rdf/vCard#postal_code"/>
      <property
          name="streetAddress"
          type="String"
          uri="http://nwalsh.com/rdf/vCard#street_address"/>
    </properties>
  </GabotoBean>
  <GabotoBean  name="Location"  type="http://www.opengis.net/gml/Point">
    <properties>
      <property  name="pos"  type="String"  uri="http://www.opengis.net/gml/pos"/>
    </properties>
    <customMethods>
      <method>

              public Double getLatitude(){
              try{
              return Double.valueOf(getPos().split(" ")[1]);
              }catch(NumberFormatException e){
              return null;
              }
              }

      </method>
      <method>

              public Double getLongitude(){
              try{
              return Double.valueOf(getPos().split(" ")[0]);
              }catch(NumberFormatException e){
              return null;
              }
              }

      </method>
    </customMethods>
  </GabotoBean>
</GabotoBeans>
<GabotoEntities>
```

```
<!-- Common bas class -->
    <GabotoEntity
        name="OxpEntity"
        type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#OxpEntity"
        abstract="true">
      <properties>
        <property
            name="images"
            type="Image"
            collection="bag"
            uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#inImage"/>
      </properties>
      <GabotoEntities>
        <GabotoEntity
            name="Image"
            type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Image">
          <properties>
            <property
                name="width"
                type="String"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#width"/>
            <property
                name="height"
                type="String"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#height"/>
            <passiveProperty
                name="imageContents"
                type="OxpEntity"
                relationshipType="N:M"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#inImage"/>
          </properties>
        </GabotoEntity>
        <GabotoEntity
            name="Place"
            type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Place">
          <properties>
            <property
                name="name"
                type="String"
                uri="http://purl.org/dc/elements/1.1/title"/>
            <property
                name="parent"
                type="Place"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContainedWithin">
              <indirectProperty
                  uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"/>
              <unstoredProperty
                  uri="http://ns.ox.ac.uk/namespace/gaboto/kml/2009/03/owl#parent"/>
            </property>
            <property
                name="location"
                type="Location"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"/>
            <property
                name="address"
                type="Address"
                uri="http://nwalsh.com/rdf/vCard#adr"/>
            <property
                name="homepage"
                type="Website"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasHomepage"/>
            <passiveProperty
                name="containedPlaces"
                type="Place"
                relationshipType="1:N"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContainedWithin"/>
          </properties>
          <GabotoEntities>
            <GabotoEntity
                name="Building"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Building">
              <properties>
                <passiveProperty
                    name="occupiedBy"
                    type="Unit"
                    relationshipType="N:M"
                    uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#occupies"/>
              </properties>
            </GabotoEntity>
            <GabotoEntity
                name="Carpark"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Carpark">
              <properties>
                <property
                    name="capacity"
                    type="int"
                    uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#capacity"/>
              </properties>
            </GabotoEntity>
            <GabotoEntity
                name="DrainCover"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#DrainCover"/>
            <GabotoEntity
                name="Entrance"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Entrance"/>
            <GabotoEntity
                name="Room"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Room"/>
            <GabotoEntity
                name="Site"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Site"/>
            <GabotoEntity
                name="WAP"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#WAP"/>
          </GabotoEntities>
        </GabotoEntity>
        <GabotoEntity
            name="Unit"
            type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Unit">
          <properties>
            <property
                name="address"
                type="Address"
                uri="http://nwalsh.com/rdf/vCard#adr"/>
            <property
                name="homepage"
                type="Website"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasHomepage"/>
            <property
                name="OUCSCode"
                type="String"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasOUCSCode"/>
            <property
                name="itHomepage"
                type="Website"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasITHomepage"/>
            <property
                name="name"
                type="String"
                uri="http://purl.org/dc/elements/1.1/title"/>
            <property
                name="occupiedBuildings"
                type="Building"
                collection="bag"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#occupies">
              <indirectProperty
                  uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"
                  n="2"/>
            </property>
            <property
                name="primaryPlace"
                type="Place"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#primaryPlace">
              <indirectProperty
                  name="location"
                  uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"
                  n="1"/>
            </property>
            <property
                name="subsetOf"
                type="Unit"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#subsetOf">
              <indirectProperty
                  uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"
                  n="3"/>
              <unstoredProperty
                  uri="http://ns.ox.ac.uk/namespace/gaboto/kml/2009/03/owl#parent"/>
            </property>
            <property
                name="weblearn"
                type="Website"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasWeblearn"/>
            <passiveProperty
                name="hasSubsets"
                type="Unit"
                relationshipType="1:N"
                uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#subsetOf"/>
          </properties>
          <customMethods>
            <method>
                    @StaticProperty("http://purl.org/dc/elements/1.1/description")
```

```
                        public String getDescription(){
                        String description = "";

                        if(null != getHomepage())
                         description += "Website:  <a href=\"" + getHomepage().getUri() + "\">" +
getHomepage().getUri() + "</a><br/>";
                        if(null != getImages() && getImages().size() > 0){
                        Image img = getImages().iterator().next();
                        description += "<img src=\"" + img.getUri() + "\" width=\"" + img.getWidth()
+ "\" height=\"" + img.getHeight() + "\"/>";
                        }

                        return description;
                        }

            </method>
          </customMethods>
          <GabotoEntities>
            <GabotoEntity
                name="College"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#College"/>
            <GabotoEntity
                name="Department"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Department"/>
            <GabotoEntity
                name="Division"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Division"/>
            <GabotoEntity
                name="Faculty"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Faculty"/>
            <GabotoEntity
                name="Group"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Group"/>
            <GabotoEntity
                name="Library"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library">
              <properties>
                <property
                    name="OLISCode"
                    type="String"
                    uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasOLISCode"/>
                <property
                    name="libraryHomepage"
                    type="Website"
                    uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLibraryHomepage"/>
              </properties>
            </GabotoEntity>
            <GabotoEntity
                name="Museum"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Museum"/>
            <GabotoEntity
                name="ServiceDepartment"
                type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#ServiceDepartment"/>
          </GabotoEntities>
        </GabotoEntity>
        <GabotoEntity
            name="Website"
```

