A Pluggable Architectural Model and a Formally Specified Programming Language Independent API for an Ontological Knowledge Base Server

Alexander Paar, Jürgen Reuter, Jaron Schaeffer

Institute for Program Structures and Data Organization
Universität Karlsruhe (TH)
Am Fasanengarten 5, 76128 Karlsruhe, Germany

`alexpaar@acm.org, reuter@ipd.uka.de, schaeffer@ipd.uka.de`

Abstract

Recently, ontology engineering has become ever more important when it comes to conceptualize knowledge. However, writing software applications that operate on ontological knowledge still suffers from a lack of connectivity provided by available ontology management systems. Interfaces of ontology management systems are either based on error prone programming language agnostic remoting protocols or they are restricted to one particular programming language. We implemented an ontological Knowledge Base Server, which can expose the functionality of arbitrary off-the-shelf ontology management systems via arbitrary remoting protocols. Based on XML Schema Definition, we defined a full-fledged API for processing OWL ontologies. Client access code can be generated automatically for virtually any object oriented programming language. Using Description Logics terminology, the Knowledge Base Server API was formally specified, such that it could be used to validate implementations based on three different adapted ontology management systems.

Keywords: Web Ontology Language (OWL), Description Logics, Ontology Management.

1 Introduction

In recent years, Semantic Web technologies like RDF(S) (RDF), DAML+OIL (DAML+OIL), and their common Description Logics (DL) (Baader, Calvanese, McGuiness, Nardi, and Patel-Schneider 2003) based successor OWL (OWL) have paved the way for standardized formal conceptualizations of all kinds of knowledge. Numerous ontologies have been developed to conceptualize a plethora of domains of discourse (Ontology Library). Since corporations from all sectors have braced to define company specific knowledge using Semantic Web technologies, ontology engineering has become a business model for a number of companies.

As the underlying standards have matured, tools for ontology engineering have emerged both in commercial as well as in academic fields. Knowledge acquisition systems like Protégé (Protégé) make it particularly easy to construct domain ontologies and to enter data. Ontology management systems like the IBM Semantic Network Base (SNOBASE) or HP Labs' Jena (Jena) can be used for loading ontologies from files and via the Internet and for creating, modifying, querying, and storing ontologies. Inference engines like RACER (RACER) provide support for query answering. There is an incessantly growing set of tools, projects, and applications for ontology languages like OWL. However, processing ontological information programmatically is still laborious and error prone. From our experience, this is caused by two main problems.

Firstly, there are no formal specifications that fully define the semantics of ontology management APIs. This is particularly problematic since typical interface methods (e.g. `listSubclasses`, `addIndividual`) are closely related to the semantics of the formal foundations of ontology languages (e.g. Description Logics).

Secondly, existing off-the-shelf ontology management systems only provide limited connectivity with respect to native support for programming languages and remoting protocols. Hence, it is particularly difficult to use ontology management systems remotely or along with a variety of different programming languages (i.e. in heterogeneous distributed computing environments). More gravely, it may be unfeasible to replace an ontology management system by alternative products without altering significant parts of client code.

This paper describes a pluggable architectural model for an ontological Knowledge Base Server, which can expose the functionality of arbitrary off-the-shelf ontology management systems via a well defined API. XML Schema Definition (XSD) data types and Description Logics terminology were used to formally specify the result sets and side effects of each method.

The architectural model of our Knowledge Base Server facilitates adding and replacing hosts for remoting protocols and replacing the access code for arbitrary off-the-shelf ontology management systems dynamically at runtime. Different implementations of the Knowledge Base Server API can automatically be tested for coherence to the specification using a JUnit (JUnit) based testing framework. In order to make it particularly easy for programmers to use the exposed API, we developed a code generation tool, which automatically generates client access code for object oriented programming languages.

The ontological Knowledge Base Server along with the code generation technologies as described in this paper is actively used in the CHIL research project. This is why it will be referred to as CHIL Knowledge Base Server in the rest of this paper. The CHIL research project aims to introduce computers into a loop of humans interacting with humans, rather than condemning a human to operate in a loop of computers. In order to implement unobtrusive user friendly services, a semantic middleware is being developed that fusions information provided by so called perceptual components in meaningful ways. Each perceptual component (e.g. image and speech recognizers, body trackers, etc.) contributes to the common domain of discourse. The Web Ontology Language (OWL) is being used to replace previous domain models based on particular programming languages. The CHIL Knowledge Base Server is used as the backend of an extensive semantic middleware (Pandis, Soldatos, Paar, Reuter, Carras, and Polymenakos 2005).

Whereas there are several approaches that aimed to provide programming language independent APIs for processing ontological knowledge bases, to the best of our knowledge, this paper presents the first work that combines the following three features.

- The CHIL Knowledge Base server API is solely based on XML Schema Definition primitive data types and it is remotely accessible by virtually every programming language capable of parsing strings and of TCP socket communication.
- Description Logics terminology was used to formally specify the result sets and side effects of each interface method.
- A JUnit based testing framework was developed for automatically validating implementations of the CHIL Knowledge Base Server API for coherence to the specification.

