TUHH
Technische Universität Hamburg-Harburg

STS
Institut für
Softwaresysteme

JAVA Embedding of a Domain Specific Language OWLlink

Project Work

submitted by
Evgeni Radev
Information and Media Technologies
TUHH

supervised by
Prof. Dr. rer.-nat. habil. Ralf Möller
Sebastian Wandelt
M. Sc. Miguel Garcia

Hamburg University of Technology (TUHH)
Software Systems Institute (STS)

Abstract

The embedding of a domain-specific language into JAVA is the goal of this paper. As a tool for the generation of the required application programming interfaces DSL2JDT shall be used. The DSL of interest is OWLlink, which provides an implementation-neutral mechanism for accessing OWL reasoner functionality. This ability should be leveraged to the user in order to query semantic reasoners via the JAVA programming language.

Declaration

I declare that:
this work has been prepared by myself,
all literal or content based quotations are clearly pointed out,
and no other sources or aids than the declared ones have been used.

Evgeni Radev

Hamburg, submitted on: September 29, 2009

Contents

List of Figures

Listings

Chapter 1

Introduction

Summary. *The reader is introduced to the task that this work is aiming at, i.e. the embedding of OWLlink into JAVA. A short preview about every chapter along with the key issues in each one of them is summarized. An outlook on the literature being used finishes this chapter off.*

1.1 General

The topic of this project work is the embedding of a domain specific language into a general programming language. The reason behind this is to offer the possibility of using a certain language instrument for people who are not acquainted with the semantics and the syntax of the domain specific language. The knowledge about the domain in question shall be transferred into the language which the user is well aware of, hence the burden of learning the specifics of a new language shall be not present. In this particular case we shall elaborate on the embedding of OWLlink into the JAVA ™ programming language.

1.2 Domain-Specific Languages

Domain-specific languages have many potential advantages in terms of software engineering ranging from increased productivity to the application of formal methods.

In contrast to a general purpose language (GPL), a DSL is a language that is expressive uniquely over the specific features of programs in a given problem domain; it is created specifically to solve problems in that domain and is not intended to be able to solve problems outside it. It is often small and more declarative than imperative; it may be textual or graphic.

The possibility of writing code in terms close to the level of abstraction of the initial problem domain is the characteristic property of domain-specific languages.

1.3 Scope of this work

The task that shall be fulfilled in this project work is to embed the OWLlink interface into the JAVA ™ programming language. The approach involves building an ecore metamodel based on OWLlink draft specification and then automatically generating the APIs required for method chaining from the metamodel using a tool. This tool

generates a class (expression builder) that encapsulates all of the methods generated from the metamodel. As a tool to support this transformation DSL2DJT shall be used. The internal DSL approach, a technique explained by Martin Fowler in his book on DSLs, which allows embedding DSL expressions in JAVA ™ code is used for the generation of DSL APIs. Additional information about the DSL2DJT tool could be found in the Eclipse Corner Article [Gar08] and the topic of internal DSLs is thoroughly presented in the online draft of Martin Fowler's book [Fow09a].

1.4 Document Structure

In this chapter 1 the reader shall be presented with the general idea behind this project work. Additional sources of information and general overview of the problem fields are also topic of this chapter.

In chapter 2 the center of interest is the field of domain-specific languages. The main characteristics of such a language in comparison to a general purpose language are outlined. Key patterns for the development of DSLs are presented, also design methodologies and implementation techniques of domain-specific languages.

In chapter 3 the reader is to be acquainted with the domain specific language, OWLlink. It provides a declarative interface to OWL reasoners.

In chapter 4 information about the implementational part of the work is presented. The tools used during the implementation phase and issues resolved in that phase are also documented. Examples are shown of a test run simulated between a JAVA client and an OWLlink enabled server, queries composed out of the internal DSL APIs generated from the Ecore models. Conclusion from the work done and outlook on the future issues round this chapter off.

The appendix A is subdivided into three parts: the first one is focused on the Existing Internal DSL, the second is populated with examples from the generated code out of the ecore models, and list of acronyms used throughout the paper rounds it off.

1.5 Literature

The sources used during the course of writing of this project work could be arbitrarily divided into two groups.

First, there was the need to gain an overview about the approaches used in the development or embedding of domain-specific languages. A good starting point offering a systematic overview on the field of domain-specific languages could be found in the annotated bibliography by van Deursen, Klint and Visser[vDKV00]. Basically, the mentioned paper and Martin Fowler's website , in particular the part on DSL `http://martinfowler.com/bliki/DomainSpecificLanguage.html`, are the two main sources of information in the chapter 2.

As a side remark, the earliest documented case of the approach that shall be used in this paper, is the one of "embedding" a language, believed to be explained for the first time in the paper by Paul Hudak from June 1996 [Hud96]. The author is reporting on the techniques used during the building of Domain-Specific *Embedded* Languages "... tailoring it in special ways to the domain of interest ...". The functional language Haskell is used as the host language, but nevertheless the ideas in the report are worth exploring.

Secondly, one should familiarize oneself with the language itself that shall be embedded into the host language. In this case as a source of information about the query language, OWLlink, the website `http://www.owllink.org/` was used. As

of the time of the writing of the text the main source was a working draft as it cannot be finalized before the OWL 2 language specification [MPSH08] has reached W3C recommendation status, which forms the base of the draft. An accompanying article [LLN+08b] showing among other things a XML binding example of a sample configuration request and a corresponding reasoner response is a useful reference.

Chapter 2

Domain-Specific Languages

Summary. *The main focus in this chapter shall be an introduction to domain-specific languages (DSLs). The main differences of such a language in comparison to a general purpose language are outlined. Key patterns for the development of DSLs are presented, also design methodologies and implementation techniques of domain-specific languages. In conclusion the technique which shall be used for embedding OWLlink is presented.*

2.1 Introduction

Domain-specific language is a broad term and can refer both to a fully implemented language and a specialized API that looks like a sublanguage. To give a definition of a domain-specific language on which all the experts in the field of computer science agree is a task doomed to fail. The goal set is to outline the core characteristics of a DSL.

At the beginning we shall start with a definition found in "Domain-Specific Languages" by van Deursen et al.[vDKV00]:

> A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.

Without going into too much detail and loosing the thread of the writing, from all the definitions that could be collected the main idea which stays behind all of them is that a DSL is a computer language targeted to a particular kind of problem, in a particular kind of domain of interest. The core functionality of a DSL should be its ability to offer a solution to a problem in specific area.

The notion that these languages are *small* stems from the fact, that in most of the cases they offer a restricted suite of notations and abstractions. In the literature one would be confronted with synonymous terms micro-languages and little languages, apparently both representing the aforementioned idea.

Domain-specific languages are usually declarative, they can be viewed as specification languages as well as programming languages. Many DSLs are supported by a DSL compiler which generates applications from DSL programs. In this case the

DSL compiler is referred to as application generator, and the language as application-specific [Cle88]. Other DSLs, such as YACC [Ben86] or Abstract Syntax Definition Language ASDL, are not aimed at programming (specifying) complete applications, but rather at generating libraries or components. Also, DSLs exist for which execution consists in generating documents (TEX), or pictures (PIC [Ben86]). A common term for DSLs geared towards building business data processing systems is 4^{th} Generation Language ($4GL$).

DSL find use in various areas: in software engineering (software architectures, databases, financial products), in systems software (description and analysis of abstract syntax trees, video device driver specification, operating system specialization), in multimedia (image manipulation, 3D animation, web computing), in telecommunication (communication protocols, telecommunication switches) to name a few.

2.2 DSL Advantages and Disadvantages

Using a domain-specific languages has its strengths and weakness, and the decision whether to adopt such a DSL approach to tackle a particular problem should be considered after the evaluation of the benefits which it bears.

In favor of using DSL one could mention:

- allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain
- DSLs embody domain knowledge, thus enabling the conservation and reuse of knowledge
- allow validation and optimization at the domain level
- DSL programs are concise, self-documenting to a large extent
- improvement of quality, productivity, reliability, maintainability, portability and reusability.

Arguments that shall be considered before the use of domain-specific language are:

- the costs of designing, implementing and maintaining
- the costs of education for DSL users
- the tools required to develop with it (IDE)
- the limited availability
- the difficulty of finding the proper scope for a DSL
- the difficulty of balancing between domain-specificity and general-purpose programming language constructs
- the potential loss of efficiency when compared with hand-coded software.

2.3 DSL Design Methodology and Implementation

The development of a domain-specific language typically involves the following steps: analysis, implementation and use. The required steps to be executed during this methodology are the following:

1. identify the problem domain
2. gather all relevant knowledge in this domain
3. cluster this knowledge semantic notions and operations on them
4. design a DSL that concisely describes applications in the domain
5. construct a library that implements the semantic notions
6. design and implement a compiler that translates DSL programs to a sequence of library calls
7. write DSL programs for all desired applications and compile them.

In this work the focus falls on the implementation part. Let us start with the interpretation or the compilation approach.

This is the classical approach to implement a new language. Standard compiler tools can be used, or tools dedicated to the implementation of DSLs.

The main advantage of building a compiler or interpreter is that the implementation is completely tailored towards the DSL and no concessions are necessary regarding notation, primitives and the like. Also, error detection, static analysis, and optimizations can be done at the domain level. Clearly, an important problem is the cost of building such a compiler or interpreter from scratch, and the lack of reuse from other implementations, although some DSL tool sets are particularly designed to overcome such problems.

As an alternative to implementing a DSL from scratch, a DSL can be implemented by extending a given base language.

The main advantage of this approach is that all features of the base language remain available and need not be reimplemented. When implementing domain-specific extensions of a base language, the implementation of the base language can be reused in three different ways:

Preprocessing or macro processing

In this approach the new constructs are translated to statements in the base language by a preprocessor. The main advantage of this approach is simplicity. Its main disadvantage is that static checking and optimization are not done at the domain level. Consequently, generated code is error prone, and the user is provided with feedback on these errors at the level of the base language, or only at run-time.

Extensible compiler or interpreter

This approach is similar to the previous one, but the preprocessing phase is now integrated in the compiler. The advantage is that more type checking and better optimization is possible.

Apart from building a dedicated DSL compiler or interpreter, or reusing the implementation of an underlying base language, other implementation techniques may be used.

Embedded languages / domain-specific libraries

In this approach, existing mechanisms such as definitions for functions or operators with user-defined syntax are used to build a library of domain-specific operations. The syntactic mechanisms of the base language are used to express the desired content of the domain.

An advantage of this approach is that the compiler or interpreter of the base language is reused as is for the DSL. The main limitation is in the expressiveness of the syntactic mechanisms in the base language. In many cases, the optimal domain-specific notation has to be compromised to fit the limitations of the base language.

Another distinction that concerns the categorization in different types of domain-specific languages is the division in external and internal DSL [Fow09a]. This categorization is more coarse-grained than the ones described above, but in essence the idea behind is the same. Internal DSLs according to Fowler's viewpoint come close to the embedded languages or domain-specific libraries. As Martin Fowler points out the internal DSL uses the host language in a particular and limited style, a handful of language constructs are used (only those which justify their use). In the annotated bibliography of Domain-Specific Languages [vDKV00] embedded languages and domain-specific libraries are used interchangeably, whereas Fowler stresses on the fact that sometimes the boundary is to close to call, therefore, he concludes, API and Internal DSL are in the majority of cases essentially the same.

Embedding OWLlink provides an internal DSL for building OWLlink statements. The approach to embed OWLlink involves building an ecore metamodel based on OWLlink draft specification and then automatically generating the APIs required for method chaining from the metamodel using DSL2JDT tool. This tool generates a class (expression builder) that encapsulates all of the methods generated from the metamodel. As already earlier mentioned an explanation of the techniques behind the fluent interface, expression builder, method chaining and the like are offered at Martin Fowler's website in the section on Implementing an Internal DSL [Fow09b].

And that is exactly the way to go in this project work. We shall generate a progressive interface out of a ecore diagram model using a tool called DSL2DJT.

For further information on the techniques behind this approach take a look at the article at the Eclipse Corner Article [Gar08].

The concept of embedding a DSL in Java is an area getting a fair amount of attention from researchers and the like. Many efforts were made to improve language integration by providing fluent interfaces or other lightweight wrappers around SQL, JDBC. In section A.1 of the appendix some of these efforts are briefly presented and references to them are provided.

