JAVA FRAMEWORK FOR MANAGING SEMANTIC REPOSITORIES BASED ON RDF STANDARD

František BABIČ*, Jozef WAGNER*, Peter BEDNÁR**
*Department of Cybernetics and Artificial Intelligence, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic, tel.: +421 55 602 4220, e-mail: frantisek.babic@tuke.sk, jozef.wagner@tuke.sk
**Centre for Information Technologies, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Němcovej 3, 042 00 Košice, Slovak Republic, tel.: +421 55 602 4219, e-mail: peter.bednar@tuke.sk

ABSTRACT

This paper presents a new approach for managing RDF repositories without necessary knowledge or experiences with RDF standard. The main differences between proposed solution and existing ones are adaptability on various semantic repositories, internal memory model for manipulation with RDF graphs and utilization of the Java annotations for mapping between ontology and java classes. Persistence API (P-API) represents Java library offered features for separation of business logic from RDF and allows simple integration of the semantic applications with selected RDF repository. The developers don't need to work directly with RDF query languages; they work only with Java classes. The P-API can also help developers with manipulation within RDF graphs constructed from data stored in repository, as the proposed RDF Memory model fits perfectly with such purposes. This Memory model can be used for more complex and straightforward manipulation with ontological knowledge models. The P-API was designed and implemented as a supporting solution on the middleware layer of the collaborative system in order to make bilateral communication effective and faster. The exploitation within KP-Lab system is described in a section no.3 as an illustrative example for possible integration approach.

Keywords: *RDF, Java classes, memory model*

1. INTRODUCTION

The Persistence API (P-API) represents a Java client library that provides the generic RDF (Resource Description Framework) persistence framework. The RDF (Resource Description Framework[1]) is W3C standard for modelling information based on making statements in form of triplets (subject, predicate, and object). P-API provides services for serialization and de-serialization of Java objects into RDF triplets, based on the RQL (RDF Query Language) and RUL (RQL Update Language) languages. This library was implemented mainly to separate business logic from RDF because of the fact that not many developers are familiar with this standard. In this case, by simple annotations, user can connect his Java Bean class with RDF repository, without using RQL/RUL at all.

1.1. Related work

Exploitation of Java libraries as accessing points to the repositories was at first applied in the domain of traditional databases, e.g. relational database. These libraries provide object-centric view on the data in form of Java beans and users can easily connect their DAO's (Data Access Object) to respective resources stored in the repository. Hibernate [1] dominates the open-source field of object-relational mapping libraries for the Java language.

As knowledge repositories are much younger in comparison with relational databases, persistence frameworks are scarce. In fact, there are few Java frameworks which support semantic data at all:

JRDF [3] (Java RDF) is an attempt to create a standard set of API's and base implementations to RDF. A key aspect is to ensure a high degree of modularity and to follow standard Java conventions. In this case we discussed several open questions with relevant JRDF authors and this collaboration resulted in the many similar characteristics of the both solutions, especially JRDF uses P-API memory model.

Sesame [4] is an open-source framework for querying and analyzing RDF data. It was created, and is still being maintained, by the Dutch software company Aduna. Only Sesame provides usable RDF to Java object API, similar to Persistence API, called Elmo. Elmo [5] is a Java bean pool implementation for the Sesame RDF repository, providing static Java bean interfaces to RDF resources. One of the main differences between Elmo and Persistence API is the use of underlying repository and query/update languages. Persistence API uses SWKM/RQL/RUL and Elmo uses Sesame RDF repository.

Jena [2] represents an open source Semantic Web framework for Java. It provides an API to extract data from and write to RDF graphs. Jena is similar to Sesame; though, unlike Sesame, Jena provides support for OWL (Web Ontology Language). However, Jena doesn't support Java bean persistence.

2. RDFSUITE

The Resource Description Framework (RDF) enables the creation and exchange of resource metadata as any other web data. To interpret metadata within or across user communities, RDF allows the definition of appropriate schema vocabularies (using RDFS). RDF is intended for situations in which this information needs to be processed by applications, rather than being only

[1] http://www.w3.org/RDF/

displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to applications other than those for which it was originally created.

The RDF data model is based upon the idea of making statements about web resources in the form of subject-predicate-object expressions, called triples. In RDF, resources are identified using Uniform Resource Identifiers[2] or URI's and describing resources in terms of simple properties and property values.

The semantic repository provides scalable persistence services for large volumes of knowledge objects descriptions and ontologies. Typical representative of open-source solution in this case is RDFSuite that is developed at FORTH-ICS[3] in Greece.