The outline of this paper is as follows. Section 2 will give an overview of the connectivity capabilities and API specifications of some of the most widely used ontology management systems. Section 3 will elucidate the architectural model and the implementation of the CHIL Knowledge Base Server. Section 4 will describe the design of the CHIL Knowledge Base Server API and how this API is remotely accessible via several different ports. Examples of formal specifications of CHIL Knowledge Base Server interface methods and a testing framework, which can be used to automatically validate implementations of the CHIL Knowledge Base Server API, will be introduced in Section 5. A conclusion and an overview of ongoing- and future work will be given in Section 6 followed by acknowledgements in Section 7.

2 Related Work

This section gives an overview of some widely used ontology management systems and reasoning engines along with a bird's eye view of some generic interface specifications for DL systems and related attempts to improve the remoting capabilities of such APIs.

2.1 Off-the-shelf Ontology Management Systems

This subsection gives an overview of existing off-the-shelf ontology management systems that can be used to manage RDF(S) data. In particular, the following three dimensions are considered. Firstly, remoting capabilities are assessed based on the number of supported remoting protocols (e.g. Java RMI, SOAP, and CORBA). Secondly, the extent of native support for programming languages such as Java or C++ is considered. Thirdly, the way how API specifications were devised is listed.

KAON 2 (KAON) is an open source ontology management infrastructure targeted for business applications. It ships as a Java library file. The core of the KAON 2 Java library are two APIs for RDF and the KAON ontology language. These APIs are represented by Java interfaces for which several implementations exist. Remote access is supported but limited to the Java programming language. Moreover, additional application server software is required, which may be impractical in practice. KAON 2 comes with plain text specifications of its RDF and KAON ontology language APIs.

HP Labs' Jena 2 (Jena) provides an ontological framework for the Java language environment. The internal representation of ontological data in Jena is tightly bound to the RDF model of triples. Originally designed for DAML+OIL, but later adopted to OWL, Jena 2 ships with a layered API. On the upper layers, it offers a unified view onto the features of the DAML+OIL and OWL languages, while providing access to specific ontology language dependent constructs via specific ontology models. Jena 2 makes it possible to configure and to replace the underlying reasoning engine, which is why there are only informal specifications of the exposed Jena 2 API. Remote access is not supported.

Ont. Mgmt. System.	**Remoting support**	**Programming Languages**	**API Specification**
KAON 2	Limited to Java	Java	Java Docs
Jena 2	No	Java	Java Docs
Snobase	No	Java	Java Docs
Protégé	No	Java	Java Docs
Sesame	HTTP	Java, Python	Java Docs
RACER	HTTP, Sockets	Java, Lisp	DIG

Table 1: Off-the-shelf ontology management systems

The IBM Ontology Management System (also known as SNOBASE, for Semantic Network Ontology Base) (SNOBASE) is a Java framework that provides a mechanism for querying ontologies and an easy-to-use programming interface for interacting with vocabularies of standard ontology specification languages such as OWL. Applications can query against the created ontology models and the inference engine deduces the answers and returns result sets similar to JDBC (Java Data Base Connectivity) result sets (JDBC). In theory, Java based remote access is possible but not available yet.

Stanford University School of Medicine's Protégé (Protégé) is an interactive Java application with a GUI that focuses on creating and editing ontologies. Having started as a project before OWL was available; it was designed to support a variety of different ontology languages. For ontology management Protégé focuses on user input through the graphical user interface. It also supports an API for plug-ins that essentially can be used as a management API. However, there is no support for remote access and the Protégé OWL API is specified only by example.

Sesame (Sesame) is an open source RDF database with support for RDF(S) inferencing and querying. It can be deployed on top of a variety of storage systems and offers a significant number of wrappers that facilitate HTTP based access to the Sesame system for a number of programming languages. However, the semantics of the Sesame API for processing OWL data are defined by third party extensions, which up to now only implement fragments of the OWL specification.

RACER (RACER) is a Semantic Web inference engine for query answering over RDF documents, and, with respect to specified RDF(S)/DAML ontologies, registering permanent queries. RACER implements a Description Logics reasoning system with support for TBoxes with generalized concept inclusions, ABoxes, and concrete domains. It supports native access from Java and Lisp and implements the DIG protocol (Bechhofer 2002) via XML-over-HTTP.

2.2 Knowledge Base Interface Specifications

The DIG protocol, which is a simple API for a general Description Logics system, is one representative of a class of interface definitions that consist of simple mechanisms to tell and ask DL knowledge bases. These mechanisms follow foundational aspects that have been well-studied over time (Levesque 1984). Many previous frame-oriented knowledge representation systems such as the Generic Frame Protocol (Chaudhri, Farquhar, Fikes, Karp, and Rice 1997) and OKBC (Open Knowledge Base Connectivity) (Chaudhri, Farquhar, Fikes, and Karp 1998) also embody such distinctions.

Although well defined, the DIG specification merely defines an XML schema that has to be used along with HTTP as the underlying communication protocol. There is no specific support for a particular programming language. In contrast, the KRSS specification (Patel-Schneider and Swartout 1993), which is an earlier approach to define a number of tell- and ask operations that a DL system should implement, was tightly bound to the LISP (Graham 1995) syntax, which may not be adequate for programmers who prefer other languages such as Java or C#.