Chapter 3

OWLlink

Summary. *In the following chapter the focus is on the proposed draft of OWLlink interface, which provides an implementation-neutral mechanism for accessing OWL reasoner functionality. From the UML diagrams presented in this specification draft ecore models were created which served for the generation of the source code that could be found in the following part of the appendix A.2.*

3.1 Overview

As main source of information for this chapter the author used the working draft as of April 11th 2008 [LLN08a]. This version of the OWLlink structural specification refers to the public OWL 2 working draft as of April 11th 2008 [MPSH08].

OWLlink provides a declarative interface to OWL reasoners. It is intended to be a mechanism giving client applications access to reasoning functionality provided by a server. The OWLlink core defines primitives for handling and manipulation of OWL Knowledge Bases (KBs), such as asserting axioms, as well as primitives for asking questions about KB entities. Each KB given to the reasoner is a self-contained collection of facts and axioms.

3.2 Basic Protocol

OWLlink is a client-server protocol. In the following only a few of the UML class diagrams are presented in the form of ecore diagrams. The latter were used later on in the generation of the emdedded JAVA source code. For further information the reader is advised to check the current version of the aforementioned draft document. Due to the fact that OWLlink is based on OWL a great number of classes are taken from the OWL 2 specification.

3.2.1 Sessions and Messages

The basic interaction pattern of OWLlink is that of request-response. The fundamental objects used in the protocol are presented in figure 3.1. OWLlink provides the building blocks needed to communicate to the server, whereas the actual implementation is left to the transport mechanism used to access the this very OWLlink server.

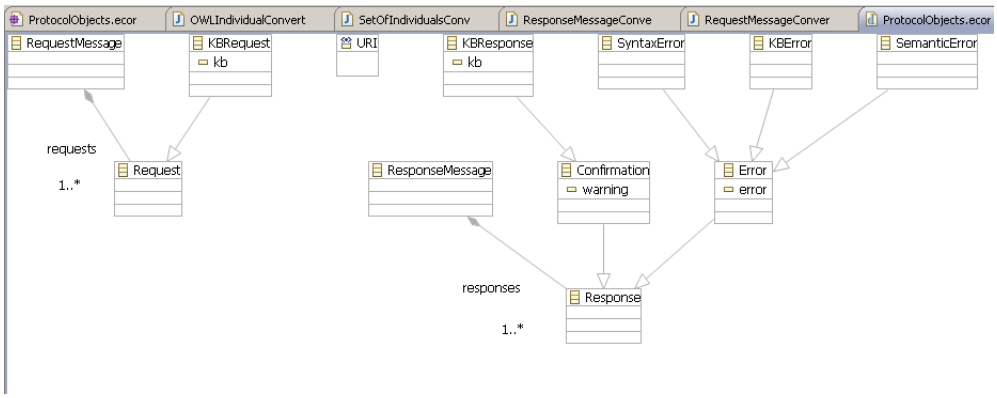

Figure 3.1: OWLlink Protocol Objects

The requests and the responses are packed into messages; one `RequestMessage` object containing a list of `Request` objects, the same holds for the `ResponseMessage` objects.

For every request the server must return a corresponding answer which depending on the ask could contain additional information. Is the case that a request fails for whatever reason the client should receive an `Error` response.

3.2.2 Basic Asks

OWLlink provides means for retrieving information about the Knowledge Base that could be inferred from the axioms in KB. In the next figures general requests for retrieving information about KB entities 3.2 are shown along with KB status requests. The generated source code could be found at the back of this work, the expression builder for retrieving knowledge base entities at A.2.2.

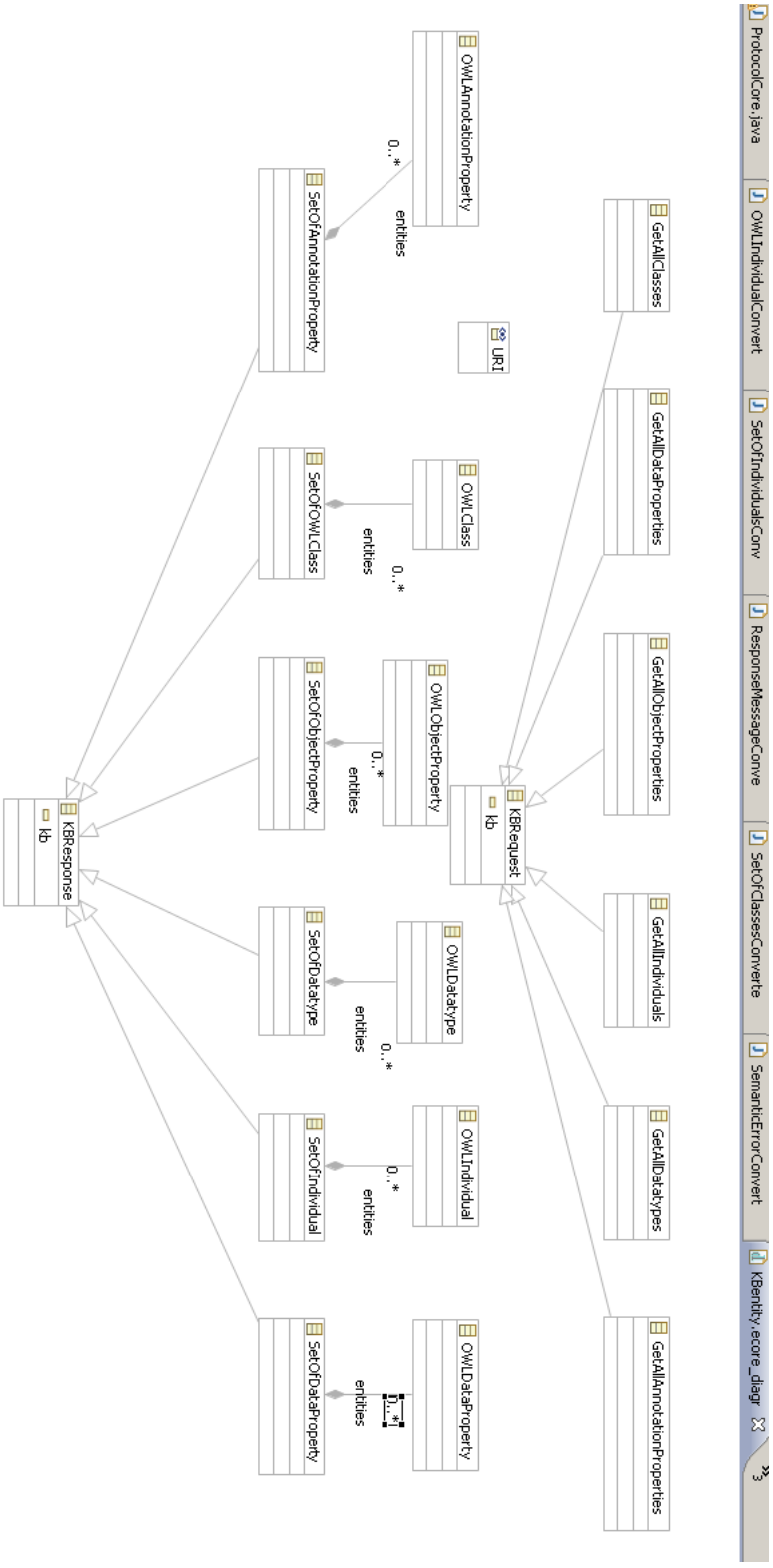

Figure 3.2: Basic Asks for Retrieving KB Entities

Chapter 4

Implementation

Summary. *The main focus in this part shall be the implementation side of the project work. The tools used to fulfill the ask of embedding OWLlink into JAVA shall be presented as well as the approaches that led to the successful communication with the RacerPro reasoner.*

To fulfill the task of embedding OWLlink into the JAVA programming language a couple of tools shall be used. Information about each one of them could be found in the subsequent parts of this chapter.

The first step in the embedding of OWLlink into JAVA will be achieved by using DSL2JDT tool. By using it one automatically generates the code of the API from Ecore models which describe the syntax of the domain-specific language to be embedded.

The second step involves the serialization of the OWLlink messages from JAVA objects into XML files. These files are to be sent to a reasoner in suitable form in order to be processed. The responses received from the server in turn would be deserialized to JAVA objects, respectively to JAVA embedded OWLlink responses. A handy library which would be used in this project work is XStream.

The final step in the successful implementation represents the communication to the reasoner. In order to test the OWLlink messages created in JAVA one needs an application that can send these to the reasoner. In this case the reasoner that would answer the requests from the client application shall be the RacerPro. As the approach is to access the server from a JAVA environment the choice was to use a RacerPro API, DIG Client in JAVA. This application offers a possibility to communicate with the RacerPro server via HTTP using the DIG protocol. As the interest in this particular work lies in OWLlink the only functionality used shall be the HTTP transmission of XML files containing the OWLlink messages and subsequent storing of the responses into XML files.

4.1 DSL2DJT

As a starting point of the generation of the OWLlink messages in JAVA the Ecore model corresponding to the UML classes from the specification are taken. Out of this metamodel a .genmodel is generated which in its turn is used to generate the embedded DSL. After the desired ExpressionBuilder is produced and added to the folder with the model generated source code OWLlink messages can be constructed. In figure 4.1 the model out of which the general building blocks for the OWLlink

messages in JAVA are needed is shown. The ProtocolObjectsExpression Builder's source code could be found in this section A.2.1.

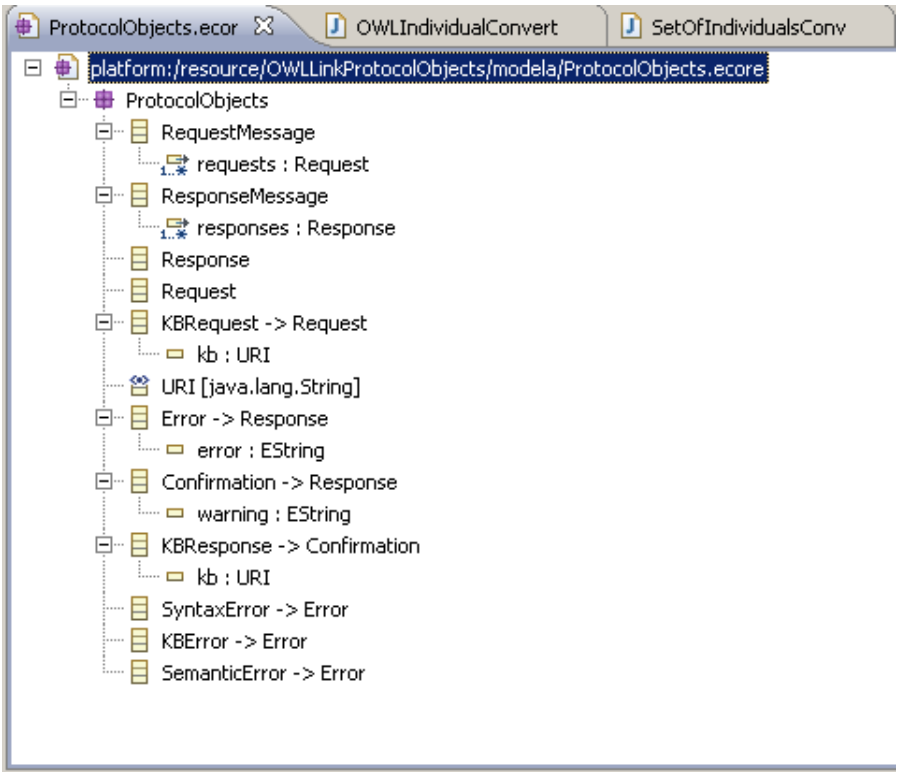

Figure 4.1: OWLlink Protocol Objects Ecore model

In this manner expression builders are created for all the classes in OWLlink specification and with this so called Internal DSL any user can construct OWLlink messages. The next step would be to send this messages to a reasoner application. To accomplish this an approach is needed to serialize the OWLlink constructs into XML files.