RDFSuite covers a suite of high level scalable tools [6], [7]:

- The Validating RDF Parser (VRP) - a tool for analyzing, validating and processing RDF schemas and resource description
- The Schema-Specific Data Base (RSSDB) - a persistent RDF Store for loading resource descriptions in an object-relational DBMS by exploiting the available RDF Schema knowledge
- Interpreter for the RDF Query Language (RQL) - the declarative language for uniformly querying RDF schemas and resource descriptions
- Interpreter for RDF Update Language (RUL) - the declarative language for uniformly updating resource descriptions.

The RDFSuite can be used by academics and software developers to produce scalable applications that rely on validating RDF/S compatible data for the Semantic Web.

2.1. Semantic Web Knowledge Middleware

The objective of the knowledge repositories is to provide generic management services for capturing and archiving; discovering and accessing; combining, modifying and tracking [8] of formal forms of explicit knowledge (i.e. declarative, procedural and causal).

Knowledge repository relies on ontologies in order to support interactions among learners and knowledge artefacts involved in learning processes. Ontology-based interactions include but are not limited to the ones that deal with organizing or annotating shared objects created by an individual or a group as well as sorting, classifying and retrieving all objects relevant to the problem at hand. Several ontologies are used to capture declarative (about), procedural (how), causal (why) or temporal (when) knowledge regarding a problem addressed by a group. Moreover, ontologies themselves might be possibly formed collaboratively based on the individual conceptualizations, if a consensus on the concepts and relations that are relevant to the task at hand can be reached.

In the case of KP-Lab project, the purpose of the Semantic Web Knowledge Middleware is to provide a set of services for manipulation with RDF data, e.g.:

The Knowledge Repository services provide a basic set of services for the remote manipulation of an RDF Knowledge Base (query, update, import, export). Via the use of these services the final user can import RDF Schemas or RDF files containing instances in the database, export schemas or instances, query upon the already stored material via the use of RQL queries or update the stored instances via RUL queries.

The Knowledge Mediator services (change, comparison, versioning and registry) aim at providing functionalities to support evolving ontologies and RDF Knowledge Bases (KB's). Upon a change request, the change service will automatically determine the effects and side-effects of the request and present it to the caller for validation. A comparison service is necessary to allow one to compare two versions of an ontology or RDF KB and identify their differences. The above functionalities are coupled with a versioning system, which is used to make different versions of the same ontology (or RDF KB) persistent, and with the registry service, which allows the user to classify the stored ontologies, using some related metadata for easy access and manipulation.

Last version of the SKWM services introduces concept of graph space, which serves as a grouping mechanism of statements, where the group is identified with an URI. This opens up new possibilities of RDF data query, manipulation and optimization.

P-API is connected to these services on the middleware layer.

3. PERSISTENCE API

P-API represents a client library used for managing all knowledge repository tasks as generating RQL/RUL and persisting/fetching of data from repository. The P-API provides the generic RDF persistence framework, which allows serialization and deserialization of the Java objects into RDF repositories. As well, this library can be used for manipulation with RDF graphs through internal RDF Object Model. The main functionalities of the P-API include:

Serialization of the Java objects to the RDF repository:

- generates RUL statements to create or update RDF representation of the specified Java Data Access Objects (DAO)
- assigns generated URI for newly created DAO
- batch processing; developer can aggregate several independent updates into one, increasing overall performance.

Deserialization of the RDF statements into Java objects:

- generates RQL statements to obtain RDF representation of the specified DAO
- processing native RQL queries

[2] http://www.w3.org/TR/uri-clarification/

[3] http://139.91.183.30:9090/RDF/

Use of RDF model: ability to populate a resource-based RDF Object Model from RQL query result.

3.1. P-API Internal Architecture

P-API architecture consists of several modules that are depicted on the Fig. 1. The consumers of P-API functionalities can use modules shaded in grey. The rest of modules are internal and are not normally accessed by applications developers.