```
              type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Website">
            <properties>
              <passiveProperty
                  name="isWeblearnIn"
                  type="OxpEntity"
                  relationshipType="1:N"
                  uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasWeblearn"/>
              <passiveProperty
                  name="isITHomepageIn"
                  type="OxpEntity"
                  relationshipType="1:N"
                  uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasITHomepage"/>
              <passiveProperty
                  name="isHomepageIn"
                  type="OxpEntity"
                  relationshipType="1:N"
                  uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasHomepage"/>
              <passiveProperty
                  name="isLibraryHomepageIn"
                  type="OxpEntity"
                  relationshipType="1:N"
                  uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLibraryHomepage"/>
            </properties>
          </GabotoEntity>
        </GabotoEntities>
      </GabotoEntity>
    </GabotoEntities>
</Gaboto>
```

3.3.1 General Structure

The configuration file is divided into three sections: A general configuration, One to define GabotoBeans (complex properties) and one to define the entities.

```
<Gaboto>
  <config/>
  <GabotoBeans/>
  <GabotoEntities/>
</Gaboto>
```

3.3.2 The config Section

The config section defines the database connection to be used, the namespaces under which data and graphs are stored and additional namespace prefixes, that will be added to the default prefixes returned by *GabotoPredefinedQueries.getStandardPrefixes*.

3.3.2.1 Defining Database Connection

Gaboto uses Jena, for its database connection (section on persistent models in Jena). It comes shipped with the necessary JDBC libraries for MySQL and PostgreSQL.

Example configuration for MySQL:

```
<database>
  <engineName>MySQL</engineName>
  <url>jdbc:mysql://localhost:8889/oxp</url>
  <user>gabotoUser</user>
  <password>gabotoPassword</password>
  <driver>com.mysql.jdbc.Driver</driver>
</database>
```

Example configuration for PostgreSQL:

```
<database>
  <engineName>PostgreSQL</engineName>
  <url>jdbc:postgresql://localhost:5432/oxpoints?ssl=true</url>
  <user>gabotoUser</user>
  <password>gabotoPassword</password>
  <driver>org.postgresql.Driver</driver>
</database>
```

3.3.3 Defining GabotoBeans

Let's have a look at the simple **Address** bean:

```
<GabotoBean
    name="Address"
    type="http://nwalsh.com/rdf/vCard#Address">
  <properties>
    <property
        name="postCode"
        type="String"
        uri="http://nwalsh.com/rdf/vCard#postal_code"/>
    <property
        name="streetAddress"
        type="String"
        uri="http://nwalsh.com/rdf/vCard#street_address"/>
  </properties>
</GabotoBean>
```

The `<GabotoBean>` element has two mandatory attributes:

name The name attribute, describes the bean's name. It should start with an uppercase letter and follow the rules for Java class names, as it will be used as the class name for the corresponding Java class.

type The type attribute defines the type (RDF:type) of the bean. It has to be a URI.

Next, we have to define the properties our bean is going to have. We do this in the `<properties>` section. Each `<property>` element has three mandatory attributes and an optional collection attribute:

name (mandatory) The property's name is used to derive the method names for the resulting class. **postCode** would produce the names: **getPostCode** and **setPostCode**.

type (mandatory) Defines the property's datatype. The datatype can either be **String** (mapped to an RDF Literal), the name of an GabotoBean or the name of an GabotoEntity.

uri (mandatory) Defines the predicate URI this property should be matched to. This has to be a valid URI.

collection Defines whether the property stores exactly one value, or whether it stores a list of values (mapped to RDF Collections). Supported values are:

- **bag**: an unordered list.

3.3.3.1 The bean's RDF

As we have said, GabotoBeans (or complex properties) are properties that are connected to an GabotoEntity in the underlying RDF via a blank node. Here is an example of the RDF produced by the Address bean.

```
<http://ns.ox.ac.uk/namespaces/Gaboto/data/unit/c40051d6>
v:adr [ a       v:Address ;
v:postal_code "OX2 6LY" ;
v:street_address "1 Church Walk, Oxford"
] ;
```

3.3.3.2 The Resulting Class

From a `<GabotoBean>` element, Gaboto creates a custom Java class. Here is the result for the Address bean class.

```
public class Address extends GabotoBean {
private String postCode;
private String streetAddress;
@Override
public String getType(){
return "http://nwalsh.com/rdf/vCard#Address";
}
@SimpleLiteralProperty("http://nwalsh.com/rdf/vCard#postal_code")
public String getPostCode(){
return this.postCode;
}
@SimpleLiteralProperty("http://nwalsh.com/rdf/vCard#postal_code")
public void setPostCode(String postCode){
this.postCode = postCode;
}
@SimpleLiteralProperty("http://nwalsh.com/rdf/vCard#street_address")
public String getStreetAddress(){
return this.streetAddress;
}
@SimpleLiteralProperty("http://nwalsh.com/rdf/vCard#street_address")
public void setStreetAddress(String streetAddress){
this.streetAddress = streetAddress;
}
public void loadFromResource(Resource res, GabotoSnapshot snapshot, GabotoEntityPool pool) {
super.loadFromResource(res, snapshot, pool);
Statement stmt;
stmt = res.getProperty(snapshot.getProperty("http://nwalsh.com/rdf/vCard#postal_code"));
if(null != stmt && stmt.getObject().isLiteral())
this.setPostCode(((Literal)stmt.getObject()).getString());
stmt = res.getProperty(snapshot.getProperty("http://nwalsh.com/rdf/vCard#street_address"));
if(null != stmt && stmt.getObject().isLiteral())
this.setStreetAddress(((Literal)stmt.getObject()).getString());
}
}
```

We will have a look at the meaning of the annotations and the **loadFromResource** method in *4. Part III - Extending Gaboto*.

3.3.3.3 Custom Methods

In some cases, you might want to add custom helper methods to a bean class. For this purpose you can use the `<customMethods>` element (at the same level as `<properties>`) containing `<method>` elements. `<method>` can then contain plain Java code, that is added to the class.

3.3.4 Defining GabotoEntities

The configuration for GabotoEntities is similar to GabotoBeans but as you might guess a bit more complex. Let's start with a simple entity: **Image**