In addition to RACER, the FaCT reasoner (Horrocks 1998, Horrocks 1999) from the University of Manchester is another implementation of the DIG 1.0 interface specification, which also requires further application server software.

Bechhofer *et al.* proposed a CORBA interface to the FaCT system (Bechhofer, Horrocks, Patel-Schneider, and Tessaris 1999). Beyond the fact that CORBA may not be an appropriate remoting technology in today's service oriented- and XML based computing environments, Bechhofer *et al.* note that "the CORBA IDL does not support the definition of the kinds of recursive data types that may be required for the representation of DL concepts and roles". This is why an XML based workaround was devised to pass ontological concepts and roles as single data items. Previous approaches to augment DL knowledge base interfaces with remoting capabilities include the wines- (Brachman, McGuinness, Patel-Schneider, Resnick, and Borgida 1991) and stereo (McGuiness, Resnick, and Isbell 1995) configuration demonstration systems.

Common to all mentioned DL knowledge base interface specifications is the lack of support for arbitrary state-of-the-art remoting protocols and adequate error- and exception handling. In particular, there are no detailed error messages passed to clients in case invalid requests are passed to the ontology management system.

3 CHIL Knowledge Base Server Architectural Model

The CHIL Knowledge Base Server is an adapter written in Java for off-the-shelf ontology management systems that implements a formally specified and well defined API to client components that may be written in a variety of different programming languages. Moreover, it supports dynamic replacements of adapted reasoners and ontology management systems.

3.1 Technical Requirements

For the CHIL Knowledge Base Server, the following most crucial requirements had to be met. Since the server is targeted for a distributed system, it must be accessible both locally and remotely through a single interface. Since client components in the CHIL research project may be written in a variety of different languages (e.g. Java, C#, C++, Python, Tcl/Tk), the remote interface must be programming language independent. Data representation must be architecture independent, such that mixed use of architectures with little-endian and big-endian byte order does not lead to interoperability issues. The CHIL Knowledge Base Server must be capable of handling multiple, potentially competing incoming requests in parallel without corrupting the underlying database (i.e. thread-safe server design). Since in the CHIL research project the Web Ontology Language OWL is used, the CHIL Knowledge Base Server API should be tailored specifically to OWL, rather than providing a more verbose and potentially error-prone interface to a more general ontology model. Finally, it must be possible to dynamically replace adapted ontology management systems.

3.2 Implementation

The architectural model on a class level of the CHIL Knowledge Base Server is partly depicted in Fig. 1. It makes extensive use of the Factory Design Pattern (Gamma, Helm, Johnson, and Vlissides 1995) and reflection capabilities of the Java programming language in order to be able to plug in hosts for arbitrary remoting protocols dynamically at runtime.

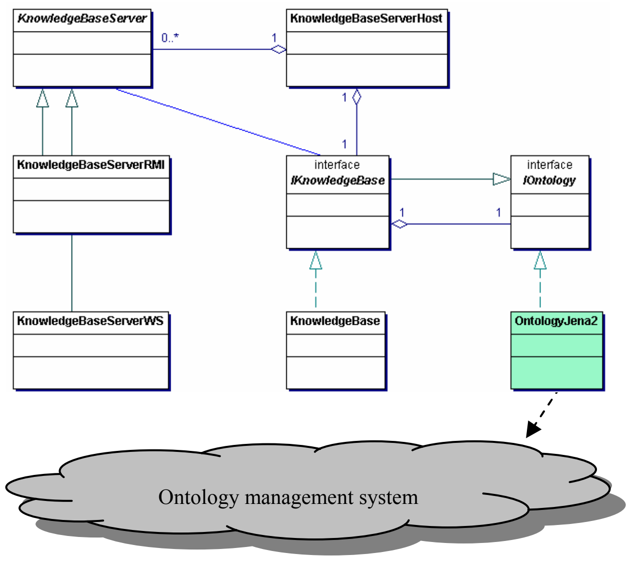


Fig. 1: Architectural model of the CHIL Knowledge Base Server

Ontology management system adapter classes implement the interface `IOntology`, which extends `IAskingABox`, `IAskingTBox`, `ITellingABox`, and `ITellingTBox`. Currently, about 150 methods are defined to manipulate and query OWL ontologies.

The green coloured exemplary class `OntologyJena2` would adapt the Jena 2 Semantic Web Framework. The interface `IKnowledgeBase` extends the interfaces `IOntology` and `java.rmi.Remote` to identify it as an interface whose methods may be invoked from a non-local Java virtual machine (for performance reasons the Java RMI remoting port is not implemented as a distinct `KnowledgeBaseServer` remoting host).