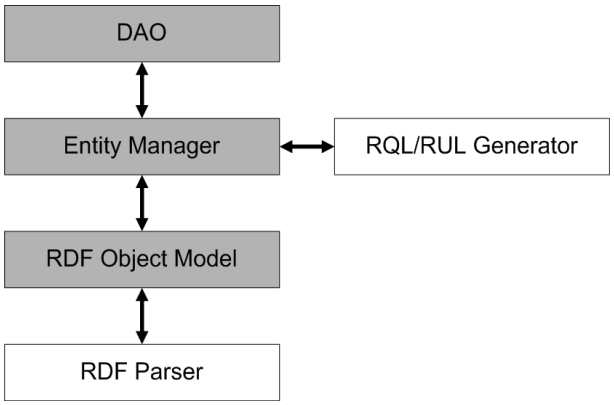

Fig. 1 P-API modules

- *Data Access Objects (DAO)* - DAO classes of the managed instances are implemented by the applications developer, which specifies mappings to the RDF model via Java annotations inserted in the source code. DAO are independent and can be used as any other Java Bean classes (e.g. they can be serialized).
- *Entity Manager* - creates instances of the deserialized Java objects and populates their properties from the RDF Object Model. In addition it updates RDF object model with values from the serialized Java objects. Entity Manager uses reflection API and annotations of classes or properties specified by application developer to map their properties to RDF model.
- *RUL/RQL Generator* - Entity Manager uses RUL/RQL generator to generate statements for fetching or updating RDF data stored in the RDF repository.
- *RDF Object Model* - is a memory representation of RDF in form of Java classes, managing in-memory representation of RDF statements loaded from the repository. Instances of this model can be populated from RQL query results. RDF Object Model provides Resource and Literal classes which represents RDF resources and literals respectively. Developers have an ability to traverse RDF graph in both directions (through direct and inverse properties). It is aimed to be resource-centric and independent on underlying RDF storage.
- *RDF Parser* - RDF parser creates instances of the RDF object model for the received RDF triples encoded in XML (or another RDF serialization format).

3.2. Entity Manager

Main purpose of Entity Manager is to separate business logic from RDF based on these several basic functionalities:

- persist()
  - Save/update
  - Automatically create URI for new items
  - Can persist more than one item
- find() - find item in RDF repository by type and URI
- remove() - removes any resource in RDF repository by URI
- manage() - use if you persist existing object not previously loaded with find()

Development process of an end-user tool, which will manage objects with the specified RDF Scheme; can follow these several steps:

- Developers create Java classes that correspond to the RDF types specified in the RDF Scheme. Developers can also implement various different Java classes (views) for one RDF type.
- In the source code of implemented Java classes, developer must use Java annotations provided by the persistence framework to specify mappings of the Java classes and properties to the RDF model.

Important note is that in this case developers work only with his DAO classes and calls Entity Manager methods. They don't need to know specifications of RQL/RUL and in fact they don't need to be very familiar even with RDF language. The simple example shows DAO class that can be used in Entity Manager:

```
@RDFResource(
    Namespace="http://www.kp-lab.org/ontologies/TLOModel#",
    RDFType="Task" )
public class Task implements Serializable {

    @URId
    protected URI uri;

    @Property(name="dc:title")
    private String title;

    @Property(name="subTask", targetEntity=Task.class)
    protected List<Task> subTasks;

    @InverseProperty(name="subTask")
    protected Task parentTask;
     }
```

3.3. Java annotations

Java annotations represent a special form of syntactic metadata that can be added to Java source code. P-API offers several types of annotations:

RDFResource - annotation of a class

- namespace parameter - Define Namespaces used in relevant DAO
- types parameter - RDF Types of relevant DAO

URId - required annotation, identifies which field represents URI

Property

- represents direct RDF property (resource is a subject)
- URI parameter - URI of a property
- targetEntity parameter - Type of target entity, used in collections

InverseProperty

- represents inverse RDF property (resource is an object)
- URI parameter - URI of a property
- targetEntity parameter - Type of target entity, used in collections

3.4. Lazy connections

P-API specifies and utilizes lazy collections in order to greatly improve performance when de-serializing from RDF repository. Main idea behind lazy collections is that DAO are fetched from RDFSuite only when they are really needed.

This feature of P-API is very similar to one in Hibernate. Lazy collections are transparent to users and they don't need to change they code in any way in order to utilize this feature. Users need to be aware of this feature though, and they need to keep Entity Manager connection open in order to be able to use lazy fetching *(*if connection was closed, exception would be thrown). In our case, only List<T> and Set<T> collections have support for lazy initialization. Class T must be of annotated DAO type.

3.5. RDF Object Model

RDF Object Model used in P-API is a bit similar to JRDF Model. We have added support for datatypes and resource view. It is aimed to be resource-centric and independent on underlying RDF storage. Fig. 2 shows UML class diagram of RDF Object model used in P-API to parse result from SWKM.