```
<GabotoEntity
    name="Image"
    type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Image">
  <properties>
    <property
        name="width"
        type="String"
        uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#width"/>
    <property
        name="height"
        type="String"
        uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#height"/>
    <property
        name="imageOf"
        type="GabotoEntity"
        collection="bag"
        uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#imageOf"/>
  </properties>
</GabotoEntity>
```

An GabotoEntity is defined using the `<GabotoEntity>` element. This element has two mandatory and one optional attributes:

name (mandatory) The name attribute, describes the entity's name. It should start with an uppercase letter and following the rules for Java class names, as it will be used as the class name for the corresponding Java class.

type (mandatory) The type attribute defines the type (RDF:type) of the entity. It has to be a URI.

abstract If set to *true*, then the resulting class will be labeled as abstract.

Each `<property>` element has three mandatory attributes and an optional collection attribute:

name (mandatory) The property's name is used to derive the method names for the resulting class. **imageOf** would produce the names: **getImageOf**, **setImageOf** and **addImageOf**[5].

type (mandatory) Defines the property's datatype. The datatype can either be **String** (mapped to an RDF Literal), the name of an GabotoBean or the name of an GabotoEntity.

uri (mandatory) Defines the predicate URI this property should be matched to. This has to be a valid URI.

collection Defines whether the property stores exactly one value, or whether it stores a list of values (mapped to RDF Collections). Supported values are:

- **bag**: an unordered list.

So far not much new. Let's have a look at a more complex entity:

```
<GabotoEntity
    name="Place"
    type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Place">
  <properties>
    <property
        name="name"
        type="String"
        uri="http://purl.org/dc/elements/1.1/title"/>
    <property
        name="parent"
        type="Place"
        uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContainedWithin">
      <indirectProperty
          uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"/>
    </property>
    <property
        name="location"
        type="Location"
        uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"/>
    <property
        name="address"
        type="Address"
        uri="http://nwalsh.com/rdf/vCard#adr"/>
    <property
        name="homepage"
        type="Website"
```

[5] The add method is added whenever you have a collection property.

```
        uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasHomepage"/>
    <passiveProperty
        name="containedPlaces"
        type="Place"
        relationshipType="1:N"
        uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContainedWithin"/>
  </properties>
  <GabotoEntities>
    <GabotoEntity
        name="Building"
        type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Building">
      <properties>
        <passiveProperty
            name="occupiedBy"
            type="Unit"
            relationshipType="N:M"
            uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#occupies"/>
      </properties>
    </GabotoEntity>
    <GabotoEntity
        name="Carpark"
        type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Carpark"/>
    <GabotoEntity
        name="DrainCover"
        type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#DrainCover"/>
    <GabotoEntity
        name="Entrance"
        type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Entrance"/>
    <GabotoEntity
        name="Room"
        type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Room"/>
    <GabotoEntity
        name="Site"
        type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Site"/>
    <GabotoEntity
        name="WAP"
        type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#WAP"/>
  </GabotoEntities>
</GabotoEntity>
```

3.3.4.1 Inheritance

`<GabotoEntity>` can contain a `<GabotoEntities>` element. With this it is possible to create an inheritance structure which is then mapped to Java. In the case of Place the entities Building, Carpark, DrainCover, Entrance, Room, Site and WAP are sub types and therefore inherit all the properties defined on Place.

3.3.4.2 Indirect Properties

Each `<property>` element can have multiple `<indirectProperty>` child elements. An indirect property serves two objectives:

1. In case you do not have a direct property defined (active, static and unstored) for this URI, then Gaboto will search for that property in the entity returned by the outer property definition.

2. In case you have a direct property defined for this URI, Gaboto uses the indirect property as a fallback solution in case the return value from the direct property is null.

Let us have a look at the example. In the example, the property (with the uri **http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContainedWithin**) has an indirect property definition for the property **http://ns.ox.ac.uk/namespace/oxpoints/2009/02/ow** This means, that if you ask an object of type Place for the property **hasLocation** it will since it has a direct property defined for it, first check the direct property and return its value if it is not null. If it is null, it will take the result of the property **physicallyContainedWithin** (which returns an GabotoEntity) and search for the **hasLocation** property there.

Another example (now case 2) would be the location of a Unit. A Unit does not have a direct property for **hasLocation**. But we have defined that the location of a Unit is the location of its primary building (described by the property **hasPrimaryBuilding**).

Since indirect properties try to search for a property at a different entity, they can naturally only be defined on properties, that describe a relationship with another entity (store another entity).

3.3.4.2.1 Multiple Alternatives

In some cases you might want to specify more than one ”alternative” using indirect properties. For example for Units, we might want to say, that the location should be searched for in its primary building, if this fails, then go to the super Unit and search for the location there. We can do this by adding another attribute **n** on the `<indirectProperty>` elements which takes a natural number (starting with 1) and defines the order in which the indirect properties should be processed.[6]

```
<GabotoEntity
    name="Unit"
    type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Unit">
  <properties> ...
  <property
        name="primaryPlace"
        type="Place"
        uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#primaryPlace">
      <indirectProperty
          uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"
          n="1"/>
    </property>
    <property
        name="subsetOf"
        type="Unit"
        uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#subsetOf">
      <indirectProperty
          uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"
          n="2"/>
    </property>
      ...
  </properties>
...