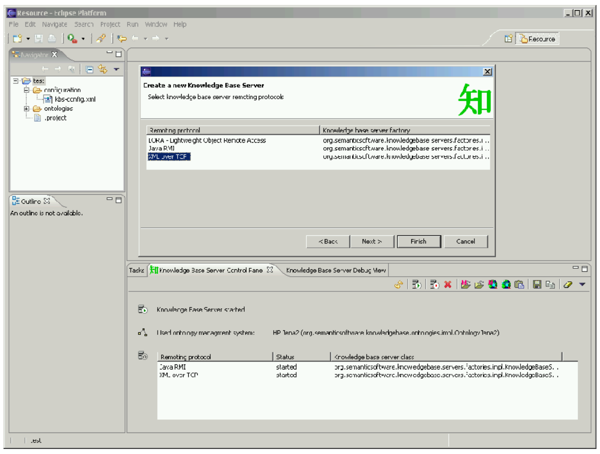
There is only one instance of the class `KnowledgeBase` at a time. This instance handles all incoming requests from Java RMI and all remoting hosts aggregated by the `KnowledgeBaseServerHost` instance.

The abstract base class `KnowledgeBaseServer` defines methods to bind and unbind the respective remoting host and to query status information. By dynamically binding and unbinding remoting hosts at runtime it is possible to update implementations with newer versions.

Persistence capabilities are defined by the `IPersistence` interface inheriting to `IKnowledgeBase`. `IPersistence` implementations can either use persistence capabilities of the underlying ontology management system or extend these features. It is possible to use off-the-shelf inference-, manipulation-, and persistence features from a variety of different systems to combine the best of several worlds.

3.3 Implementation as an Eclipse Plug-In

The front-end of the CHIL Knowledge Base Server was implemented as an Eclipse plug-in (Eclipse). This decision was taken to benefit from the capabilities and the standard behaviour provided by the Eclipse Rich Client Platform and to make it particularly easy to manage both software application- as well as ontology projects in one single Eclipse workspace.


Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Resource
Navigator
test
configuration
kbs-config.xml
ontologies
.project
Outline
< Back
Next >
Finish
Cancel
Tasks
Knowledge Base Server Control Pane
Knowledge Base Server Debug View
test

Create a new Knowledge Base Server

Select knowledge base server remoting protocols

知

Remoting protocol	Knowledge base server factory
LORA - Lightweight Object Remote Access	org.semanticsoftware.knowledgebase.servers.factories.i...
Java RMI	org.semanticsoftware.knowledgebase.servers.factories.i...
XML over TCP	org.semanticsoftware.knowledgebase.servers.factories.i...

An outline is not available.

Knowledge Base Server started

Used ontology management system: HP Jena2 (org.semanticsoftware.knowledgebase.ontologies.impl.OntologyJena2)

Remoting protocol	Status	Knowledge base server class
Java RMI	started	org.semanticsoftware.knowledgebase.servers.factories.impl.KnowledgeBaseS...
XML over TCP	started	org.semanticsoftware.knowledgebase.servers.factories.impl.KnowledgeBaseS...

Fig. 2: CHIL Knowledge Base Server Eclipse plug-in

Fig. 2 shows a screenshot of a running instance of the CHIL Knowledge Base Server. A project type “Knowledge Base Server” was added to Eclipse. Ontology projects are created using a “New Knowledge Base Wizard” as shown. This wizard searches specific library folders for Java class files that contain implementations of the interfaces `IKnowledgeBaseFactory` and `IKnowledgeBaseServerFactory`. Thus, both the CHIL Knowledge Base Server Eclipse plug-in as well as additional implementations of ontology management system adapters and remoting protocol hosts can easily be xcopy-deployed to Eclipse installations.

4 CHIL Knowledge Base Server API

The CHIL Knowledge Base Server is designed to locally and remotely store, manage, and retrieve arbitrary ontological data that meets OWL DL. Originally designed for use in the highly distributed, heterogeneous environment of the CHIL research project, special emphasis was put on good connectivity. Moreover, Sect. 5.1 will explain – for the most part by example – how the API was formally specified in order to make it possible to consistently adapt off-the-shelf ontology management systems.

4.1 Implementation

The CHIL Knowledge Base Server exposes its API via several ports. It is natively accessible via Java and Java RMI. A remoting technology and a code generation tool called LORA were developed to automatically generate client code for virtually every object oriented programming language. Finally, an XML-over-TCP interface is provided as a port for basically every programming language capable of parsing strings and of TCP socket communication.

The CHIL Knowledge Base Server API was defined using XML Schema Definition (XSD). A root element `KnowledgeBaseServerAPI` contains six child nodes `IAskingABox`, `IAskingTBox`, `ITellingABox`, `ITellingTBox`, `IPersistence`, and `IOntQL` for asking and telling the ABox and TBox of an OWL DL based ontology, for persistency functionality, and for executing an ontology query language, respectively.

As an example, Fig. 3 depicts the XML schema fragment presenting the definition of the `listDirectSubClasses` method that takes as input the identifier of an OWL class and returns all direct subclasses in case the input concept is properly defined in the ontology and the query to the underlying ontology management system succeeds. An `UndeclaredConceptException` is thrown in case the concept does not exist.