4.2 Xstream

The task to serialize objects to XML and back again is tackled with the XStream library. Due to the fact that in the process of generation of the Internal DSL the JAVA-embedded OWLlink expressions inherit methods that are necessary only internally for the DSL2DJT tool the use of the default converters does not deliver the result required. Custom converters have to be implemented. For each and every one OWLlink class a converter's *marshal()* and *unmarshal()* methods must me implemented so the XML snippets obey the pattern that is required by the OWLlink enabled reasoner in order to process the requests. An example of this is documented in listing 4.1.

```
1  @Override
2  public void marshal(Object object, HierarchicalStreamWriter writer,
3      MarshallingContext context) {
4    OWLClassImpl classSample = (OWLClassImpl) object;
5    writer.addAttribute("URI", classSample.getKb() );
```

```
6
7   }
8
9   @Override
10  public Object unmarshal(HierarchicalStreamReader reader,
11      UnmarshallingContext context) {
12    OWLClass classHolder = shorthand.SameOld.C.oWLClass().toAST();
13    classHolder.setKb(reader.getAttribute("URI"));
14    return classHolder;
15  }
```

Listing 4.1: Converter for the OWLClass object

Into consideration has to be taken the fact that the serialized XML snippets have to be written to a well-formed XML file before being sent to the server. Another important fact not to be omitted is the encoding of the files. Apart from the obligatory declaration tag it also should be verified that UTF-8 character encoding is used while writing the snippets to a file.

Another issue that came into light during the serializing was the object equality. As the generated by DSL2DJT code did not take any care about the generation of the *equals()* method in the OWLlink JAVA classes this turned out to be a bit of problem.

Every JAVA object inherits a set of base methods from *java.lang.Object* that every client can use, among these are also the equality methods *equals(Object)* and *hashCode()*. The purpose is to determine whether the argument is equal to the current instance. This method is used in virtually all of the *java.util* collections classes, and other low-level libraries (such as Remote Method Invocation, Java Database Connectivity, etc.) implicitly depend on the correct behavior. The method should return true if the two objects can be considered equal and false otherwise. It is left up to the developer to decide what data is considered equal for each individual class.

Although there is freedom in the implementation of the aforementioned methods, some conditions should be taken into consideration in order to get the correct behavior.

First, the *equals()* contract must be preserved. The Javadoc declares that it must be:

Reflexive An object must always be equal to itself; *a.equals(a)*

Symmetric If two objects are equal, then they should be equal in both directions; *if a.equals(b)*, then *b.equals(a)*

Transitive If an object is equal to two others, then they must both be equal; if *a.equals(b)* and *b.equals(c)*, then *a.equals(c)*

Non-null An object can never be equal to null; *a.equals(null)* is always `false`.

Second, the Liskov Substitution Principle which states you should be able to use (substitute) a subclass instance where a superclass instance is required must be taken into account at developer's discretion. There is a debate in the developers' community whether one can redefine equality on several levels of the class hierarchy while keeping the *equals()* contract. There is no correct answer to this question, every implementation should justify the reasoning behind the proposed behavior.

The last issue to be resolved about the object equality is whether to use *instanceof* or *Class* comparison. The reason that some people favor the *instanceof* approach is that when you use the *getClass* approach, you have the restriction that objects are only equal to other objects of the same class, the same run time type. But on the other hand it turns out that because the *getClass* approach does let you add aspects while preserving the equals contract, other people favor it. As with the LSP issue it

is a decision to be considered on a case-by-case basis. The bottom line is that every OWLlink class that is a subclass of either `KBRequest` class or `KBResponse` class must implement its own *equals()*, *hashCode()* and *canEqual()* methods.

An example implementation is listed in 4.2.

```
1  @Override
2  public boolean equals(Object other)
3  {
4    boolean result = false;
5    if (other instanceof GetAllClassesImpl)
6    {
7      GetAllClassesImpl that = (GetAllClassesImpl) other;
8      result = (that.canEqual(this) && this.hashCode() == that.hashCode
           ());
9    }
10   return result;
11
12 }
13 @Override
14 public int hashCode()
15 {
16   if (super.kb == null)
17   {
18     return 0;
19   }
20   return super.kb.hashCode();
21 }
22
23 public boolean canEqual(Object other)
24 {
25   return (other instanceof GetAllClassesImpl);
26 }
```

Listing 4.2: Implementation of object equality methods

4.3 RacerPro API for JAVA

After the OWLlink messages are constructed in JAVA and serialized to UTF-8 encoded well-formed XML documents the next logical step is to try to send this message to OWLlink enabled reasoner. The server to whom the messages shall be sent will be RacerPro [KG09b]. On the website of the reasoner some APIs are published. The choice fell on DIG Client in JAVA. As a matter of fact this is an interface to communicate with the server via HTTP in the DIG protocol. It is not intended to use this access protocol, as OWLlink is the successor of DIG 1.1 interface. The only reason behind this JAVA API is the ability to send HTTP based messages to the RacerPro. Further details about the API and source code could be obtained in the following section of the RacerPro website [KG09a].

In essence the communication to the RacerPro reasoner boils down to starting the server in OWLlink mode by giving the following parameter `-protocol OWLlink`, building a Knowledge Base and subsequently querying the server with the RacerPro API. The client application needs as parameters the name of the server, the port on which is running the service, and paths to two files; one for the request and one for the response message. At this point the results from the previous steps step in. The OWLlink message serialized to a XML file is fed to the OWLlink enabled server via the HTTP functionality of the RacerPro API client.

4.4 Example Scenario

In this section an example run between a client and a server is documented with the help of code excerpts.

An OWLlink message containing request about the Knowledge Base will be constructed in JAVA. The client is interested in obtaining information about all the classes and individuals in KB with URI *KB_1*.

```
1    RequestMessage reqMessage = null;
2    GetAllClasses askForAllClasses = C.getAllClasses().kb("KB_1").toAST
         ();
3    GetAllIndividuals allIndi = C.getAllIndividuals().kb("KB_1").toAST()
         ;
4    reqMessage = ProtocolCore.Q.requestMessage().requests((Request)
         askForAllClasses,
5        (Request)allIndi).toAST();
```

Listing 4.3: Request for all classes and individuals in a KB

The next thing to do is to serialize the JAVA OWLlink request message to a XML snippet. First a serialization is shown using the default converters provided by the XStream library. Converting with out-of-the-box implementations yields:

```
1  <ProtocolObjects.impl.RequestMessageImpl>
2    <eFlags>1</eFlags>
3    <eContainerFeatureID>0</eContainerFeatureID>
4    <requests class="org.eclipse.emf.ecore.util.EObjectContainmentEList"
         serialization="custom">
5      <unserializable-parents/>
6      <org.eclipse.emf.common.util.BasicEList>
7        <default>
8          <size>2</size>
9        </default>
10       <int>4</int>
11       <retrieveEntity.impl.GetAllClassesImpl>
12         <eFlags>1</eFlags>
13         <eContainer class="ProtocolObjects.impl.RequestMessageImpl"
               reference="../../../.."/>
14         <eContainerFeatureID>-1</eContainerFeatureID>
15         <kb>KB_1</kb>
16       </retrieveEntity.impl.GetAllClassesImpl>
17       <retrieveEntity.impl.GetAllIndividualsImpl>
18         <eFlags>1</eFlags>
19         <eContainer class="ProtocolObjects.impl.RequestMessageImpl"
               reference="../../../.."/>
20         <eContainerFeatureID>-1</eContainerFeatureID>
21         <kb>KB_1</kb>
22       </retrieveEntity.impl.GetAllIndividualsImpl>
23     </org.eclipse.emf.common.util.BasicEList>
24     <org.eclipse.emf.ecore.util.EcoreEList>
25       <default>
26         <dataClass>ProtocolObjects.Request</dataClass>
27         <owner class="ProtocolObjects.impl.RequestMessageImpl" reference
               ="../../../.."/>
28       </default>
29     </org.eclipse.emf.ecore.util.EcoreEList>
30     <org.eclipse.emf.ecore.util.EObjectEList>
31       <default>
32         <featureID>0</featureID>
33       </default>
34     </org.eclipse.emf.ecore.util.EObjectEList>
35   </requests>
36 </ProtocolObjects.impl.RequestMessageImpl>
```

Listing 4.4: Serialization with out-of-the-box converter

Obviously this is not the format anticipated from the server nor this is the recommendation from the specification, so a custom converter should be implemented for all the objects involved in the construction of the OWLlink message. For this request necessary are implementations of *GetAllClassesConverter*, *GetAllIndividualsConverter* and *RequestMessageConverter*. The implementation details for the latter are to be seen in listing 4.5.

```
1  public void marshal(Object arg0, HierarchicalStreamWriter arg1,
2      MarshallingContext arg2) {
3    RequestMessageImpl message = (RequestMessageImpl) arg0;
4
5        arg1.addAttribute("\n xmlns:xsi", "http://www.w3.org/2001/
             XMLSchema-instance");
6        arg1.addAttribute("\n xsi:schemaLocation", "http://www.owllink.
             org/owllink-xml http://www.owllink.org/owllink-xml.xsd");
7        arg1.addAttribute("\n xmlns", "http://www.owllink.org/owllink-
             xml");
8
9
10       EObjectContainmentEList<ProtocolObjects.Request> listReq =
11         new EObjectContainmentEList<ProtocolObjects.Request>(Request.
               class, message, 0);
12       listReq = (EObjectContainmentEList<Request>) message.
             getRequests();
13       KBRequestConverter ownOne = new KBRequestConverter();
14       for(Iterator<Request> it = listReq.iterator();it.hasNext();)
15       {
16         Request toHandle = it.next();
17
18         if(toHandle.getClass().equals(ProtocolObjects.impl.
               KBRequestImpl.class))
19         {
20
21           arg1.startNode("KBRequest");
22           arg2.convertAnother(toHandle, ownOne);
23           arg1.endNode();
24         }
25         else if (toHandle.getClass().equals(GetAllClassesImpl.class))
26         {
27           arg1.startNode("GetAllClasses");
28           arg2.convertAnother(toHandle, new GetAllClassesConverter());
29           arg1.endNode();
30         }
31         else if (toHandle.getClass().equals(GetAllIndividualsImpl.
               class))
32         {
33           arg1.startNode("GetAllIndividuals");
34           arg2.convertAnother(toHandle, new GetAllIndividualsConverter
                 ());
35           arg1.endNode();
36         }
37       }
38 }
```

Listing 4.5: Marshal method for the RequestMessage class

After implementing the aforementioned converters and registering them so they can be used ...

```
1      xs.registerConverter(new RequestMessageConverter());
2
```

```
3         xs.registerConverter(new KBRequestConverter());
4
5         xs.registerConverter(new GetAllClassesConverter());
6
7         xs.registerConverter(new GetAllIndividualsConverter());
8
9         xs.registerConverter(new ResponseMessageConverter());
10
11        xs.registerConverter(new SetOfIndividualsConverter());
12
13        xs.registerConverter(new SetOfClassesConverter());
14
15        xs.registerConverter(new OWLClassConverter());
16
17        xs.registerConverter(new OWLIndividualConverter());
```

Listing 4.6: Registering the custom converters

... the next step is to write the XML snippets to a conformed XML file, i.e. UTF-8 encoded and well-formed.

```
1         OutputStream out = null;
2         OutputStreamWriter writer = null;
3         StringWriter testTo = new StringWriter();
4         testTo.write("<?xml version='1.0' encoding='UTF-8'?>\n");
5         try{
6           out = new BufferedOutputStream(new FileOutputStream("c:/temp/
              sameOldDecl.xml"));
7           writer = new OutputStreamWriter(out,"UTF-8");
8
9           writer.write("<?xml version='1.0' encoding='UTF-8'?>\n");
```

Listing 4.7: Writing the serialized object to a XML file

Serializing the *RequestMessage* object with the custom converters results into the following XML file ...

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <RequestMessage
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://www.owllink.org/owllink-xml http://www.
        owllink.org/owllink-xml.xsd"
5   xmlns="http://www.owllink.org/owllink-xml">
6    <GetAllClasses kb="KB_1"/>
7    <GetAllIndividuals kb="KB_1"/>
8 </RequestMessage>
```

Listing 4.8: Serialization with custom converter

With this XML file and the HTTP functionality of the DIG client an existing Knowledge Base can be easily queried using RacerPro. All is needed to start the server in OWLlink mode and send the file as request. The response message is saved to a file whose path is provided as second parameter to the RacerPro API client application. The response to the query can be viewed in listing 4.9.