</GabotoEntity>
```

[6]If **n** is not defined, the default **1** will be assumed.

3.3.4.2.2 Method Creation

By default no methods are created for indirect properties. If you want Gaboto to create a get method for you, you have to add a name attribute onto the `<indirectProperty>` element. Note: Currently methods to look up indirect properties are not type safe. This means that they will return an object of type **Object** and you have to manually cast it.

3.3.4.3 Passive Properties

If an entity has a relationship with another entity, but the information about this relationship is stored with the other entity, we can add a `<passiveProperty>` element to tell Gaboto to include methods, that automatically dereference the properties and allow for easy access.

Places, for example, have the property **physicallyContainedWithin**. So each place directly knows its parent. If we now wanted to be able to tell, what places a place contains, we had to dereference the property and search for it "backwards". Adding a **passive property** will do all that and we can access the contained places like any other property.

```
<passiveProperty
    name="containedPlaces"
    type="Place"
    relationshipType="1:N"
    uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContainedWithin"/>
```

The `<passiveProperty>` element has four mandatory attributes:

name The property's name. The name is used to derive the method names. In the example a method **getContainedPlaces** would be created.

type The type of the GabotoEntity where the actual property is stored.

relationshipType Defines the cardinality of the relationship. The first part describes the number of entities that take part in the relationship, of the type where the passive property is defined. The second part describes the number of entities that take part in the relationship, of the type where the actual property is defined. Allowed values are:

- **1:N** Example: One place contains many places, whereas each place is only directly contained in one other place.
- **N:M** Example: A unit occupies many buildings and each building may be occupied by multiple units.

uri The URI, that describes the property.

3.3.4.3.1 Set- and Add- Methods

Even though passive properties will create set (and sometimes add) methods, these should not be used. They are only used when loading the data from RDF and whatever is stored in a passive property will have no effect, when storing the entity.

3.3.4.4 Custom Methods

As is the case with GabotoBeans, you might want to add custom helper methods to an entity class. For this purpose you can use the `<customMethods>` element (at the same level as `<properties>`) containing `<method>` elements. `<method>` can then contain plain Java code, that is added to the class.

3.3.4.5 Static Properties

You might have noticed, that most of the methods that are automatically generated are annotated using Java Annotations. Using reflection Gaboto takes these annotations to figure out how to create RDF for the various properties[7]. More importantly however, the annotations are used by the general output transformers (such as KML, JSON, etc.).

In order to generate output for any kind of GabotoEntity, the general output transformers treat each entity not as a specific entity (such as college, building etc.) but as an GabotoEntity, that has certain properties. The KML transformer, for example, simply checks for the following properties:

http://purl.org/dc/elements/1.1/title The dublin core title property.

http://purl.org/dc/elements/1.1/description The dublin core description property, which is used as an optional description. This can contain HTML.

http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation The Gaboto location attribute, used to define the entity's location.

In some cases, as, for example, with the **http://purl.org/dc/elements/1.1/description** property, you might not want to actually store this property in the database, but be able to generate it "on the fly" by combining various bits of information. For this purpose you can create **custom methods** and annotate these with the **StaticProperty** annotation. The single parameter to the annotation is the property's URI. The method should have no parameters, and the return value should (naturally) correspond to the definition of the property (in the case of http://purl.org/dc/elements/1.1/description the return value should be a string).

```
<GabotoEntity
    name="Unit"
    type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Unit">
  <properties> ...
  </properties>
  <customMethods>
    <method>
         @StaticProperty("http://purl.org/dc/elements/1.1/description")
         public String getDescription(){
         String description = "";

         if(null != getHomepage())
           description   +=   "Website:    <a   href=\""   +   getHomepage().getUri()   +   "\">"   +
getHomepage().getUri() + "</a><br/>";
         if(null != getImages() && getImages().size() > 0){
         Image img = getImages().iterator().next();
         description += "<img src=\"" + img.getUri() + "\" width=\"" + img.getWidth() + "\" height=\""
+ img.getHeight() + "\"/>";
         }

         return description;
         }

    </method>
  </customMethods>
```

[7]This could be in fact changed to also create the appropriate methods for RDF generation during the creation of the class. However, since this is (at least at the moment) not a time critical task, reflection is more than adequate for the job

```
  <GabotoEntities> ...
  </GabotoEntities>
</GabotoEntity>
```

3.3.4.6 Unstored Properties

Unstored properties are properties that have the same value as a stored property, but that are not persisted. They are mainly used by tools (e.g. transformation tools) that work with GabotoEntities in a generic way. For example the KML transformation looks for a property called *http://ns.ox.ac.uk/namespace/gaboto/kml/2009/03/owl#parent* when it constructs the title of a placemark. If it finds the *parent* attribute it will add the parent's title to the title to create a comma separated list of titles:

```
<kml xmlns="http://www.opengis.net/kml/2.2"
>
  <Document xmlns="http://www.opengis.net/kml/2.2"
  >
    <Placemark xmlns="http://www.opengis.net/kml/2.2"
    >
      <name xmlns="http://www.opengis.net/kml/2.2"
      >All Souls College Library, All Souls College</name>
      <description xmlns="http://www.opengis.net/kml/2.2"
      >Website: <a href="http://www.all-souls.ox.ac.uk/library/">http://www.all-souls.ox.ac.uk/library/</a><br/>
      <Point xmlns="http://www.opengis.net/kml/2.2"
      >
        <coordinates xmlns="http://www.opengis.net/kml/2.2"
        >-1.253042221069336,51.75278555467572</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

Usually you have such properties already defined in a more specific ontology. Here is the example of OxPoints Unit:

```
<GabotoEntity
    name="Unit"
    type="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Unit">
  <properties> ...
  <property
        name="subsetOf"
        type="Unit"
        uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#subsetOf">
      <indirectProperty
          uri="http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"
          n="3"/>
      <unstoredProperty
          uri="http://ns.ox.ac.uk/namespace/gaboto/kml/2009/03/owl#parent"/>
    </property>
     ...
  </properties>
...
<GabotoEntities> ...
```

```
  </GabotoEntities>
</GabotoEntity>
```

You can add as many unstored properties into a property element as you like. If you add a name attribute getMethods will be created.