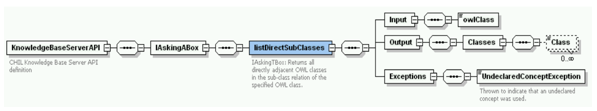

KnowledgeBaseServerAPI
CHIL Knowledge Base Server API definition
IAskingABox
listDirectSubClasses
IAskingTBox: Returns all directly adjacent OWL classes in the sub-class relation of the specified OWL class.
Input
owlClass
Output
Classes
Class
0..∞
Exceptions
UndeclaredConceptException
Thrown to indicate that an undeclared concept was used.

Fig. 3: XML Schema Definition based API definition

Using XSL Transformations (XSLT), Java interface code was automatically generated from the XML Schema Definition based method definitions. The above `listDirectSubClasses` definition resulted in the following Java code.

```
interface IAskingTBox {

  public String[]
    listDirectSubClasses(String owlClass)

  throws UndeclaredConceptException;

}
```

We want to emphasize that in contrast to other OWL API approaches, the CHIL Knowledge Base Server API does not depend on programming language specific data types. Thus, the automatically generated Java interface is completely remotable and can easily be exported via Java RMI. Moreover, the entire interface definition is rather service oriented, which does also facilitate integration with non object oriented client programming languages.

Such kinds of clients can use the XML-over-TCP port of the CHIL Knowledge Base Server. We decided to use TCP/IP via the socket interface because in state-of-the-art operating systems, local TCP/IP connections are usually routed via loop back or similar devices that bypass most of the TCP/IP stack; the advantage of having a single interface for local and remote communication when using the socket interface even for local communication therefore fully outweighs any marginal performance slowdown or latency imposed by socket communication.

Language independence was achieved by a two-step approach. Firstly, all communication is performed by transmitting and receiving XML messages over TCP/IP, rather than building upon some language dependent RPC-based mechanism. Data values are encoded with XSD data types, rather than using programming language-specific data types. While with XML messages based on XSD data types we achieve a highly programming language independent communication mechanism, we do not want to put the burden of generating and parsing XML messages on client programmers. Therefore, as a complementary step of our approach, for a selected set of programming languages, client libraries are provided that handle all XML-related work. Up to now client libraries are available for C#, C++, and Python.

The CHIL Knowledge Base Server handles multiple incoming requests with standard socket calls (i.e. listen on server port, accept request, rebind to different port). Each request is rebound to a different port and delegated to a thread of its own. In this way another connection on the server port can be accepted while the previous one is still being processed. In order to avoid corruption of the managed knowledge base data by concurrent access, without relying on the capabilities of the underlying ontology management system, all incoming requests are strictly serialized before they are executed on live data.

A remoting technology called LORA, which includes code generation features, was developed in order to make it particularly easy to reflect changes in the XSD based CHIL Knowledge Base Server API once these changes have been implemented in Java.

Both LORA remoting hosts as well as the XML-over-TCP port were devised as implementations of the `KnowledgeBaseServer` class (see Sect. 3.2).

4.1.1 Automatic Cross Language Client Integration with LORA

This subsection elucidates how client code for object oriented programming languages is automatically generated from the CHIL Knowledge Base Server API specification using LORA (Lightweight Object Remote Access), a novel and easy-to-use framework for accessing distributed objects locally or through the Internet (Schaeffer 2005).

LORA was developed as an alternative to existing remoting technologies in order to perfectly suit the requirements of the CHIL Knowledge Base Server API. In particular, we decided against the use of CORBA (CORBA) since there is not definitive reference implementation. Moreover, the CORBA specification has shown several defects that led to a number of revisions of the specification, which were not backward-compatible with existing implementations. Furthermore, the CORBA IDL does not cope well with recursively defined data types.

LORA is XML-based and offers advanced features such as automatic proxy class code generation and client session management. With LORA, every serializable class of an object oriented programming language can be used as a distributed object to offer its public methods to remote clients. In the CHIL Knowledge Base Server, LORA makes the `IKnowledgeBase` interface introduced in Sect. 3 accessible to remote clients. Up to now, LORA was implemented for Java and .NET programming languages. However, it does not depend on features exclusively offered by these two platforms. Any object oriented programming language with reflection capabilities could be used with LORA as well.

To use a class as a distributed object, LORA requires the class to inherit from the abstract base class `RemoteAccessible`. Public class methods can be annotated in a way, which is most suitable for the given programming language, to make them available for remote invocation. For example, for Java and C# annotations and attributes are used, respectively.

As method parameters and return types, LORA supports primitive language types and indexed or named compositions, which allow passing complex data structures through recursion. For each distributed class, proxy source code can be auto-generated for supported target languages. After transferring proxy code files to the client, the remote object can be transparently used as if the class was local.

5 CHIL Knowledge Base Server Testing Framework

Based on Description Logics terminology formal specifications were devised for methods of the CHIL Knowledge Base Server interfaces `IAskingTBox`, `IAskingABox`, `ITellingTBox`, and `ITellingABox`. Using these formal specifications and a reference ontology along with given result sets and side effects for particular method calls, a JUnit (JUnit) based testing framework was designed and implemented in order to automatically test adaptations of ontology management systems for consistency with the CHIL Knowledge Base Server API.