```
1 <ResponseMessage xmlns="http://www.owllink.org/owllink-xml#"
2                  xmlns:owl="http://www.w3.org/2002/07/owl#">
3
4 <SetOfClasses>
5  <owl:Class URI="C"/>
6  <owl:Class URI="B"/>
7  <owl:Class URI="A"/>
8  <owl:Class URI="E"/>
9  <owl:Class URI="D"/>
10 </SetOfClasses>
```

```
11
12 <SetOfIndividuals>
13  <owl:Individual URI="iA"/>
14 </SetOfIndividuals>
15 </ResponseMessage>
```

Listing 4.9: Answer of a OWLlink request message

In order to test the deserialization abilities of the custom converters this time necessary are implementations of *SetOfClassesConverter*, *SetOfIndividualsConverter* and *ResponseMessageConverter*. The *unmarshal()* method responsible for the deserialization of a `ResponseMessage` class follows.

```
1   public Object unmarshal(HierarchicalStreamReader reader,
2       UnmarshallingContext context) {
3     ResponseMessage response = null;
4     ResponseMessageImpl needIt = new ResponseMessageImpl();
5     SetOfOWLClass setClass = null;
6     SetOfIndividual setIndividuals = null;
7     EObjectContainmentEList<ProtocolObjects.Response> listResp =
8           new EObjectContainmentEList<ProtocolObjects.Response>
9     (Response.class, needIt, 0);
10    while(reader.hasMoreChildren())
11    {
12      reader.moveDown();
13
14      if("SetOfClasses".equals(reader.getNodeName()))
15      {
16        setClass = (SetOfOWLClass)context.convertAnother(response,
17            SetOfOWLClassImpl.class,
18            new SetOfClassesConverter());
19        listResp.add((Response) setClass);
20      }
21      else if("SetOfIndividuals".equals(reader.getNodeName()))
22      {
23        setIndividuals = (SetOfIndividual)context.
24                convertAnother(response,
25            SetOfIndividualImpl.class,
26            new SetOfIndividualsConverter());
27        listResp.add((Response) setIndividuals);
28      }
29      reader.moveUp();
30    }
31    response = shorthand.Test.ProtocolCore.Q.
32    responseMessage().
33    responses((Response)setIndividuals, (Response)setClass).
34    toAST();
35    return response;
36  }
```

Listing 4.10: unmarshal() method impementation for a ResponseMessage object

The approach is to deserialize the XML response message to a ResponseMessage object which in turn shall be serialized to a file, and this file in the end compared to the original answer from the RacerPro server.

```
1             ResponseMessage fromXml =
2             (ResponseMessage) xs.fromXML(
3                 new FileInputStream("c:/temp/answerW.xml"));
```

Listing 4.11: Deserializing from a XML file

In the final step of this run we compare the file from 4.9 against the one obtained from the deserialized object in listing 4.11. The elements are the same except the fact that the order of appearance is reversed.

```
<ResponseMessage xmlns="http://www.owllink.org/owllink-xml#" xmlns:owl="
    http://www.w3.org/2002/07/owl#">
  <SetOfIndividuals>
    <Individual URI="iA"/>
  </SetOfIndividuals>
  <SetOfClasses>
    <Class URI="C"/>
    <Class URI="B"/>
    <Class URI="A"/>
    <Class URI="E"/>
    <Class URI="D"/>
  </SetOfClasses>
</ResponseMessage>
```

Listing 4.12: Serialization of a deserialized ResponseMessage object

4.4.1 Embedded Queries

In this final part the result of the embedding of OWLlink in the host language is presented. Examples of requests and responses built from the generated expression builders are shown.

An approach for improving the usability of the generated internal DSL is to extend the expression builder class generated by the DSL2JDT tool [Pri08]. As is shown in figure 4.2 the class extended from the expression builder generatedfrom the diagram about class hierarchy queryoffers the user the statements needed to build an appropriate request and response.

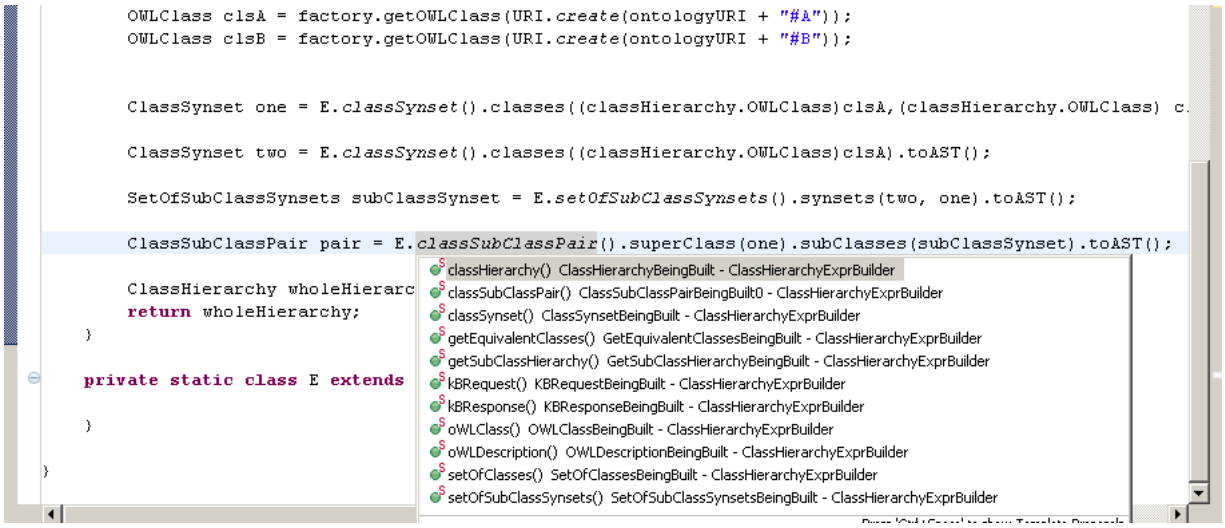

Figure 4.2: Fluent interface for a class hierarchy query

After making this minor modification the content assist of the IDE provides the methods needed when embedding a OWLlink query. Some examples follow in order to show how this works. First let us have a look how a simple response is generated with the expression builder.

```
public static KBResponse returnAnswer(){
  BooleanResponse answer = C.booleanResponse().toAST();
  answer.setResult(false);
  return answer;
```

```
5   }
```

Listing 4.13: Simple Boolean Response

Next in listing 4.14 a string response is constructed.

```
1   public static KBResponse returnStringAnswer(){
2     StringResponse answer = C.stringResponse().result("A meaningfull
          answer!").toAST();
3     return answer;
4   }
```

Listing 4.14: Simple String Response

A general request to a reasoner supporting OWLlink interface would look like the one in listing 4.15. It checks whether a Knowledge Base,i.e. a self-contained collection of facts and axioms, is satisfiable.

```
1   public static KBRequest checkStg(){
2
3     IsKBSatisfiable request = C.isKBSatisfiable().kb("http://example.com
          ").toAST();
4     return request;
5   }
```

Listing 4.15: Request for Satisfiability of a KB

4.5 Conclusion and Outlook

4.5.1 Conclusion

The embedding of OWLlink into JAVA has been achieved by using a model-driven development of Internal DSL by means of DSL2DJT tool. The communication with a OWLlink reasoner was supported by a serialization library, XStream, and by the HTTP functionality of JAVA RacerPro API, DIG client. All these tools helped to query an existing Knowledge Base. In summary, the goal of this project work was achieved by method chaining OWLlink constructs,be it `ResponseMessages` or `RequestMessages`, serialization and deserialization of these JAVA objects to XML files, and finally transmitting those files via HTTP to RacerPro, a reasoner supporting the OWLlink protocol.

4.5.2 Outlook

The future work could be split in two perspectives. First, the communication with a reasoner could be refined in a way that leaves the user with more possibilities for access to the reasoner functionality. One can start from a more comfortable and easy to use JAVA API than the DIG client implementation of the HTTP functionality and end at implementing a completely new way for communication, i.e. other than the HTTP approach used in this work.

Secondly, OWLlink relies on the OWL 2 Specification for the language used to express Knowledge Base axioms. OWL 2 has also been specified using UML, so OWLlink reuses many UML classes provided by OWL 2. As the host language for embedding the query mechanism happens to be JAVA, a reasonable choice is to use some constructs of the OWL API [oM09], a Java interface and implementation for the W3C Web Ontology Language OWL. This could lead to a seamless extension of the existing OWLAPI with OWLlink functionality, bringing a JAVA embedded mechanism for accessing OWL reasoners to the users.

How to build a potential request and response to a reasoner is shown in listing 4.16. This is how the construction of a OWLlink message would look from purely implementational perspective, if one would be using the approach provided by the OWL API.

```
1  package shorthand;
2
3  import java.net.URI;
4
5  import org.semanticweb.owl.apibinding.OWLManager;
6  import org.semanticweb.owl.model.OWLClass;
7  import org.semanticweb.owl.model.OWLDataFactory;
8  import org.semanticweb.owl.model.OWLOntologyManager;
9
10 import classHierarchy.ClassHierarchy;
11 import classHierarchy.ClassHierarchyExprBuilder;
12 import classHierarchy.ClassSubClassPair;
13 import classHierarchy.ClassSynset;
14 import classHierarchy.GetSubClassHierarchy;
15 import classHierarchy.KBRequest;
16 import classHierarchy.KBResponse;
17 import classHierarchy.SetOfSubClassSynsets;
18
19 public class ClassQuery {
20
21   public static KBRequest requestSubHierarchy(){
22     // We first need to obtain a copy of an OWLOntologyManager,
23     // which, as the name suggests, manages a set of ontologies.
24         // An ontology is unique within an ontology manager.
25         // To load multiple copies of an ontology, multiple managers
26         // would have to be used.
27         OWLOntologyManager manager = OWLManager.createOWLOntologyManager
               ();
28
29         // Now we want to specify that A is a subclass of B.
30         // To do this, we add a subclass axiom.
31         // A subclass axiom is simply an object
32         // that specifies that one class is a subclass of another class.
33         // We need a data factory to create various object from.
34         //   Each ontology has a reference
35         // to a data factory that we can use.
36         OWLDataFactory factory = manager.getOWLDataFactory();
37
38         // Let's create an ontology and name it
39         // "http://www.co-ode.org/ontologies/testont.owl".
40         // We need to set up a mapping which points to a
41         // concrete file where the ontology will
42         // be stored. (It's good practice to do this even
43         // if we don't intend to save the ontology).
44         URI ontologyURI = URI.create("http://www.co-ode.org/ontologies/
               testont.owl");
45
46         // Get hold of references to class A and class B.
47         // Note that the ontology does not
48         // contain class A or classB, we simply get references
49         // to objects from a data factory that represent
50         // class A and class B
51         OWLClass clsA = factory.getOWLClass(
52                                 URI.create(ontologyURI + "#A"));
53
54         OWLClass clsB = factory.getOWLClass(
55                                 URI.create(ontologyURI + "#B"));
56
57     GetSubClassHierarchy request = E.getSubClassHierarchy().
58                                       kb("http://example.com").
```

```
59                                          _class().
60                                         toAST();
61
62      request.setClass((classHierarchy.OWLClass) clsA);
63
64      return request;
65
66    }
67    public static KBResponse returnClassHierarchy(){
68      // We first need to obtain a copy of an OWLOntologyManager,
69      // which, as the name suggests, manages a set of ontologies.
70          // An ontology is unique within an ontology manager.
71          // To load multiple copies of an ontology, multiple managers
72          // would have to be used.
73          OWLOntologyManager manager = OWLManager.createOWLOntologyManager
                ();
74
75          // Now we want to specify that A is a subclass of B.
76          // To do this, we add a subclass axiom.
77          // A subclass axiom is simply an object that specifies
78          // that one class is a subclass of another class.
79          // We need a data factory to create various object from.
80          // Each ontology has a reference
81          // to a data factory that we can use.
82          OWLDataFactory factory = manager.getOWLDataFactory();
83
84          // Let's create an ontology and name it
85          // "http://www.co-ode.org/ontologies/testont.owl".
86          // We need to set up a mapping which points to a
87          // concrete file where the ontology will
88          // be stored.
89          // (It's good practice to do this even if we
90          // don't intend to save the ontology).
91          URI ontologyURI = URI.create(
92                  "http://www.co-ode.org/ontologies/testont.owl");
93
94          // Get hold of references to class A and class B and class C.
95          // Note that the ontology does not
96          // contain class A or classB, we simply get references
97          // to objects from a data factory that represent
98          // class A and class B and class C.
99          OWLClass clsA = factory.getOWLClass(
100             URI.create(ontologyURI + "#A"));
101
102         OWLClass clsB = factory.getOWLClass(
103             URI.create(ontologyURI + "#B"));
104
105         OWLClass clsC = factory.getOWLClass(
106             URI.create(ontologyURI + "#C"));
107
108
109     ClassSynset one = E.classSynset().classes((classHierarchy.OWLClass)
            clsA,
110                                                 (classHierarchy.OWLClass) clsB
                                                        ).
111                                         toAST();
112
113     ClassSynset two = E.classSynset().classes((classHierarchy.OWLClass)
            clsC).
114                                             toAST();
115
116     ClassSynset three = E.classSynset().
117                             classes((classHierarchy.OWLClass)clsC,
118                                 (classHierarchy.OWLClass) clsB).
119                         toAST();
```