3.3.4.7 The resulting Java Class for Place

```
package org.oucs.gaboto.entities;
import java.util.HashMap;
import java.util.Map;
import java.util.Collection;
import java.util.HashSet;
import java.util.Set;
import java.util.List;
import java.util.ArrayList;
import java.lang.reflect.Method;
import org.oucs.gaboto.entities.utils.SimpleLiteralProperty;
import org.oucs.gaboto.entities.utils.SimpleURIProperty;
import org.oucs.gaboto.entities.utils.ComplexProperty;
import org.oucs.gaboto.entities.utils.BagURIProperty;
import org.oucs.gaboto.entities.utils.BagLiteralProperty;
import org.oucs.gaboto.entities.utils.BagComplexProperty;
import org.oucs.gaboto.entities.utils.IndirectProperty;
import org.oucs.gaboto.entities.utils.UnstoredProperty;
import org.oucs.gaboto.entities.utils.PassiveProperty;
import org.oucs.gaboto.entities.utils.StaticProperty;
import org.oucs.gaboto.vocabulary.*;
import org.oucs.gaboto.entities.GabotoEntity;
import org.oucs.gaboto.entities.pool.GabotoEntityPool;
import org.oucs.gaboto.entities.pool.EntityExistsCallback;
import org.oucs.gaboto.entities.pool.PassiveEntitiesRequest;
import org.oucs.gaboto.model.GabotoSnapshot;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.Statement;
import com.hp.hpl.jena.rdf.model.StmtIterator;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.Literal;
import com.hp.hpl.jena.rdf.model.Bag;
import com.hp.hpl.jena.rdf.model.NodeIterator;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.ontology.OntClass;
import org.oucs.gaboto.entities.OxpEntity;
import org.oucs.gaboto.beans.Location;
import org.oucs.gaboto.beans.Address;
```

```
/**
*<p>This class was automatically generated by Gaboto<p>
*/
public class Place extends OxpEntity {
private String name;
private Place parent;
private Location location;
private Address address;
private Website homepage;
private Collection<Place> containedPlaces;
private static Map<String, List<Method>> indirectPropertyLookupTable;
static{
indirectPropertyLookupTable = new HashMap<String, List<Method>>();
List<Method> list;
try{
list = new ArrayList<Method>();
list.add(Place.class.getMethod("getParent", (Class<?>[])null));
indirectPropertyLookupTable.put("http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation",
list);
} catch (Exception e) {
e.printStackTrace();
}
}
@Override
public String getType(){
return "http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Place";
}
@SimpleLiteralProperty(
value = "http://purl.org/dc/elements/1.1/title",
datatypeType = "javaprimitive",
javaType = "String"
)
public String getName(){
return this.name;
}
@SimpleLiteralProperty(
value = "http://purl.org/dc/elements/1.1/title",
datatypeType = "javaprimitive",
javaType = "String"
)
public void setName(String name){
this.name = name;
}
@UnstoredProperty({"http://ns.ox.ac.uk/namespace/gaboto/kml/2009/03/owl#parent"})
@IndirectProperty({"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"})
```

```
@SimpleURIProperty("http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContainedWithin")
public Place getParent(){
if(! this.isDirectReferencesResolved())
this.resolveDirectReferences();
return this.parent;
}
@SimpleURIProperty("http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContainedWithin")
public void setParent(Place parent){
if( null != parent)
this.removeMissingReference( parent.getUri() );
this.parent = parent;
}
@ComplexProperty("http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation")
public Location getLocation(){
return this.location;
}
@ComplexProperty("http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation")
public void setLocation(Location location){
this.location = location;
}
@ComplexProperty("http://nwalsh.com/rdf/vCard#adr")
public Address getAddress(){
return this.address;
}
@ComplexProperty("http://nwalsh.com/rdf/vCard#adr")
public void setAddress(Address address){
this.address = address;
}
@SimpleURIProperty("http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasHomepage")
public Website getHomepage(){
if(! this.isDirectReferencesResolved())
this.resolveDirectReferences();
return this.homepage;
}
@SimpleURIProperty("http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasHomepage")
public void setHomepage(Website homepage){
if( null != homepage)
this.removeMissingReference( homepage.getUri() );
this.homepage = homepage;
}
@PassiveProperty(
uri = "http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContainedWithin",
entity = "Place"
)
public Collection<Place> getContainedPlaces(){
```

```
if(! isPassiveEntitiesLoaded() )
loadPassiveEntities();
return this.containedPlaces;
}
@PassiveProperty(
uri = "http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContainedWithin",
entity = "Place"
)
private void setContainedPlaces(Collection<Place> containedPlaces){
this.containedPlaces = containedPlaces;
}
private void addContainedPlace(Place containedPlace){
if(null == this.containedPlaces)
this.containedPlaces = new HashSet<Place>();
this.containedPlaces.add(containedPlace);
}
public Collection<PassiveEntitiesRequest> getPassiveEntitiesRequest(){
Collection<PassiveEntitiesRequest> requests = super.getPassiveEntitiesRequest();
if(null == requests)
requests = new HashSet<PassiveEntitiesRequest>();
requests.add(new PassiveEntitiesRequest(){
public String getType() {
return "http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Place";
}
public String getUri() {
return "http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContainedWithin";
}
public int getCollectionType() {
return GabotoEntityPool.PASSIVE_PROPERTY_COLLECTION_TYPE_NONE;
}
public void passiveEntityLoaded(GabotoEntity entity) {
addContainedPlace((Place)entity);
}
});
return requests;
}
public void loadFromSnapshot(Resource res, GabotoSnapshot snapshot, GabotoEntityPool pool) {
super.loadFromSnapshot(res, snapshot, pool);
Statement stmt;
stmt = res.getProperty(snapshot.getProperty("http://purl.org/dc/elements/1.1/title"));
if(null != stmt && stmt.getObject().isLiteral())
this.setName(((Literal)stmt.getObject()).getString());
stmt = res.getProperty(snapshot.getProperty("http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#physicallyContaine
if(null != stmt && stmt.getObject().isResource()){
Resource missingReference = (Resource)stmt.getObject();
EntityExistsCallback callback = new EntityExistsCallback(){
public void entityExists(GabotoEntityPool pool, GabotoEntity entity) {
setParent((Place)entity);
}
};
this.addMissingReference(missingReference, callback);
}
stmt = res.getProperty(snapshot.getProperty("http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation"));
if(null != stmt && stmt.getObject().isAnon()){
Location location = new Location();
location.loadFromResource((Resource)stmt.getObject(), snapshot, pool);
setLocation(location);
}
stmt = res.getProperty(snapshot.getProperty("http://nwalsh.com/rdf/vCard#adr"));
if(null != stmt && stmt.getObject().isAnon()){
Address address = new Address();
address.loadFromResource((Resource)stmt.getObject(), snapshot, pool);
setAddress(address);
}
stmt = res.getProperty(snapshot.getProperty("http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasHomepage"));
if(null != stmt && stmt.getObject().isResource()){
Resource missingReference = (Resource)stmt.getObject();
EntityExistsCallback callback = new EntityExistsCallback(){
public void entityExists(GabotoEntityPool pool, GabotoEntity entity) {
setHomepage((Website)entity);
}
};
this.addMissingReference(missingReference, callback);
}
}
public List<Method> getIndirectMethodsForProperty(String propertyURI){
List<Method> list = super.getIndirectMethodsForProperty(propertyURI);
if(null == list)
return indirectPropertyLookupTable.get(propertyURI);
else{
List<Method> tmp = indirectPropertyLookupTable.get(propertyURI);
if(null != tmp)
list.addAll(tmp);
}
return list;
}
}
```