5.1 A Formal Specification of the CHIL Knowledge Base Server API

A formal specification of the CHIL Knowledge Base Server API was devised in order to make it possible to consistently adapt off-the-shelf ontology management systems. In particular, ambiguities had to be resolved that may be caused by informal specifications like “This method returns all super classes of the given class”. In such cases it mostly remains unclear if the result set will contain the OWL top level concept `http://www.w3.org/2002/07/owl#Thing` or not.

With a more rigid specification, it would be clear if in an adapter class the top level concept from the result set of an adapted method had to be removed in case the underlying ontology management system yields it. In addition, a more formal specification is machine readable, such that result sets could be validated according to the specification.

For the formal specification of the CHIL Knowledge Base Server API, we used the Z notation (Spivey 1992, ISO/IEC Information Technology 2002). Since the CHIL Knowledge Base Server API is specific to Description Logics, we added to the Z notation the syntax and semantics of Description Logics. Additionally, the semantics of the ‘ $\sqsubseteq$ ’-sign, which in Z denotes a sub-bag relation, was overwritten, such that it stands for the subsumption relation as defined by Description Logics.

Following, four examples are given how methods from the CHIL Knowledge Base Server interfaces `IAskingTBox`, `IAskingABox`, `ITellingTBox`, and `ITellingABox` were formally specified.

The method `listDirectSubClasses(String owlClass)` from the `IAskingTBox` interface, which returns all classes that are directly subsumed by the given class `owlClass`, was defined as shown in Fig. 4.

The method `listPropertyValues-OfIndividual(String role, String individual)`, which is defined in the `IAskingABox` interface, yields all values of role `R` of individual `IND`. The result set returned by this method was defined as depicted in Fig. 5.

listDirectSubClasses

$$\begin{array}{|l|}
\hline
\Xi\ \text{KnowledgeBase} \\
\text{owlClass?} : \text{String} \\
\text{subClasses!} : \mathbb{P}\ \text{String} \\
\hline
\text{subClasses!} = \{\, X.\text{toString}() \mid X \sqsubseteq \text{owlClass} \wedge \\
\qquad \forall\, X,Y.Y \sqsubseteq X \Rightarrow X \equiv Y \,\} \\
\hline
\end{array}$$

Fig. 4: Method listDirectSubClasses

listPropertyValuesOfIndividual

$$\begin{array}{|l|}
\hline
\Xi\ \text{KnowledgeBase} \\
\text{role?} : \text{String} \\
\text{individual?} : \text{String} \\
\text{propertyValues!} : \mathbb{P}\ \text{String} \\
\hline
\text{propertyValues!} = \{\, \text{val.toString}() \mid \exists\ \text{role.D} \\
\qquad \wedge\ \exists\ \text{val.(individual, val)} \in \text{role}^{\mathcal{I}} \\
\qquad \wedge\ \text{val} \in D^{\mathcal{I}} \,\} \\
\hline
\end{array}$$

Fig. 5: Method listPropertyValuesOfIndividual

The interface `ITellingTBox` comprises methods that can be used to modify the set of terminological axioms, which are defined in a knowledge base. The method `addClass(String class, String superClass)` adds a class `class`, which is subsumed by the class `superClass`, to the ontology. Accordingly, the axiom $\texttt{class} \sqsubseteq \texttt{superClass}$ where $\texttt{class}^{\mathcal{I}} \subseteq \texttt{superClass}^{\mathcal{I}}$ is added to the knowledge base as shown in Fig. 6.

The method `addPropertyValue-OfIndividual(String role, String individual, String value)`, which is defined in the ITellingABox interface, can be used to add a role assertion as depicted in Fig. 7.

addClass

Δ KnowledgeBase
class? : String
superClass? : String

class? $\in$ KnowledgeBase
superClass? $\in$ KnowledgeBase
KnowledgeBase' = KnowledgeBase $\cup$
{ class $\sqsubseteq$ superClass }

Fig. 6: Method addClass

addPropertyValueOfIndividual

Δ KnowledgeBase
role? : String
individual? : String
value? : String

role? $\in$ KnowledgeBase
individual? $\in$ KnowledgeBase
KnowledgeBase' = KnowledgeBase $\cup$ {
role(individual, value) }
where (individual$^{\mathcal{I}}$, value$^{\mathcal{I}}$)
$\in$ role$^{\mathcal{I}}$

Fig. 7: Method addPropertyValueOfIndividual

The next subsection will elucidate how these formal specifications were exploited by a testing framework in order to validate particular implementations of the CHIL Knowledge Base Server API.

5.2 A JUnit Based Testing Framework

The CHIL Knowledge Base Server explicitly supports replacing adapted off-the-shelf ontology management systems with different reasoning- and ontology management engines. In order to make these changes transparent, it is crucial to preserve the semantics of the CHIL Knowledge Base Server API according to the specifications as introduced in the previous subsection. A JUnit (JUnit) based testing framework as depicted in Fig. 8 was developed in order to automate the testing of particular adaptations.