```
120
121     SetOfSubClassSynsets subClassSynsetAB = E.setOfSubClassSynsets().
122                                             synsets(two, one).
123                                             toAST();
124
125
126     ClassSubClassPair pair = E.classSubClassPair().
127                               superClass(three).
128                               subClasses(subClassSynsetAB).
129                               toAST();
130
131     ClassHierarchy wholeHierarchy = E.classHierarchy().
132                                       pairs(pair).
133                                       toAST();
134
135     return wholeHierarchy;
136   }
137
138   private static class E extends ClassHierarchyExprBuilder{
139
140   }
141
142 }
```

Listing 4.16: Request and Response for Satisfiability of a KB

Appendix A

Appendix

A.1 Existing Internal DSL

A brief and not exhaustive overview of some of the existing embedded domain-specific languages follows. The idea is to give an insight about the work done in this area.

A.1.1 Squill

Example of a typesafe SQL EDSL for Java. Squill is a slick internal DSL for writing SQL queries in pure Java. It uses the database metadata and generics to catch as many errors as possible during compilation and is almost completely typesafe. At the same time it is designed to allow everything SQL allows you to do, exactly the way SQL is meant to do it. This means that you're encouraged to select only the data you need and no hidden queries are generated for you, leaving you in full control of the query performance. Squill also supports database-specific extensions, allowing you to both use advanced features and fully tweak your queries. In the excerpt A.1 a typical query from the framework is shown.

```
1
2  ComplaintTable c = new ComplaintTable();
3
4   for (Tuple2 tuple2:
5     squill
6       .from(c, c.customer)
7       .where(
8         gt(c.customer.isActive, 0),
9         notNull(c.percentSolved),
10        notNull(c.refoundSum))
11      .orderBy(desc(c.customer.id))
12      .selectList(
13        c.customer.lastName,
14        c.percentSolved)) {
15    System.out.println(
16      "Customer " + tuple2.v1 + " has a complaint solved " + tuple2.v2 +
            "%");
17  }
```

Listing A.1: Squill snippet

A.1.2 JEQUEL

Java Embedded QUEry Language or JEQUEL

A Domain Specific Language for the Structured Query Language (SQL) embedded in Java which offers a syntax highlighting, code completion for tables, fields and keywords and operations, spell and error checking for tables, fields and keywords. As could be seen in the listing A.2 this implementation offers unit tests for SQL fragments along with test parameters and expected results for fragments. For more information have a look at `http://www.jequel.de/`.

```
1  public void testSimpleSql() {
2          final SqlString sql =
3          select(ARTICLE.OID)
4          .from(ARTICLE, ARTICLE_COLOR)
5          .where(ARTICLE.OID.eq(ARTICLE_COLOR.ARTICLE_OID)
6          .and(ARTICLE.ARTICLE_NO.is_not(NULL)));
7
8          assertEquals("select ARTICLE.OID" +
9          " from ARTICLE, ARTICLE_COLOR" +
10         " where ARTICLE.OID = ARTICLE_COLOR.ARTICLE_OID" +
11         " and ARTICLE.ARTICLE_NO is not NULL", sql.toString());
12  }
```

Listing A.2: JEQUEL snippet

A.1.3 Quaere

Quaere is an open source, extensible framework that adds a querying syntax reminiscent of SQL to Java applications. Quaere allows you to filter, enumerate and create projections over a number of collections and other queryable resources using a common, expressive syntax. Quaere detaches queries from the query API used by different queryable resources such as databases, catalogs and other structured data, allowing one language to be used to query numerous resources. The framework's download page is `http://quaere.codehaus.org/`.

```
1  // Setup a JPA entity manager...
2  SessionFactory sessionFactory=new Configuration().configure().
       buildSessionFactory();
3  EntityManagerFactory entityManagerFactory =
4      new EntityManagerFactoryImpl(sessionFactory,
           PersistenceUnitTransactionType.RESOURCE_LOCAL, true);
5  QueryableEntityManager entityManager =
6      new QueryableEntityManager(entityManagerFactory.createEntityManager
           ());
7
8  // Select all customers in the Washington region
9  Iterable<Customer> waCustomers =
10         from("c").in(entityManager.entity(Customer.class)).
11         where(eq("c.getRegion()", "WA")).
12         select("c");
13
14 System.out.println("These customers are located in the Washington region
       :");
15 for (Customer c : waCustomers) {
16     System.out.println(c.getCompanyName());
17 }
```

Listing A.3: Quaere query: Select customers in the Washington region

A.1.4 Typesafe SQL DSL

Kabanov and Raudjärv explain in their paper [KR08] the design guidelines to achieve typesafe embeddings and how manually to build an embedded domain-specific lan-

guage. Some of the conclusions propose mixing the Fluent Interface[1] idiom with static functions, metadata and closures which should provide a better experience to the users than pure method chaining. This helps for easier extension of the DSL and easier reuse of the DSL fragments.

```
Person p = new Person();
List<Tuple2<String, Integer>> rows =
new QueryBuilder(datasource)
.from(p)
.select(p.name,
unchecked(Integer.class,
"util.count_children(id)"))
.list();
```

Listing A.4: Typesafe SQL query

A.2 Source Code

The following selected classes represent the generated expression builders from the DSL2JDT tool, which were transformed to TEX code with the Eclipse plugin Java2HTML, available at the following site `http://www.java2html.de/`. Links to the diagrams which correspond to the code are given also.

A.2.1 ProtocolObjectsExprBuilder.java

The corresponding Ecore diagram is located at 3.1.

```
1  package ProtocolObjects;
2
3  import java.lang.String;
4
5  public class ProtocolObjectsExprBuilder {
6    public static RequestMessageBeingBuilt requestMessage(){
7      return new RequestMessageBeingBuilt (
8          ProtocolObjectsFactory.eINSTANCE.
9                           createRequestMessage());
10   }
11   public static ResponseMessageBeingBuilt responseMessage() {
12     return new ResponseMessageBeingBuilt (
13         ProtocolObjectsFactory.eINSTANCE.
14                         createResponseMessage() );
15   }
16   public static ResponseBeingBuilt response() {
17     return new ResponseBeingBuilt (
18         ProtocolObjectsFactory.eINSTANCE.
19                           createResponse() );
20   }
21   public static RequestBeingBuilt request() {
22     return new RequestBeingBuilt (
23         ProtocolObjectsFactory.eINSTANCE.
24                        createRequest() );
25   }
26   public static KBRequestBeingBuilt kBRequest() {
```

[1] `http://martinfowler.com/dslwip/InternalOverview.html#FluentAndPush-buttonApis`

```
27      return new KBRequestBeingBuilt (
28          ProtocolObjectsFactory.eINSTANCE.
29                          createKBRequest() );
30    }
31    public static ErrorBeingBuilt error() {
32      return new ErrorBeingBuilt (
33          ProtocolObjectsFactory.eINSTANCE.
34                            createError() );
35    }
36    public static ConfirmationBeingBuilt confirmation() {
37      return new ConfirmationBeingBuilt (
38          ProtocolObjectsFactory.eINSTANCE.
39                        createConfirmation() );
40    }
41    public static KBResponseBeingBuilt kBResponse() {
42      return new KBResponseBeingBuilt (
43          ProtocolObjectsFactory.eINSTANCE.
44                           createKBResponse() );
45    }
46    public static SyntaxErrorBeingBuilt syntaxError() {
47      return new SyntaxErrorBeingBuilt (
48          ProtocolObjectsFactory.eINSTANCE.
49                            createSyntaxError() );
50    }
51    public static KBErrorBeingBuilt kBError() {
52      return new KBErrorBeingBuilt (
53          ProtocolObjectsFactory.eINSTANCE.
54                                createKBError() );
55    }
56    public static SemanticErrorBeingBuilt semanticError() {
57      return new SemanticErrorBeingBuilt (
58          ProtocolObjectsFactory.eINSTANCE.
59                          createSemanticError() );
60    }
61
62    public static class RequestMessageBeingBuilt  {
63      private final RequestMessage myExpr;
64      public RequestMessage toAST()
65      { return this.myExpr; }
66      RequestMessageBeingBuilt( RequestMessage  arg) {
67        this.myExpr = arg;
68      }
69
70      public RequestMessageBeingBuilt requests(
71          Request... items) {
72        this.myExpr.getRequests().clear();
73        this.myExpr.getRequests().addAll(
74            java.util.Arrays.asList(items) );
75        return this; }
76    }
77
78    public static class ResponseMessageBeingBuilt  {
```

```
79      private final ResponseMessage myExpr;
80      public ResponseMessage toAST()
81      { return this.myExpr; }
82      ResponseMessageBeingBuilt( ResponseMessage  arg) {
83        this.myExpr = arg;
84      }
85
86      public ResponseMessageBeingBuilt responses(
87          Response... items) {
88        this.myExpr.getResponses().clear();
89        this.myExpr.getResponses().addAll(
90            java.util.Arrays.asList(items) );
91        return this; }
92    }
93
94    public static class ResponseBeingBuilt  {
95      private final Response myExpr;
96      public Response toAST()
97      { return this.myExpr; }
98      ResponseBeingBuilt( Response  arg) {
99        this.myExpr = arg;
100     }
101   }
102
103   public static class RequestBeingBuilt  {
104     private final Request myExpr;
105     public Request toAST()
106     { return this.myExpr; }
107     RequestBeingBuilt( Request  arg) {
108       this.myExpr = arg;
109     }
110   }
111
112   public static class KBRequestBeingBuilt  {
113     private final KBRequest myExpr;
114     public KBRequest toAST()
115     { return this.myExpr; }
116     KBRequestBeingBuilt( KBRequest  arg) {
117       this.myExpr = arg;
118     }
119     public KBRequestBeingBuilt kb( String arg ){
120       this.myExpr.setKb(arg);
121       return this; }
122   }
123
124   public static class ErrorBeingBuilt  {
125     private final Error myExpr;
126     public Error toAST(){
127       return this.myExpr;
128     }
129     ErrorBeingBuilt( Error  arg){
130       this.myExpr = arg;
```

```
131     }
132     public ErrorBeingBuilt error( String arg ){
133       this.myExpr.setError(arg);
134       return this; }
135   }
136
137   public static class ConfirmationBeingBuilt  {
138     private final Confirmation myExpr;
139     public Confirmation toAST()
140     { return this.myExpr; }
141     ConfirmationBeingBuilt( Confirmation  arg) {
142       this.myExpr = arg;
143     }
144     public ConfirmationBeingBuilt warning  ( String arg )  {
145       this.myExpr.setWarning(arg);
146       return this; }
147   }
148
149   public static class KBResponseBeingBuilt  {
150     private final KBResponse myExpr;
151     public KBResponse toAST()
152     { return this.myExpr; }
153     KBResponseBeingBuilt( KBResponse  arg) {
154       this.myExpr = arg;
155     }
156     public KBResponseBeingBuilt warning  ( String arg )  {
157       this.myExpr.setWarning(arg);
158       return this; }
159   }
160
161   public static class SyntaxErrorBeingBuilt  {
162     private final SyntaxError myExpr;
163     public SyntaxError toAST() { return this.myExpr; }
164     SyntaxErrorBeingBuilt( SyntaxError  arg) {
165       this.myExpr = arg;
166     }
167     public SyntaxErrorBeingBuilt error  ( String arg )  {
168       this.myExpr.setError(arg);
169       return this; }
170   }
171
172   public static class KBErrorBeingBuilt  {
173     private final KBError myExpr;
174     public KBError toAST() {
175       return this.myExpr;
176     }
177     KBErrorBeingBuilt( KBError  arg) {
178       this.myExpr = arg;
179     }
180     public KBErrorBeingBuilt error  ( String arg )  {
181       this.myExpr.setError(arg);
182       return this; }
```

```
183   }
184
185   public static class SemanticErrorBeingBuilt {
186     private final SemanticError myExpr;
187     public SemanticError toAST() {
188       return this.myExpr;
189     }
190     SemanticErrorBeingBuilt( SemanticError  arg) {
191       this.myExpr = arg;
192     }
193     public SemanticErrorBeingBuilt error  ( String arg )  {
194       this.myExpr.setError(arg);
195       return this; }
196   }
197 }
```

A.2.2 RetrieveEntityExprBuilder.java

The following source code was generated from the retrieve-KB-Entities Ecore diagram 3.2.