3.4 Generating the Java Files from the Configuration

3.4.1 From the Command Line

To generate the Java classes from the configuration file run:

```
org.oucs.gaboto.helperscripts.CreateClassesFromConfiguration        configfile    outputdirEntities
outputdirBeans outputdirMisc
```

You should then copy all the beans to org.oucs.gaboto.beans, all the entities to org.oucs.gaboto.entities and all the misc files to org.oucs.gaboto.util and recompile the Gaboto Library.

3.4.2 From within Java

The GabotoLibrary offers a method to trigger the generation of classes.

3.4.3 Using Apache Ant

See *4.1.3. Compiling Gaboto*.

Chapter 4

Part III - Extending Gaboto

4.1 Working with the Gaboto Code

This section describes tools that should be used when developing Gaboto and how to compile new versions.

4.1.1 Test Driven Development with JUnit

Most of Gaboto's more important objects were/are tested using unit tests (using JUnit). Currently Gaboto contains a test suit with 50+ test cases (in package: org.oucs.gaboto.test). Whenever any changes to Gaboto are made, these tests should still run successfully (unless you have a very good explanation) [1].

As some of the tests add and remove triples the tests should not be run against a productive system.

4.1.2 Transforming Ontologies into Java Helper Classes

Typing in long URIs is hazardous and error prone. Jena therefore offers a small utility to create Java helper classes for OWL and RDFS ontologies called schemagen. The utility can be run from the command line, or via an ant task. Here is an example ant task to transform the vCard ontology. For more examples have a look at the ant build file **build.xml**.

```
<java  classname="jena.schemagen"  classpathref="eclp"  fork="yes">
  <arg  value="-i"/>
  <arg  value="ontologies/vCard.owl"/>
  <arg  value="-n"/>
  <arg  value="VCard"/>
  <arg  value="-o"/>
  <arg  value="${vocab.dir}"/>
  <arg  value="--ontology"/>
  <arg  value="--package"/>
  <arg  value="org.oucs.gaboto.vocabulary"/>
</java>
```

To run the schema generation call:

[1] The only test that is allowed to fail is **testModelSizeEquality**. However, the difference between the expected and the actual value shouldn't be too high (java.lang.AssertionError: expected:14616 but was:14615, would be ok).

```
ant schemas
```

4.1.3 Compiling Gaboto

Gaboto can be compiled using the provided ant build script. Available ant targets are:

compile Compiles gaboto.

debploy This is the default ant task. It will compile Gaboto and copy a Gaboto.jar file and a Gaboto.zip (containing documentation, Gaboto.jar and all necessary libraries) to the deploy directory.

gabotoConfiguration Reads the configuration and copies the created classes to the correct positions.

configureAndDeploy Creates the necessary classes from the configuration, compiles and deploys Gaboto.

schemas Transforms defined ontologies into Java helper classes (see *4.1.2. Transforming Ontologies into Java Helper Classes*).

4.2 Output Transformation

Gaboto allows for the dynamic transformation of GabotoEntityPools into specified output formats. These transformations are independent from the entities inside the pool. In this section we'll have a look at existing transformers, their implementations and how to create new transformers.

4.2.1 General Approach

Basically, there are two approaches to writing a transformer. The first approach is to transform entire entities including all their properties into an output format that can then be processed by another system. The **JSON transformer** does exactly that and transforms entire entities into a JSON representation that can then be easily processed from within JavaScript or in any other system that allows to import JSON.

The **KML transformer** follows a different approach. It looks for few very specific properties that it needs to create valid KML documents and thereby ignores most of an entity's properties. It also adds specific KML transformation properties, that an entity might ”implement” to guide the KML transformation process.