Together with a reference ontology, the formal specification of the CHIL Knowledge Base Server API was taken to manually compute the result sets and side effects of each interface method. The same kind of output is computed by the adaptation of an off-the-shelf ontology management system. Both results are automatically compared with each other in order to validate the adapter.

The JUnit based testing component was implemented as an Eclipse plug-in. Thus, it can be managed in the same workspace along with the CHIL Knowledge Base Server and the adapter project.

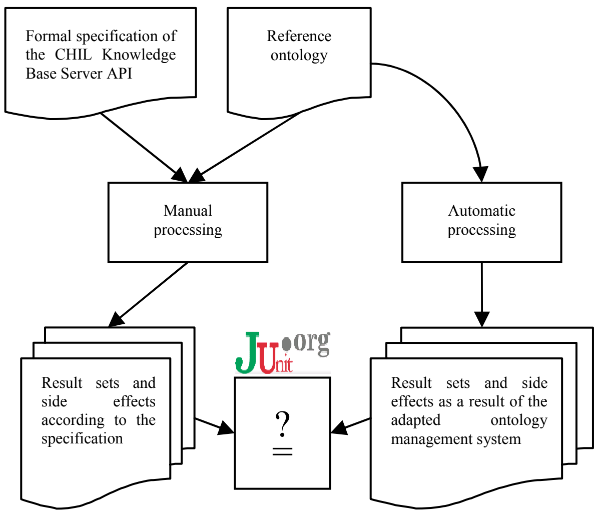

Formal specification of the CHIL Knowledge Base Server API
Reference ontology
Manual processing
Automatic processing
Result sets and side effects according to the specification
JUnit.org
?
=
Result sets and side effects as a result of the adapted ontology management system

Fig. 8: JUnit based testing framework

Current work includes the adaptations of HP Labs' Jena 2, of the IBM Semantic Network Base, and of the KAON 2 ontology management infrastructure, which has been completed for significant portions of the CHIL Knowledge Base Server API.

5.3 Building the Reference Ontology and Choosing Test Cases

The design of the reference ontology and the set of test cases were crucial in order to validate the CHIL Knowledge Base Server API as complete as possible.

The reference ontology had to comprise both a TBox and an ABox. Moreover, special effort was put in covering ontology features that are specific to $\mathcal{SHIQ}$ Description Logics on which the Web Ontology Language OWL is based (e.g. transitive roles, transitivity of the subsumption relation and nominals).

Test cases were chosen in a similar way, such that, with a presumably minimal number of method calls, a maximum of reasoning- and ontology management features could be covered. Hence, for a TBox C1 $\sqsubseteq$ C2 $\sqsubseteq$ C3 the method listSubClasses(String owlClass) is rather called with concept C3 and not with C2 in order to be able to check for the indirectly subsumed subclass C1 as well.

We would like to emphasize, that the testing framework is not to validate the behaviour of off-the-shelf ontology management systems against the specification of the Web Ontology Language. Rather, adapters of such systems are tested to comply with the specification of the CHIL Knowledge Base Server API with respect to one given reference ontology.

6 Conclusion and Outlook

We developed and implemented a pluggable architectural model for an ontological knowledge base server, which can be used to adapt off-the-shelf ontology management systems.

Based on XML Schema Definition and on a combination of the Z notation and formal Description Logics terminology, a programming language independent API was defined as a common interface to the CHIL Knowledge Base Server. The API supports forwarding of exception information to clients in order to provide programmers with as much information as possible without being restricted to one particular programming language.

The well defined ontology management API proved to be suitable both for developing auxiliary Eclipse plug-ins (e.g. for ontology visualization) and for accessing the CHIL Knowledge Base Server from a variety of perceptual components in the CHIL research project.

A code generation tool was devised to automatically generate client access code for object oriented programming languages. An Eclipse front-end was implemented in order to make it particularly easy to manage both software- and ontology projects in one single Eclipse workspace, which proved to be a good experience.

A JUnit based testing framework was developed in order to automatically validate adaptations of different ontology management systems, which include HP Labs' Jena 2, KAON 2, and the IBM Semantic Network Base.

Up to now, the reference ontology and the set of test cases had to be devised manually. Future work will include research on to what extent test cases could be generated automatically considering particular ontology features.

7 Acknowledgements

This work is part of the FP6 CHIL project (FP6-506909), partially funded by the European Commission under the Information Society Technology (IST) program. The authors acknowledge valuable help and contributions from all partners of the project.

In particular, we would like to thank Jürgen Moßgraber from Fraunhofer Gesellschaft's IITB for contributing to the architecture and API definition of the CHIL Knowledge Base Server and Tobias Kluge from Universität Karlsruhe's Interactive Systems Lab and Siamak Azodolmolky from Athens Information Technology for their feedback on deployment issues.

8 References

RDF: RDF Primer, W3C Recommendation 10 Feb 2004. http://www.w3.org/TR/rdf-primer/. Accessed on 11 Oct 2005.

DAML+OIL: DARPA's Information Exploitation Office. http://www.daml.org/2001/03/daml+oil-index.html. Accessed on 11 Oct 2005.