```
1  package retrieveEntity;
2
3  import java.lang.String;
4
5  public class RetrieveEntityExprBuilder
6  {
7    public static KBRequestBeingBuilt kBRequest() {
8      return new KBRequestBeingBuilt (
9          RetrieveEntityFactory.eINSTANCE.
10                        createKBRequest() );
11   }
12   public static GetAllClassesBeingBuilt getAllClasses() {
13     return new GetAllClassesBeingBuilt (
14         RetrieveEntityFactory.eINSTANCE.
15                     createGetAllClasses() );
16   }
17   public static GetAllDataPropertiesBeingBuilt
18                             getAllDataProperties() {
19     return new GetAllDataPropertiesBeingBuilt (
20         RetrieveEntityFactory.eINSTANCE.
21                                createGetAllDataProperties() );
22   }
23   public static GetAllObjectPropertiesBeingBuilt
24                                 getAllObjectProperties()
25   {
26     return new GetAllObjectPropertiesBeingBuilt (
27         RetrieveEntityFactory.eINSTANCE.
28                                  createGetAllObjectProperties() );
29   }
30   public static GetAllIndividualsBeingBuilt getAllIndividuals() {
31     return new GetAllIndividualsBeingBuilt (
```

```
32          RetrieveEntityFactory.eINSTANCE.
33                                  createGetAllIndividuals() );
34    }
35    public static GetAllDatatypesBeingBuilt getAllDatatypes() {
36      return new GetAllDatatypesBeingBuilt (
37          RetrieveEntityFactory.eINSTANCE.
38                            createGetAllDatatypes() );
39    }
40    public static GetAllAnnotationPropertiesBeingBuilt
41                                    getAllAnnotationProperties()
42    {
43      return new GetAllAnnotationPropertiesBeingBuilt (
44          RetrieveEntityFactory.eINSTANCE.
45                                createGetAllAnnotationProperties() );
46    }
47    public static KBResponseBeingBuilt kBResponse() {
48      return new KBResponseBeingBuilt (
49          RetrieveEntityFactory.eINSTANCE.createKBResponse() );
50    }
51    public static SetOfOWLClassBeingBuilt setOfOWLClass() {
52      return new SetOfOWLClassBeingBuilt (
53          RetrieveEntityFactory.eINSTANCE.createSetOfOWLClass() );
54    }
55    public static SetOfAnnotationPropertyBeingBuilt
56                                setOfAnnotationProperty()
57    {
58      return new SetOfAnnotationPropertyBeingBuilt (
59          RetrieveEntityFactory.eINSTANCE.
60                               createSetOfAnnotationProperty() );
61    }
62    public static SetOfObjectPropertyBeingBuilt setOfObjectProperty() {
63      return new SetOfObjectPropertyBeingBuilt (
64          RetrieveEntityFactory.eINSTANCE.
65                                  createSetOfObjectProperty() );
66    }
67    public static SetOfDataPropertyBeingBuilt setOfDataProperty() {
68      return new SetOfDataPropertyBeingBuilt (
69          RetrieveEntityFactory.eINSTANCE.
70                                  createSetOfDataProperty() );
71    }
72    public static SetOfIndividualBeingBuilt setOfIndividual() {
73      return new SetOfIndividualBeingBuilt (
74          RetrieveEntityFactory.eINSTANCE.
75                                  createSetOfIndividual() );
76    }
77    public static SetOfDatatypeBeingBuilt setOfDatatype() {
78      return new SetOfDatatypeBeingBuilt (
79          RetrieveEntityFactory.eINSTANCE.
80                                  createSetOfDatatype() );
81    }
82    public static OWLClassBeingBuilt oWLClass() {
83      return new OWLClassBeingBuilt (
```

```
84          RetrieveEntityFactory.eINSTANCE.createOWLClass() );
85    }
86    public static OWLAnnotationPropertyBeingBuilt oWLAnnotationProperty()
87    {
88      return new OWLAnnotationPropertyBeingBuilt (
89          RetrieveEntityFactory.eINSTANCE.
90                                  createOWLAnnotationProperty() );
91    }
92    public static OWLObjectPropertyBeingBuilt oWLObjectProperty()
93    {
94      return new OWLObjectPropertyBeingBuilt (
95          RetrieveEntityFactory.eINSTANCE.
96                              createOWLObjectProperty() );
97    }
98    public static OWLIndividualBeingBuilt oWLIndividual() {
99      return new OWLIndividualBeingBuilt (
100         RetrieveEntityFactory.eINSTANCE.
101                                  createOWLIndividual() );
102   }
103   public static OWLDatatypeBeingBuilt oWLDatatype() {
104     return new OWLDatatypeBeingBuilt (
105         RetrieveEntityFactory.eINSTANCE.
106                                  createOWLDatatype() );
107   }
108   public static OWLDataPropertyBeingBuilt oWLDataProperty() {
109     return new OWLDataPropertyBeingBuilt (
110         RetrieveEntityFactory.eINSTANCE.
111                               createOWLDataProperty() );
112   }
113
114   public static class KBRequestBeingBuilt  {
115     private final KBRequest myExpr;
116     public KBRequest toAST() { return this.myExpr; }
117     KBRequestBeingBuilt( KBRequest  arg) {
118       this.myExpr = arg;
119     }
120     public KBRequestBeingBuilt kb  ( String arg )  {
121       this.myExpr.setKb(arg);
122       return this; }
123   }
124
125   public static class GetAllClassesBeingBuilt  {
126     private final GetAllClasses myExpr;
127     public GetAllClasses toAST() { return this.myExpr; }
128     GetAllClassesBeingBuilt( GetAllClasses  arg) {
129       this.myExpr = arg;
130     }
131     public GetAllClassesBeingBuilt kb  ( String arg )  {
132       this.myExpr.setKb(arg);
133       return this; }
134   }
135
```

```
136   public static class GetAllDataPropertiesBeingBuilt  {
137     private final GetAllDataProperties myExpr;
138     public GetAllDataProperties toAST() { return this.myExpr; }
139     GetAllDataPropertiesBeingBuilt( GetAllDataProperties  arg) {
140       this.myExpr = arg;
141     }
142     public GetAllDataPropertiesBeingBuilt kb( String arg ){
143       this.myExpr.setKb(arg);
144       return this; }
145   }
146
147   public static class GetAllObjectPropertiesBeingBuilt  {
148     private final GetAllObjectProperties myExpr;
149     public GetAllObjectProperties toAST()
150     { return this.myExpr; }
151     GetAllObjectPropertiesBeingBuilt(
152         GetAllObjectProperties  arg) {
153       this.myExpr = arg;
154     }
155     public GetAllObjectPropertiesBeingBuilt kb( String arg )  {
156       this.myExpr.setKb(arg);
157       return this; }
158   }
159
160   public static class GetAllIndividualsBeingBuilt  {
161     private final GetAllIndividuals myExpr;
162     public GetAllIndividuals toAST()
163     { return this.myExpr; }
164     GetAllIndividualsBeingBuilt( GetAllIndividuals  arg) {
165       this.myExpr = arg;
166     }
167     public GetAllIndividualsBeingBuilt kb  ( String arg )  {
168       this.myExpr.setKb(arg);
169       return this; }
170   }
171
172   public static class GetAllDatatypesBeingBuilt  {
173     private final GetAllDatatypes myExpr;
174     public GetAllDatatypes toAST() { return this.myExpr; }
175     GetAllDatatypesBeingBuilt( GetAllDatatypes  arg) {
176       this.myExpr = arg;
177     }
178     public GetAllDatatypesBeingBuilt kb( String arg )  {
179       this.myExpr.setKb(arg);
180       return this; }
181   }
182
183   public static class GetAllAnnotationPropertiesBeingBuilt  {
184     private final GetAllAnnotationProperties myExpr;
185     public GetAllAnnotationProperties toAST() { return this.myExpr; }
186     GetAllAnnotationPropertiesBeingBuilt(
187                   GetAllAnnotationProperties  arg)
```

```
188     {
189       this.myExpr = arg;
190     }
191     public GetAllAnnotationPropertiesBeingBuilt kb( String arg )
192     {
193       this.myExpr.setKb(arg);
194       return this;
195     }
196   }
197
198   public static class KBResponseBeingBuilt  {
199     private final KBResponse myExpr;
200     public KBResponse toAST() { return this.myExpr; }
201     KBResponseBeingBuilt( KBResponse  arg) {
202       this.myExpr = arg;
203     }
204   }
205
206   public static class SetOfOWLClassBeingBuilt  {
207     private final SetOfOWLClass myExpr;
208     public SetOfOWLClass toAST() { return this.myExpr; }
209     SetOfOWLClassBeingBuilt( SetOfOWLClass  arg) {
210       this.myExpr = arg;
211     }
212
213     public SetOfOWLClassBeingBuilt entities( OWLClass... items)
214     {
215       this.myExpr.getEntities().clear();
216       this.myExpr.getEntities().addAll(
217           java.util.Arrays.asList(items) );
218       return this; }
219   }
220
221   public static class SetOfAnnotationPropertyBeingBuilt
222   {
223     private final SetOfAnnotationProperty myExpr;
224     public SetOfAnnotationProperty toAST() { return this.myExpr; }
225     SetOfAnnotationPropertyBeingBuilt( SetOfAnnotationProperty  arg)
226     {
227       this.myExpr = arg;
228     }
229
230     public SetOfAnnotationPropertyBeingBuilt entities(
231         OWLAnnotationProperty... items) {
232       this.myExpr.getEntities().clear();
233       this.myExpr.getEntities().addAll(
234           java.util.Arrays.asList(items) );
235       return this; }
236   }
237
238   public static class SetOfObjectPropertyBeingBuilt  {
239     private final SetOfObjectProperty myExpr;
```

```
240     public SetOfObjectProperty toAST() { return this.myExpr; }
241     SetOfObjectPropertyBeingBuilt( SetOfObjectProperty  arg)
242     {
243       this.myExpr = arg;
244     }
245
246     public SetOfObjectPropertyBeingBuilt entities(
247         OWLObjectProperty... items)
248     {
249       this.myExpr.getEntities().clear();
250       this.myExpr.getEntities().addAll(
251           java.util.Arrays.asList(items) );
252       return this; }
253   }
254
255   public static class SetOfDataPropertyBeingBuilt  {
256     private final SetOfDataProperty myExpr;
257     public SetOfDataProperty toAST() { return this.myExpr; }
258     SetOfDataPropertyBeingBuilt( SetOfDataProperty  arg) {
259       this.myExpr = arg;
260     }
261
262     public SetOfDataPropertyBeingBuilt entities(
263         OWLDataProperty... items)
264     {
265       this.myExpr.getEntities().clear();
266       this.myExpr.getEntities().addAll(
267           java.util.Arrays.asList(items) );
268       return this; }
269   }
270
271   public static class SetOfIndividualBeingBuilt  {
272     private final SetOfIndividual myExpr;
273     public SetOfIndividual toAST() { return this.myExpr; }
274     SetOfIndividualBeingBuilt( SetOfIndividual  arg) {
275       this.myExpr = arg;
276     }
277
278     public SetOfIndividualBeingBuilt entities(
279         OWLIndividual... items)
280     {
281       this.myExpr.getEntities().clear();
282       this.myExpr.getEntities().addAll(
283           java.util.Arrays.asList(items) );
284       return this; }
285   }
286
287   public static class SetOfDatatypeBeingBuilt  {
288     private final SetOfDatatype myExpr;
289     public SetOfDatatype toAST() { return this.myExpr; }
290     SetOfDatatypeBeingBuilt( SetOfDatatype  arg) {
291       this.myExpr = arg;
```