4.2.2 The KML Transformer

In its simplest form the KML transformer looks for three properties:

- http://purl.org/dc/elements/1.1/title
- http://purl.org/dc/elements/1.1/description
- http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation

It uses the generic methods provided by the GabotoEntity object (*getPropertyValue*) to see whether the entity that it currently transforms has these properties or not.

A simple transformation of two entities from OxPoints could produce this KML:

```
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<Placemark>
<name>All Souls College</name>
<description><![CDATA[Website: <a href="http://www.all-souls.ox.ac.uk">http://www.all-souls.ox.ac.uk</a><br/><im
src="http://www.oucs.ox.ac.uk/oxpoints/images/collegepictures/alls.jpg"             width="167px"
height="125px"/>]]></description>
<Point>
<coordinates>-1.253042221069336,51.75278555467572</coordinates>
</Point>
</Placemark>
<Placemark>
<name>Computing Services</name>
<description><![CDATA[Website: <a href="http://www.oucs.ox.ac.uk/">http://www.oucs.ox.ac.uk/</a><br/>]]></descri
<Point>
<coordinates>-1.26035,51.76001</coordinates>
</Point>
</Placemark>
</Document>
</kml>
```

Besides the properties mentioned above, the KML transformer also looks for a property specific to its KML transformation: **http://ns.ox.ac.uk/namespace/gaboto/kml/2009/03/owl#parent**. When constructing the name of a placemark the KML transformer tries to establish a hierarchy using the **parent** property in order to add the names of all direct ancestors to the KML `<name>` element.

Since this property is seldom something an entity actually stores under this name, it is usually modelled using an **unstored property** (see *3.2.6. Unstored Properties*). For example, the All Souls College Library is a subunit of All Souls College

```
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<Placemark>
<name>All Souls College Library, All Souls College</name>
<description><![CDATA[Website: <a href="http://www.all-souls.ox.ac.uk/library/">http://www.all-souls.ox.ac.uk/li
<Point>
<coordinates>-1.253042221069336,51.75278555467572</coordinates>
</Point>
</Placemark>
</Document>
</kml>
```

4.2.3 The JSON Transformer

The JSON transformer follows a very different approach. It does not care about specific properties an entity might have or might not have, but it transforms all properties an entity has. For this, it also uses the generic methods made available by the GabotoEntity object (*getAllDirectProperties*, *getAllPassiveProperties* and *getAllIndirectProperties*). All properties are transformed into JSON key value pairs. If the value is another entity, a sub object is created and depending on the configured nesting level all of its properties are added.

This is the JSON output for **All Souls College Library** with a nesting level of 1 (this means that the properties of directly referenced entities are not included in the output):

```
[{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/451091d3",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasOUCSCode":"O_lib_asc",
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#subsetOf":{
"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/eacfd81c",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#College",
"nestingToDeep":true
}
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasITHomepage":null,
"http://purl.org/dc/elements/1.1/description":"Website: <a href=\"http://www.all-souls.ox.ac.uk/library/\">http:
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasHomepage":{
"uri":"http://www.all-souls.ox.ac.uk/library/",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Website",
"nestingToDeep":true
},
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLibraryHomepage":{
"uri":"http://www.lib.ox.ac.uk/libraries/guides/ASC.html",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Website",
"nestingToDeep":true
},
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasWeblearn":null,
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#occupies":null,
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#primaryPlace":null,
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasOLISCode":"ASC",
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#inImage":null,
"http://ns.ox.ac.uk/namespace/gaboto/kml/2009/03/owl#parent":{
"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/eacfd81c",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#College",
"nestingToDeep":true
},
"http://purl.org/dc/elements/1.1/title":"All Souls College Library",
"http://nwalsh.com/rdf/vCard#adr":null,
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#hasLocation":{
"http://www.opengis.net/gml/pos":"-1.253042221069336 51.75278555467572"
},
"passiveProperties":{
"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#subsetOf":[
{ "uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/9152267c",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/ae209cfc",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/de0e1221",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/74c2f3f6",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/bb83b72f",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/dca87473",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/e64e46e5",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/95331e00",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/568a19b6",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/c4573969",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/3651f49e",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/0b80626d",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/5762f9fd",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/bd13f56e",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/2a770d05",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/139108dc",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true},
{"uri":"http://ns.ox.ac.uk/namespaces/oxpoints/data/unit/0c5d61a7",
"type":"http://ns.ox.ac.uk/namespace/oxpoints/2009/02/owl#Library",
"nestingToDeep":true}
]
}
}]
```

4.2.4 Adding new Output Transformations to Gaboto

New transformers should implement the **EntityPoolTransformer** interface.

4.3 Insights into Gaboto

In this section I want to explain some of the ideas behind Gaboto's implementation.

4.3.1 JavaBeans vs. GabotoEntity and GabotoBean

Java Beans are reusable software components that follow certain coding guidelines and allow introspection (accessing information about themselve at runtime). GabotoEntities and Beans follow the Java Bean coding guidelines, but introspection is handled differently.

As described in *3.2. Properties*, Gaboto knows of a range of different property types, that have to be treated differently when trying to retrieve or store values. For example an indirect property follows a path to a different GabotoEntity and searches for the property in question there. In order to differentiate between the different properties, Gaboto makes use of Java annotations.

Java annotations allow us to tag classes, methods and fields with defined annotations that can be accessed at runtime. This technique is called reflection. Through the java reflection API, we can now, for example, loop over the methods of an object, and test each method for present annotations.

4.3.1.1 Gaboto's Annotation

In GabotoEntities and GabotoBeans methods are annotated to describe the sort of property this method represents. GabotoBeans can only have direct properties.

4.3.1.1.1 Direct Properties

Direct properties are properties that are stored together with the entity/bean or that can be directly retrieved from the entity/bean.

Direct Stored Properties

Properties that are stored together with the entity/bean.