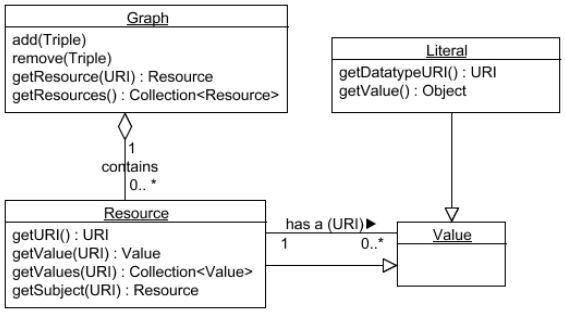

Fig. 2 Overview of RDF Object model

3.6. P-API and KP-Lab System

P-API was designed and implemented under KP-Lab project[4] as middleware component in KP-Lab System. KP-Lab System represents modular and flexible collaborative system with several integrated tools for different working or learning practices, e.g. operations with shared objects, modeling of various types of processes, virtual meeting support, visual modeling functionalities, integration of some Google Apps as Calendar or Docs, usage of semantic features as vocabularies or tagging and many others. Detailed information about this system can be found in [9] or [10]. KP-Lab System contains platform and virtual user environment called KP-environment.

[4] http://www.knowledgepractices.info/

The KP-Lab platform is a set of services that are based on heterogeneous technologies, which provide interoperability that is neither language nor platform dependent. For these reasons, the whole platform is based on web services with the "common language" for the communications and functionalities around shared objects in the whole system. Simple outline of the whole architecture is depicted on Fig. 3.

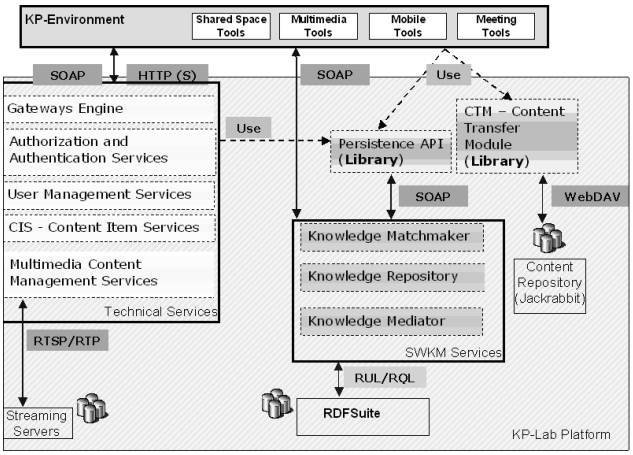

Fig. 3 KP-Lab System architecture [11]

- Semantic Knowledge Middleware Services (**SWKM Services**), providing storage and management of the metadata created by the KP-Lab tools.
- Content Management Services (**CTM**) are dedicated to creation and management of regular content (documents in various format) used in knowledge artefacts (content described by metadata), either towards KP-Lab's own content repositories or external content repositories. KP-Lab content repositories are implemented through Jackrabbit for the compatibility with the JSR-170 standard.
- **Technical services** cover those middleware support services, dedicated to the authorization and identity management, the user management, routing etc.

4. CONCLUSIONS

The Persistence API is a library developed within IST FP6 project KP-Lab for the purposes of manipulation with semantic data stored within RDF standard without so many experiences and knowledge in this field. From its first release, it has been used by many project partners in their services. It allows developers to focus on the application logic rather than on the RDF language or RQL/RUL, the low level mechanism of storing the knowledge data in the RDFSuite. While simple at interfaces, P-API provides advanced functionalities, completely transparent for its users. One of the main P-API characteristics is fast execution of the operations based on following features as concatenation of RQL and RUL statements in order to minimize number of queries; connection pool, in order to minimize delay when establishing connection; queue update operations speeding up whole process and giving control to the developer on

these batch operations. The P-API can be compared with some other existing solutions in this field as Elmo, JRDF or Jena to identify differences between them and possible advantages of our solution as: internal memory model, Java annotations and Java bean persistence. Besides RDF/OWL oriented models, there are also solutions for other logical formalisms [14] used in similar projects [13].

P-API comes with extensive documentation [12] including FAQ and several tutorials, from which users can learn to use Persistence API effectively in very short time.

ACKNOWLEDGMENTS

This work is the result of the project implementation: Centre of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120020) supported by the Research & Development Operational Program funded by the ERDF.

REFERENCES

[1] KING, G. – CHRISTIAN, B.: Java Persistence with Hibernate (Second ed.), Manning Publications, 2006, pp. 880, ISBN 1932394885.

[2] Mc BRIDE, B.: Jena: a semantic Web toolkit, Internet Computing, IEEE, Volume 6, Issue 6, Nov/Dec 2002, pp. 55-59.