```
292     }
293
294     public SetOfDatatypeBeingBuilt entities( OWLDatatype... items)
295     {
296       this.myExpr.getEntities().clear();
297       this.myExpr.getEntities().addAll(
298           java.util.Arrays.asList(items) );
299       return this; }
300   }
301
302   public static class OWLClassBeingBuilt  {
303     private final OWLClass myExpr;
304     public OWLClass toAST() { return this.myExpr; }
305     OWLClassBeingBuilt( OWLClass  arg) {
306       this.myExpr = arg;
307     }
308   }
309
310   public static class OWLAnnotationPropertyBeingBuilt  {
311     private final OWLAnnotationProperty myExpr;
312     public OWLAnnotationProperty toAST() { return this.myExpr; }
313     OWLAnnotationPropertyBeingBuilt( OWLAnnotationProperty  arg) {
314       this.myExpr = arg;
315     }
316   }
317
318   public static class OWLObjectPropertyBeingBuilt  {
319     private final OWLObjectProperty myExpr;
320     public OWLObjectProperty toAST() { return this.myExpr; }
321     OWLObjectPropertyBeingBuilt( OWLObjectProperty  arg) {
322       this.myExpr = arg;
323     }
324   }
325
326   public static class OWLIndividualBeingBuilt  {
327     private final OWLIndividual myExpr;
328     public OWLIndividual toAST() { return this.myExpr; }
329     OWLIndividualBeingBuilt( OWLIndividual  arg) {
330       this.myExpr = arg;
331     }
332   }
333
334   public static class OWLDatatypeBeingBuilt  {
335     private final OWLDatatype myExpr;
336     public OWLDatatype toAST() { return this.myExpr; }
337     OWLDatatypeBeingBuilt( OWLDatatype  arg) {
338       this.myExpr = arg;
339     }
340   }
341
342   public static class OWLDataPropertyBeingBuilt  {
343     private final OWLDataProperty myExpr;
```

```
344     public OWLDataProperty toAST() { return this.myExpr; }
345     OWLDataPropertyBeingBuilt( OWLDataProperty  arg) {
346       this.myExpr = arg;
347     }
348   }
349 }
```

A.2.3 Constructing OWLlink messages

The building of the OWLlink messages and the subsequent serialization to XML are documented in the following source code file. The answer received from the RacerPro reasoner in the form of a XML file is deserialized to OWLlink message.

```
1  package shorthand.Test;
2
3
4  import java.io.BufferedInputStream;
5  import java.io.BufferedOutputStream;
6  import java.io.FileInputStream;
7  import java.io.FileNotFoundException;
8  import java.io.FileOutputStream;
9  import java.io.IOException;
10 import java.io.InputStream;
11 import java.io.OutputStream;
12 import java.io.OutputStreamWriter;
13 import java.io.UnsupportedEncodingException;
14
15 import com.thoughtworks.xstream.XStream;
16
17
18 import ProtocolObjects.Confirmation;
19 import ProtocolObjects.Error;
20 import ProtocolObjects.KBRequest;
21 import ProtocolObjects.KBResponse;
22 import ProtocolObjects.ProtocolObjectsExprBuilder;
23 import ProtocolObjects.RequestMessage;
24 import ProtocolObjects.ResponseMessage;
25 import ProtocolObjects.SemanticError;
26 import ProtocolObjects.SyntaxError;
27 import ProtocolObjects.impl.ConfirmationImpl;
28 import ProtocolObjects.impl.ErrorImpl;
29 import ProtocolObjects.impl.KBRequestImpl;
30 import ProtocolObjects.impl.KBResponseImpl;
31 import ProtocolObjects.impl.RequestMessageImpl;
32 import ProtocolObjects.impl.ResponseMessageImpl;
33 import ProtocolObjects.impl.SemanticErrorImpl;
34
35 public class ProtocolCore {
36
37   public static RequestMessage RequestExample(){
38
```

```
39      RequestMessage  message1 = null;
40
41      KBRequest one = Q.kBRequest().kb("KB_1").toAST();
42
43      KBRequest two = Q.kBRequest().kb("KB_2").toAST();
44
45      message1 = Q.requestMessage().requests(one, two).toAST();
46
47      return message1;
48    }
49
50
51    public static ResponseMessage ResponseExample(){
52      ResponseMessage response1 = null;
53      Error positiveAnswer = Q.
54                              error().
55                              error("No such class found!").
56                              toAST();
57      Confirmation confirm = Q.confirmation().toAST();
58      KBResponse back = Q.
59                        kBResponse().
60                        warning("Say something!").
61                        toAST();
62      SemanticError second = Q.
63                              semanticError().
64                              error("A semantic error in the request was found!").
65                              toAST();
66      response1 = Q.
67                  responseMessage().
68                  responses(back, second, confirm,positiveAnswer).
69                  toAST();
70      return response1;
71    }
72
73    public static class Q extends ProtocolObjectsExprBuilder{
74
75    }
76
77     public static void main(String[] args){
78
79      //serialize the object, get an xstream object
80      XStream xs = new XStream();
81          // http://markmail.org/message/jjzeohwbng2lph6e
82          //link explanation of the following code
83  //    xs.registerConverter(
84  //                new ReflectionConverter(xs.getMapper(),
85  //    xs.getReflectionProvider()),
86  //                            XStream.PRIORITY_LOW);
87      // work with annotations
88  //    xs.processAnnotations(ResponseMessageImpl.class);
89
90      ResponseMessage response1 = null;
```

```
91
92      ResponseMessage response2 = null;
93
94          // generate a request message
95      // corresponds to the xml file requestMessage
96      RequestMessage  message1 = null;
97
98      KBRequest one = Q.kBRequest().kb("KB_1").toAST();
99
100     KBRequest two = Q.kBRequest().kb("KB_2").toAST();
101
102 //    xs.alias("KBRequest", KBRequestImpl.class);
103
104 //    xs.useAttributeFor( KBRequestImpl.class, "kb");
105
106 //    xs.aliasField("kb", KBRequestImpl.class, "KBRequest");
107
108 //    xs.addImplicitCollection( RequestMessageImpl.class, "requests");
109
110 //    xs.addImplicitCollection( ResponseMessageImpl.class, "responses");
111
112 //    xs.omitField(ResponseMessageImpl.class, "serialization");
113
114
115
116     message1 = Q.requestMessage().requests(one, two).toAST();
117
118 //    xs.omitField(ProtocolObjectsPackage.class, "int");
119
120 //  <Register-converters-section>
121     xs.registerConverter(new RequestMessageConverter());
122     xs.registerConverter(new KBRequestConverter());
123     xs.registerConverter(new ResponseMessageConverter());
124     xs.registerConverter(new SemanticErrorConverter());
125     xs.registerConverter(new KBResponseConverter());
126     xs.registerConverter(new SyntaxErrorConverter());
127     xs.registerConverter(new SetOfIndividualsConverter());
128     xs.registerConverter(new SetOfClassesConverter());
129     xs.registerConverter(new OWLClassConverter());
130     xs.registerConverter(new OWLIndividualConverter());
131 //  </Register-converters-section>
132     SyntaxError negativeAnswer2 = Q.
133                                   syntaxError().
134                                   error("No such match found!").
135                                   toAST();
136
137     SyntaxError negativeAnswer = Q.
138                                  syntaxError().
139                                  error("No such class found!").
140                                  toAST();
141
142     Confirmation confirm = Q.confirmation().toAST();
```

```
143
144     KBResponse kbResponse = Q.
145                             kBResponse().
146                             warning("Say something!").
147                             toAST();
148
149     SemanticError semanticError = Q.
150                                   semanticError().
151                                   error("A semantic error in the request was found!").
152                                   toAST();
153
154     response1 = Q.
155                 responseMessage().
156                 responses( kbResponse, semanticError, confirm, negativeAnswer).
157                 toAST();
158
159     response2 = Q.
160                 responseMessage().
161                 responses(negativeAnswer2).
162                 toAST();
163
164     xs.alias("ResponseMessage", ResponseMessageImpl.class);
165
166     xs.alias("RequestMessage", RequestMessageImpl.class);
167
168     xs.alias("KBRequest", KBRequestImpl.class);
169
170 //    xs.useAttributeFor(ConfirmationImpl.class, "warning");
171
172 //    xs.aliasField("KBResponse", KBResponseImpl.class, "warning");
173
174 //    xs.alias("KBResponse", KBResponseImpl.class);
175
176 //    xs.alias("SemanticError", SemanticErrorImpl.class);
177
178 //    xs.useAttributeFor(ErrorImpl.class, "error");
179
180 //    xs.aliasField("SemanticError", SemanticErrorImpl.class, "error");
181
182 //    xs.alias("OK", ConfirmationImpl.class);
183
184 //    xs.alias("Error", ErrorImpl.class);
185
186 //    xs.addImplicitCollection(ResponseMessageImpl.class, "responses");
187
188 //    xs.omitField(BasicEList.class, "size");
189
190 //    xs.omitField(BasicEList.class, "default");
191 //
192 //    xs.omitField(EObjectImpl.class, "eFlags");
193 //
194 //    xs.omitField(EObjectImpl.class, "eContainerFeatureID");
```

```
195 //
196 //    xs.omitField(EObjectImpl.class, "eContainer");
197
198
199 //    xs.omitField(ResponseMessageImpl.class, "default");
200
201 //    xs.omitField(List.class, "int");
202
203 //    xs.omitField(BasicEList.class, "int");
204
205     try{
206
207       FileOutputStream fs1 =
208         new FileOutputStream("c:/temp/responseMessage.xml");
209
210       xs.toXML(response1, fs1);
211
212       OutputStream out = null;
213       OutputStreamWriter writer = null;
214       try{
215         out = new BufferedOutputStream(
216             new FileOutputStream("c:/temp/tx.xml"));
217         writer = new OutputStreamWriter(out,"UTF-8");
218         writer.write("<?xml version='1.0' encoding='UTF-8'?>\n");
219       }
220       catch (FileNotFoundException e1)
221       {
222         e1.printStackTrace();
223       }
224       catch (UnsupportedEncodingException e2) {
225
226         e2.printStackTrace();
227       } catch (IOException e) {
228         // TODO Auto-generated catch block
229         e.printStackTrace();
230       }
231
232
233             FileOutputStream fs2 =
234               new FileOutputStream("c:/temp/kbRequest.xml");
235             xs.toXML(one, fs2);
236
237             FileOutputStream fs3 =
238               new FileOutputStream("c:/temp/requestMessage.xml");
239             xs.toXML(message1, fs3);
240
241             FileOutputStream fs4 =
242               new FileOutputStream("c:/temp/responseMessage2.xml");
243             xs.toXML(response2, fs4);
244
245             xs.toXML(message1, writer);
246
```

```
247             ResponseMessage fromXml =
248             (ResponseMessage) xs.fromXML(
249                 new FileInputStream("c:/temp/answerW.xml"));
250
251             FileOutputStream fs5 =
252               new FileOutputStream("c:/temp/deserialize.xml");
253
254             xs.toXML(fromXml, fs5);
255
256     }
257     catch (FileNotFoundException exception1)
258     {
259       exception1.printStackTrace();
260     }
261
262   }
263 }
```

A.2.4 ResponseMessageConverter.java

The specially crafted *marshal()* and *unmarshal()* methods implemented to deal with the XML serialization and deserialization of OWLlink response message.