Baader, F., Calvanese, D., McGuiness, D., Nardi, D. and Patel-Schneider, P.F. (2003): The Description Logic Handbook. Cambridge University Press.

OWL: OWL Web Ontology Language Overview, W3C Recommendation 10 Feb 2004. http://www.w3.org/TR/owl-features/. Accessed on 11 Oct 2005.

Ontology Library: DAML Ontology Library, DARPA's Information Exploitation Office. http://www.daml.org/ontologies/. Accessed on 11 Oct 2005.

Protégé: Protégé knowledge acquisition system, Stanford University School of Medicine. http://protege.stanford.edu/. Accessed on 11 Oct 2005.

SNOBASE: IBM Ontology Management System, IBM Alphaworks. http://alphaworks.ibm.com/tech/snobase. Accessed on 11 Oct 2005.

Jena: Jena 2 - A Semantic Web Framework, HP Labs. http://www.hpl.hp.com/semweb/jena.htm. Accessed on 11 Oct 2005.

RACER: Racer Systems GmbH & Co. KG. http://www.racer-systems.com/. Accessed on 11 Oct 2005.

XSD: XML Schema Part 0: Primer Second Edition, W3C Recommendation 28 Oct 2004. http://www.w3.org/TR/xmlschema-0/. Accessed on 11 Oct 2005.

JUnit: JUnit testing framework. http://www.junit.org/. Accessed on 11 Oct 2005.

Pandis, I., Soldatos, J., Paar, A., Reuter, J., Carras, M. and Polymenakos, L. (2005): An Ontology-based Framework for Dynamic Resource Management in Ubiquitous Computing Environments. *Proc. International Conference on Embedded Software and Systems (ICESS)*, Xi'an, P. R. China.

KAON: Universität Karlsruhe (TH), Germany. http://kaon2.semanticweb.org/. Accessed on 11 Oct 2005.

JDBC: Sun Developer Network. http://java.sun.com/products/jdbc/. Accessed on 11 Oct 2005.

Sesame: Sesame RDF database, openRDF.org. http://www.openrdf.org/. Accessed on 11 Oct 2005.

Bechhofer, S. (2002): *The DIG Description Logic Interface: DIG/1.0*. University of Manchester, Oxford Road, Manchester M13 9PLA.

Levesque, H.J. (1984): Foundations of a functional approach to knowledge representation. *Artificial Intelligence* **23**:155-212.

Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D. and Rice, J. (1997): The Generic Frame Protocol 2.0. Technical Report. Artificial Intelligence Center, SRI International, Menlo Park, CA, USA.

Chaudhri, V.K., Farquhar, A., Fikes, R. and Karp, P.D. (1998): Open Knowledge Base Connectivity 2.0. Technical Report KSL-09-06. Stanford University KSL.

Patel-Schneider, P.F. and Swartout, B. (1993): Description-logic knowledge representation system specification from the KRSS group of the ARPA knowledge sharing effort. Technical report. AI Principles Research Department, AT&T Bell Laboratories.

Graham, P. (1995): *ANSI Common LISP*. Prentice Hall.

Horrocks, I. (1998): The FaCT system. *Proc. of the 2nd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX)*, Oisterwijk, The Netherlands, **1397**:307-312, Lecture Notes in Artificial Intelligence.

Horrocks, I. (1999): FaCT and iFaCT. *Proc. of the 1999 Description Logic Workshop (DL'99)*, Linköping, Sweden, 133-135, CEUR Electronic Workshop Proceedings.

Bechhofer, S., Horrocks, I., Patel-Schneider, P.F. and Tessaris, S. (1999): A proposal for a Description Logic interface. *Proc. of the 1999 Description Logic Workshop (DL'99)*, Linköping, Sweden, 33-36, CEUR Electronic Workshop Proceedings.

Brachman, R.J., McGuinness, D.L., Patel-Schneider, P.F., Resnick, L.A. and Borgida, A. (1991): Living with CLASSIC: When and how to use KL-ONE-like language. *Principles of Semantic Networks* 401-456, Morgan Kaufmann, Los Altos.

McGuiness, D.L., Resnick, L.A. and Isbell, C. (1995): Description Logic in practice: A CLASSIC application. *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, Montréal, Québec, Canada, 2045-2046.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995): *Design Patterns – Elements of Reusable Object-Oriented Software* 107–116, Addison-Wesley Publishing Company, Reading, Massachusetts.

Eclipse: Eclipse Foundation. http://www.eclipse.org/. Accessed on 11 Oct 2005.

XSLT: XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 Nov 1999. http://www.w3.org/TR/xslt. Accessed on 11 Oct 2005.

Schaeffer, J. (2005): LORA – Lightweight Object Remote Access. Studienarbeit. Universität Karlsruhe (TH), Germany.

CORBA: Object Management Group (OMG). http://www.omg.org/. Accessed on 11 Oct 2005.

Spivey, J.M. (1992): *The Z Notation: A Reference Manual*, 2nd edition. Prentice-Hall International Series in Computer Science, Prentice Hall.

ISO/IEC Information Technology (2002): Z Formal Specification Notation – Syntax, Type System and Semantics. ISO/IEC 13568:2002.