[3] NEWMAN, A. – YUAN-FANG LI – HUNTER, J.: Scalable Semantics - the Silver Lining of Cloud Computing", In 4th IEEE International Conference on e-Science (e-Science 2008), Indianapolis, Indiana, USA, December 7-12, 2008.

[4] BROEKSTRA, J. – KAMPMAN, A. – van HARMELEN, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. ISWC 2002, LNCS, Volume 2342/2002, pp. 54-68, DOI: 10.1007/3-540-48005-6_7.

[5] Elmo, toolkit for developing Semantic Web applications, http://www.openrdf.org/doc/elmo/1.3/

[6] MAGIRIDOU, M. – SAHTOURIS, S. – CHRISTOPHIDES, V. – KOUBARAKIS, M.: RUL: A Declarative Update Language for RDF. Fourth International Semantic Web Conference (ISWC'05), Galway, Ireland, November, 2005.

[7] ALEXAKI, S. – CHRISTOPHIDES, V. – KARVOUNARAKIS, G. – PLEXOUSAKIS, D. – TOLLE, K.: The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases, In Proc. of the 2nd International Workshop on the Semantic Web (SemWeb'01), in conjunction with Tenth International World Wide Web Conference (WWW10). Hongkong, (2001) 1-13.

[8] Knowledge Management Case Study: Knowledge Management at Ernst&Young, 1997, http://www.itmweb.com/essay537.htm

[9] LAKKALA, M. – PAAVOLA, S. – KOSONEN, K. – MUUKKONEN, H. – BAUTERS, M. – MARKKANEN, H.: Main functionalities of the Knowledge Practices Environment (KPE) affording knowledge creation practices in education, In C. O'Malley, D. Suthers, P. Reimann, & A. Dimitracopoulou (Eds.), Computer supported collaborative learning practices: CSCL 2009 conference proceedings, pp. 297-306, Rhodes, Creek: International Society of the Learning Sciences (ISLS).

[10] MARKKANEN, H. – HOLI, M. – BENMERGUI, L. – BAUTERS, M. – RICHTER, C.: The Knowledge Practices Environment: a Virtual Environment for Collaborative Knowledge Creation and Work around Shared Artefacts. In J. Luca & E. R. Weippl (Eds.), Proceedings of ED-Media 2008, World Conference on Educational Media, Hypermedia and Telecommunications, Vienna, pp. 5035-5040, Chesapeake, VA: AACE.

[11] IONESCU, M. et al.: KP-Lab Platform Architecture Dosier. KP-Lab public deliverable D4.2.3. November, 2007.

[12] Persistence API documentation, http://kplab.tuke.sk/trac/wiki/kms-persistence-3

[13] SARNOVSKÝ, M. – FURDÍK, K. – TOMÁŠEK, M.: Application of Knowledge Management and Semantic Technologies in IT Service Management. In Proceedings of the 31th International Conference Information Systems Architecture and Technology (ISAT 2010), ISBN 978-83-7493-544-9, pp. 32-41, Wroclaw, Poland, 2010.

[14] TOMÁŠEK, M.: Controlling Communication and Mobility by Types with Behavioral Scheme. Acta Polytechnica Hungarica, Vol. 5, No. 4, ISSN 1785-8860, pp. 29-40, Budapest, 2008.

Received October 12, 2010, accepted February 11, 2011

BIOGRAPHIES

František Babič graduated (MSc.) at the Department of Cybernetics and Artificial Intelligence of the Faculty of Electrical Engineering and Informatics at Technical University of Košice in 2005. In the same year he began his PhD. study at the same department and successfully finished it with PhD. in 2009. He works also as a researcher in the Centre of information technologies, common workplace of Institute of Informatics Slovak Academy of Sciences in Bratislava and Technical university of Košice. He has been participating in several international and national research projects, actually as assistant professor at the original department. His scientific research is focusing on knowledge management, knowledge discovery, process modeling and project management.

Jozef Wagner received his Master degree in 2005 by the Technical University in Košice. Since 2005 he is working as a researcher in the Centre of Information Technologies, common workplace of Institute of Informatics, Slovak Academy of Sciences in Bratislava, and Technical University of Košice. In 2008 he began his PhD. study at the original department. His scientific research is focusing on the areas of semantic technologies, pattern discovering and programming (LISP, C++, Java).

Peter Bednár received his Master degree in 2001 and PhD degree in 2010 at the Technical University in Košice. Since 2005 he is working as a researcher in the Centre of Information Technologies, common workplace of Institute of Informatics, Slovak Academy of Sciences in Bratislava, and Technical University of Košice. His scientific research is focusing on the areas of text mining, knowledge management and application of the semantic technologies in eGovernment and eBusiness.