```
1  package shorthand.Test;
2
3  import java.util.Iterator;
4
5  import org.eclipse.emf.ecore.util.EObjectContainmentEList;
6
7  import retrieveEntity.SetOfIndividual;
8  import retrieveEntity.SetOfOWLClass;
9  import retrieveEntity.impl.SetOfIndividualImpl;
10 import retrieveEntity.impl.SetOfOWLClassImpl;
11 import ProtocolObjects.Response;
12 import ProtocolObjects.ResponseMessage;
13 import ProtocolObjects.impl.ConfirmationImpl;
14 import ProtocolObjects.impl.KBResponseImpl;
15 import ProtocolObjects.impl.ResponseMessageImpl;
16 import ProtocolObjects.impl.SemanticErrorImpl;
17 import ProtocolObjects.impl.SyntaxErrorImpl;
18
19 import com.thoughtworks.xstream.converters.Converter;
20 import com.thoughtworks.xstream.converters.MarshallingContext;
21 import com.thoughtworks.xstream.converters.UnmarshallingContext;
22 import com.thoughtworks.xstream.io.HierarchicalStreamReader;
23 import com.thoughtworks.xstream.io.HierarchicalStreamWriter;
24
25 public class ResponseMessageConverter implements Converter {
26
27   @Override
28   public void marshal(Object arg0, HierarchicalStreamWriter arg1,
29       MarshallingContext arg2) {
```

```
30      ResponseMessageImpl message = (ResponseMessageImpl) arg0;
31      arg1.addAttribute("xmlns", "http://www.owllink.org/owllink-xml#");
32          arg1.addAttribute("xmlns:owl", "http://www.w3.org/2002/07/owl#");
33
34          EObjectContainmentEList<ProtocolObjects.Response> listResp =
35            new EObjectContainmentEList<ProtocolObjects.Response>
36            (Response.class, message, 0);
37
38
39          listResp = (EObjectContainmentEList<Response>) message.getResponses();
40          for(Iterator<Response> it = listResp.iterator(); it.hasNext();)
41          {
42            Response toHandle = it.next();
43            if(toHandle.getClass().equals(SemanticErrorImpl.class))
44            {
45              arg1.startNode("SemanticError");
46              arg2.convertAnother(toHandle, new SemanticErrorConverter());
47              arg1.endNode();
48            }
49            else if(toHandle.getClass().equals(KBResponseImpl.class))
50            {
51              arg1.startNode("KBResponse");
52              arg2.convertAnother(toHandle, new KBResponseConverter());
53              arg1.endNode();
54            }
55            else if(toHandle.getClass().equals(ConfirmationImpl.class))
56            {
57              arg1.startNode("OK");
58              arg1.endNode();
59            }
60            else if(toHandle.getClass().equals(SyntaxErrorImpl.class))
61            {
62              arg1.startNode("SyntaxError");
63              arg2.convertAnother(toHandle, new SyntaxErrorConverter());
64              arg1.endNode();
65            }
66            else if(toHandle.getClass().equals(SetOfIndividualImpl.class))
67            {
68              arg1.startNode("SetOfIndividuals");
69              arg2.convertAnother(toHandle, new SetOfIndividualsConverter());
70              arg1.endNode();
71            }
72            else if(toHandle.getClass().equals(SetOfOWLClassImpl.class))
73            {
74              arg1.startNode("SetOfClasses");
75              arg2.convertAnother(toHandle, new SetOfClassesConverter());
76              arg1.endNode();
77            }
78            else
79            {
80              arg1.startNode("Unknown");
81              arg1.endNode();
```

```
82            }
83          }
84
85    }
86
87    @Override
88    public Object unmarshal(HierarchicalStreamReader reader,
89        UnmarshallingContext context) {
90      ResponseMessage response = null;
91      ResponseMessageImpl needIt = new ResponseMessageImpl();
92      SetOfOWLClass setClass = null;
93      SetOfIndividual setIndividuals = null;
94      EObjectContainmentEList<ProtocolObjects.Response> listResp =
95            new EObjectContainmentEList<ProtocolObjects.Response>
96      (Response.class, needIt, 0);
97      while(reader.hasMoreChildren())
98      {
99        reader.moveDown();
100
101       if("SetOfClasses".equals(reader.getNodeName()))
102       {
103         setClass = (SetOfOWLClass)context.convertAnother(response,
104             SetOfOWLClassImpl.class,
105             new SetOfClassesConverter());
106         listResp.add((Response) setClass);
107       }
108       else if("SetOfIndividuals".equals(reader.getNodeName()))
109       {
110         setIndividuals = (SetOfIndividual)context.
111                 convertAnother(response,
112             SetOfIndividualImpl.class,
113             new SetOfIndividualsConverter());
114         listResp.add((Response) setIndividuals);
115
116       }
117 //      if(setClass != null && setIndividuals!= null)
118 //      {
119 //        response = shorthand.Test.ProtocolCore.Q.
120 //        responseMessage().
121 //        responses((Response)setIndividuals, (Response)setClass).
122 //        toAST();
123 //      }
124       reader.moveUp();
125     }
126     response = shorthand.Test.ProtocolCore.Q.
127     responseMessage().
128     responses((Response)setIndividuals, (Response)setClass).
129     toAST();
130     return response;
131   }
132   @SuppressWarnings("unchecked")
133   @Override
```

```
134   public boolean canConvert(Class classToConvert) {
135     return classToConvert.equals(ResponseMessageImpl.class);
136   }
137 }
```

A.2.5 OWLIndividualConverter.java

The specially crafted *marshal()* and *unmarshal()* methods implemented to deal with the XML serialization and deserialization of JAVA OWLlink individual objects.

```
1  package shorthand.Test;
2
3  import retrieveEntity.OWLIndividual;
4  import retrieveEntity.impl.OWLIndividualImpl;
5
6  import com.thoughtworks.xstream.converters.Converter;
7  import com.thoughtworks.xstream.converters.MarshallingContext;
8  import com.thoughtworks.xstream.converters.UnmarshallingContext;
9  import com.thoughtworks.xstream.io.HierarchicalStreamReader;
10 import com.thoughtworks.xstream.io.HierarchicalStreamWriter;
11
12 public class OWLIndividualConverter implements Converter {
13
14   @Override
15   public void marshal(Object arg0, HierarchicalStreamWriter arg1,
16       MarshallingContext arg2) {
17     OWLIndividualImpl indi = (OWLIndividualImpl) arg0;
18     arg1.addAttribute("URI", indi.getKb());
19
20   }
21
22   @Override
23   public Object unmarshal(HierarchicalStreamReader reader,
24       UnmarshallingContext context) {
25     OWLIndividual indi = shorthand.SameOld.C.oWLIndividual().toAST();
26     indi.setKb(reader.getAttribute("URI"));
27     return indi;
28   }
29   @SuppressWarnings("unchecked")
30   @Override
31   public boolean canConvert(Class classToConvert) {
32     return classToConvert.equals(OWLIndividualImpl.class);
33   }
34
35 }
```

A.2.6 SetOfIndividualsConverter.java

The specially crafted *marshal()* and *unmarshal()* methods implemented to deal with the XML serialization and deserialization of JAVA OWLlink set of individuals objects.

```
1  package shorthand.Test;
2
3  import java.util.Iterator;
4
5  import org.eclipse.emf.ecore.util.EObjectContainmentEList;
6
7  import retrieveEntity.OWLIndividual;
8  import retrieveEntity.SetOfIndividual;
9  import retrieveEntity.impl.OWLIndividualImpl;
10 import retrieveEntity.impl.SetOfIndividualImpl;
11
12
13 import com.thoughtworks.xstream.converters.Converter;
14 import com.thoughtworks.xstream.converters.MarshallingContext;
15 import com.thoughtworks.xstream.converters.UnmarshallingContext;
16 import com.thoughtworks.xstream.io.HierarchicalStreamReader;
17 import com.thoughtworks.xstream.io.HierarchicalStreamWriter;
18
19 public class SetOfIndividualsConverter implements Converter {
20
21   @Override
22   public void marshal(Object arg0, HierarchicalStreamWriter arg1,
23       MarshallingContext arg2) {
24     SetOfIndividualImpl setToHandle = (SetOfIndividualImpl) arg0;
25
26     EObjectContainmentEList<retrieveEntity.OWLIndividual> listIndividuals =
27       new EObjectContainmentEList<OWLIndividual>(OWLIndividual.class, setToHandle, 0);
28
29     listIndividuals = (EObjectContainmentEList<OWLIndividual>) setToHandle.getEntities();
30
31     for(Iterator<OWLIndividual> it = listIndividuals.iterator(); it.hasNext();)
32     {
33       OWLIndividual toHandle = it.next();
34       if(toHandle.getClass().equals(OWLIndividualImpl.class))
35           {
36             arg1.startNode("Individual");
37             arg2.convertAnother(toHandle, new OWLIndividualConverter());
38             arg1.endNode();
39           }
40           else
41           {
42             arg1.startNode("Unknown");
43             arg1.endNode();
44           }
45     }
46   }
47
48   @Override
49   public Object unmarshal(HierarchicalStreamReader reader,
50       UnmarshallingContext context) {
51     SetOfIndividual setIndi = shorthand.SameOld.C.setOfIndividual().toAST();
52     SetOfIndividual doINeedIt = null;
```

```
53      while(reader.hasMoreChildren())
54      {
55        reader.moveDown();
56        if("owl:Individual".equals(reader.getNodeName()))
57        {
58          OWLIndividual newIndi = (OWLIndividual) context.convertAnother(doINeedIt,
59              OWLIndividualImpl.class,
60              new OWLIndividualConverter());
61
62          setIndi.getEntities().add(newIndi);
63        }
64        reader.moveUp();
65      }
66
67      return setIndi;
68    }
69    @SuppressWarnings("unchecked")
70    @Override
71    public boolean canConvert(Class classToConvert) {
72      return classToConvert.equals(SetOfIndividualImpl.class);
73    }
74
75  }
```

A.3 List of Acronyms

4GL 4^{th} Generation Language

API Application Programming Interface

ASDL Abstract Syntax Definition Language

AST Abstract Syntax Tree

DIG DL Implementation Group

DL Description Logics

DSL Domain-Specific Language

EDSL Embedded Domain-Specific Language

HTTP Hypertext Transport Protocol

IDE Integrated Development Environment

JDT Java Development Tools

J2SE Java^{TM} 2 Standard Edition

KB Knowledge Base

LSP Liskov Substitution Principle

OWL Web Ontology Language

OWLlink DIG 2.0

SMTP Simple Mail Transport Protocol

SQL Structured Query Language

TCL Tools Command Language

UML Unified Modeling Language

UNA Unique Name Assumption

URI Uniform Resource Identifier

W3C World Wide Web Consortium

XML Extensible Markup Language

Bibliography

[Ben86] Jon Bentley. Programming pearls: little languages. *Commun. ACM*, 29(8):711–721, August 1986. http://dx.doi.org/10.1145/6424.315691.

[Cle88] J. C. Cleaveland. Building application generators. *IEEE Software*, pages 25–33, July 1988.

[Fow09a] Martin Fowler. Domain specific languages. http://martinfowler.com/dslwip/index.html, 2009.

[Fow09b] Martin Fowler. Dsls idioms explained. http://martinfowler.com/dslwip/InternalOverview.html#FluentAndPush-buttonApis, 2009.

[Gar08] Miguel Garcia. Automating the embedding of Domain Specific Languages in Eclipse JDT, Eclipse Technical Article, 2008. http://www.eclipse.org/articles/Article-AutomatingDSLEmbeddings/index.html.

[Hud96] Paul Hudak. Building domain-specific embedded languages. *ACM Comp. Surv.*, June 1996.

[KG09a] Racer Systems GmbH & Co. KG. Native libraries support. http://www.racer-systems.com/products/download/nativelibraries.phtml, 2009.

[KG09b] Racer Systems GmbH & Co. KG. Renamed Abox and Concept Expression Reasoner RacerPro website. http://www.racer-systems.com/products/racerpro/index.phtml, 2009.

[KR08] Jevgeni Kabanov and Rein Raudjärv. Embedded Typesafe Domain Specific Languages for Java. *Principles and Practice of Programming in Java*, 2008.

[LLN08a] Thorsten Liebig, Marko Luther, and Olaf Noppens. *OWLlink:Structural Specification*, October 2008. Working draft.

[LLN+08b] Thorsten Liebig, Marko Luther, Olaf Noppens, Mariano Rodriguez, Diego Calvanese, Michael Wessel, Ralf Möller, Matthew Horridge, Sean Bechhofer, Dmitry Tsarkov, and Evren Sirin. OWLlink: DIG for OWL 2. *OWLED 2008 Workshop*, 2008. http://www.owllink.org/publications/owllink-OWLED08.pdf.

[MPSH08] Boris Motik, Peter Patel-Schneider, and Ian Horrocks. *OWL 2 Web Ontology Language:Structural Specification and Functional-Style Syntax*, April 2008. W3C Working draft,http://www.w3.org/TR/2008/WD-owl2-syntax-20080411/.

[oM09] University of Manchester. The OWL API. `http://owlapi.sourceforge.net/index.html`, 2009.

[Pri08] Rakesh Prithiviraj. IDE customization to support language embeddings. Master's thesis, University of Technology Hamburg-Harburg, 2008.

[vDKV00] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages. *n.a.*, November 2000. `http://ftp.cwi.nl/CWIreports/SEN/SEN-R0032.pdf`.