SimpleLiteralProperty

SimpleURIProperty

ComplexProperty

BagLiteralProperty

BagURIProperty

BagComplexProperty

Direct Unstored Properties

Properties that are not stored, but that can be retrieved directly from the entity/bean.

StaticProperty

UnstoredProperty

4.3.1.1.2 IndirectProperty

Indirect properties are properties that are not stored directly with an entity, but with an entity that can be referenced (not necessarily directly) from this entity.

4.3.1.1.3 PassiveProperty

4.4 To Do

Here are a list of ideas of how Gaboto might be further improved.

4.4.1 Multiple Inheritance

RDF supports multiple inheritance. To map this to java, Gaboto needs to create interfaces and interface implementations from the configuration file rather than just the implementations.

4.4.2 Get rid of Reflection when creating RDF from a GabotoPool/GabotoEntity

This would increase the speed of adding data to the system and of generating RDF output.

To achieve this, specialized methods (similar to *loadFromSnapshot*) need to be created during the configuration process. The *getTriplesFor* method of the GabotoEntity object has then to be changed to use the dynamically created method instead of the reflection object (*RDFContainerTripleGeneratorImpl*).

4.4.3 More simple Datatypes

As Jena offers the full range of XSD datatypes, Gaboto should support more than the currently supported primitive java types for literal properties.

4.4.4 Make GabotoQuery know about existing Transformations

Currently GabotoQueryImpl only knows about the already existing transformations. It should be possible to make it aware of newly available transformations.

4.4.5 Add new Import Mechanisms (e.g., JSON Importer)

JSON could easily be used to add data into the system. In order to achieve this a JSON to GabotoEntity converter had to be written.

4.4.6 Create option to leave out Time Dimension

In many projects the Time Dimension might not be necessary. It should be easy for these projects to get rid of the time dimension and not having to create the same snapshot all the time.

4.4.7 Remove Custom Methods from Config File

It would be nicer to be able to create custom methods for entities/beans directly in your IDE instead of in the XML configuration file. For this each entity/bean needs to consist of a super and a subclass. The super class is the one that is generated from the configuration, while the subclass is the one that is actually used by the user. Here the user can easily add custom methods.

4.4.8 Allow for specific Jena Database configuration

Jena offers a couple of configuration properties for persistent models (see `http://jena.sourceforge.net/DB/creating-db-models.html`). It would be nice if these could be configured from within the Gaboto configuration file.

Appendix A

Configuration File Schema

```
<xsd:schema  elementFormDefault="qualified"  xml:base="../schemas/Gaboto.xsd"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xsd:element  name="Gaboto">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element  ref="config"/>
        <xsd:element  ref="GabotoBeans"/>
        <xsd:element  ref="GabotoEntities"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element  name="config">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element  ref="namespaces"/>
        <xsd:element  ref="namespacePrefixes"/>
        <xsd:element  ref="database"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element  name="namespaces">
    <xsd:complexType>
      <xsd:attribute  name="data"  use="required"  type="xs:anyURI"/>
      <xsd:attribute  name="graphs"  use="required"  type="xs:anyURI"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element  name="namespacePrefixes">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element  ref="namespacePrefix"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element  name="namespacePrefix">
    <xsd:complexType>
      <xsd:attribute  name="ns"  use="required"  type="xs:anyURI"/>
      <xsd:attribute  name="prefix"  use="required"  type="xs:NCName"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element  name="database">
```

```
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="engineName"/>
        <xsd:element ref="url"/>
        <xsd:element ref="user"/>
        <xsd:element ref="password"/>
        <xsd:element ref="driver"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="engineName" type="xs:NCName"/>
  <xsd:element name="url" type="xs:anyURI"/>
  <xsd:element name="user" type="xs:NCName"/>
  <xsd:element name="password" type="xs:NCName"/>
  <xsd:element name="driver" type="xs:NCName"/>
  <xsd:element name="GabotoBeans">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" ref="GabotoBean"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="GabotoBean">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="properties"/>
        <xsd:element minOccurs="0" ref="customMethods"/>
      </xsd:sequence>
      <xsd:attribute name="name" use="required" type="xs:NCName"/>
      <xsd:attribute name="type" use="required" type="xs:anyURI"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="properties">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" ref="property"/>
        <xsd:element minOccurs="0" maxOccurs="unbounded" ref="passiveProperty"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="property">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element minOccurs="0" ref="indirectProperty"/>
        <xsd:element minOccurs="0" ref="unstoredProperty"/>
      </xsd:sequence>
      <xsd:attribute name="collection" type="xs:NCName"/>
      <xsd:attribute name="name" use="required" type="xs:NCName"/>
      <xsd:attribute name="type" use="required" type="xs:NCName"/>
      <xsd:attribute name="uri" use="required" type="xs:anyURI"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="indirectProperty">
    <xsd:complexType>
      <xsd:attribute name="n" type="xs:integer"/>
      <xsd:attribute name="uri" use="required" type="xs:anyURI"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="unstoredProperty">
    <xsd:complexType>
      <xsd:attribute name="uri" use="required" type="xs:anyURI"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="passiveProperty">
    <xsd:complexType>
      <xsd:attribute name="name" use="required" type="xs:NCName"/>
      <xsd:attribute name="relationshipType" use="required" type="xs:NMTOKEN"/>
      <xsd:attribute name="type" use="required" type="xs:NCName"/>
      <xsd:attribute name="uri" use="required" type="xs:anyURI"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="customMethods">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" ref="method"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="method" type="xs:string"/>
  <xsd:element name="GabotoEntities">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" ref="GabotoEntity"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="GabotoEntity">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="GabotoEntities"/>
        <xsd:element ref="customMethods"/>
        <xsd:element ref="properties"/>
      </xsd:choice>
      <xsd:attribute name="abstract" type="xs:boolean"/>
      <xsd:attribute name="name" use="required" type="xs:NCName"/>
      <xsd:attribute name="type" use="required" type="xs:anyURI"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```