OntoMaven API4KB - A Maven-based API for Knowledge Repositories

Adrian Paschke

Corporate Semantic Web, Institute of Computer Science,
Koenigin-Luise-Str. 24, 14195 Berlin, Germany
paschke@inf.fu.berlin.de
http://www.corporate-semantic-web.de

Abstract. In this paper we introduce OntoMaven which adopts the Maven-based development methodology and adapts its concepts to manage knowledge artifacts stored in distributed OntoMaven KB repositories. With OntoMaven we address OMG's API for Knowledge Bases standardization (OMG API4KB) by supporting design time and life cycle management KB functionalities.

Keywords: Semantic Web, Ontology Repositories, Ontology Development, Ontology Engineering, Ontology Modularization, Ontology Management, Ontology Life Cycle Management, Corporate Semantic Web, Agile Knowledge Engineering, OMG API4KB

1 Introduction

In the life science domain there is a growing number of semantic knowledge bases (KBs) published on the Web, e.g. as linked data stores in the Linked Open Data (LOD) cloud or as semantic Deep Web sources which can be accessed via (e.g. SPARQL) query and service interfaces. While these services and knowledge bases support run-time interfaces for accessing and querying their knowledge, there is only limited support for life cycle management of the published knowledge artifacts (ontologies, rules possibly together with their instance data, facts) and for sharing and reusing them in new ontology-based engineering projects.

API4KB [1] is a standardization within OMG that aims at defining standardized application programming interface for knowledge bases (KBs). Typical API functionalities can be distinguished into design time, run time, and management functionalities. While run time functionalities address, e.g., the parsing, access, retrieval, and reasoning from KBs, with OntoMaven [2] we provide standardized interfaces implementing design time and life cycle management functionalities for Maven-based KB repositories, such as versioning and management of reusable KB artifacts in distributed remote and local OntoMaven repositories, dependency management, documentation, testing, packaging and deployment, etc. Although the OntoMaven approach is generic and suited for different types of knowledge bases (including e.g. rule-based KBs and their APIs), in this paper we focus on ontological KBs (ontologies) which OntoMaven supports as follows:

[1] www.omgwiki.org/API4KB/

[2] http://www.corporate-semantic-web.de/ontomaven.html

- OntoMaven remote repositories enable distributed publication of ontologies as **ontology development artifacts**, including their metadata information about life cycle management, versioning, authorship, provenance, licensing, knowledge aspects, dependencies, etc.
- OntoMaven local repositories enable the reuse of existing ontology artifacts in the users' local ontology development projects.
- OntoMaven's support for the different development phases from the design, development to testing, deployment and maintenance provides a flexible life cycle management enabling iterative agile ontology development methods, such as COLM [3], with support for collaborative development by, e.g., OntoMaven's dependency management, version management, documentation and testing functionalities, etc.
- OntoMave API plug-ins provide a flexible and light-weight way to extended the OntoMaven tool with existing functionalities and tools to access and work with knowledge bases and knowledge artifacts, such as semantic version management (e.g., SVont - Subversion for Ontologies [2, 5]), semantic documentation (e.g., SpecGen Concept Grouping [1]), dependency management of aspect-oriented ontology artifacts (e.g. [6]), automated testing (e.g., with the W3C OWL test cases and external reasoners such as Pellet), etc.
- Maven's API allows easy integration of OntoMaven into other ontology engineering tools and their integrated development environments (IDE).

In this paper we introduce the OntoMaven's conceptual approach and describe its implementation as Maven plug-ins.

2 OntoMaven's Design and Concept

In the following subsections we adapt the main concepts of Maven, so that they can be used in ontology development and ontology life cycle management. In particular, we focus on the (distributed) management of knowledge artifacts and their versioning, import and dependency management, documentation, testing, and deployment.

The main concepts of OntoMaven are:

- The Project Object Model (POM) is the main declarative XML description for managing an ontology project and its ontology artifacts. Based on the instructions in a POM file OntoMaven automates the different project goals in life cycle phases.
- Plug-ins implement the API functionalities which are interfaced by different OntoMaven goals. The plug-ins are executed using the descriptions in the POM file. There are three predefined life cycles, namely the *Clean* life cycle, which cleans the project, the *Default* life cycle, which processes, builds, tests and installs locally or deploys remotely, and the *Site* life cycle, which reports, documents and deploys the created HTML documentation, e.g. on an KB server.
- OntoMaven's local and remote repositories manage the used KB artifacts and plug-ins which implement support for versioning and dependency management. The distributed repository approach supports sharing and reuse of existing knowledge artifacts. The information about the used artifacts and their remote addresses (typically a URL) as well as dependency information are declaratively described in the POM file of a project. The downloaded artifacts have their own POM files in order to support, e.g., transitive dependencies.

Typical design time and management functionalities are, e.g.

– *Description, Management, and Versioning of Ontology Artifacts* - OntoMaven adopts Maven's artifact concept. It describes and manages ontologies and ontology modules as reusable ontology artifacts in a Maven Project Object Model (POM) which includes metadata about their id, grouping, location in remote and local repositories, versions, dependencies, etc.
– *Import and Dependency Management of Ontology Artifacts* - ontology artifacts are imported to and from OntoMaven repositories and transitive dependencies to existing ontology artifacts described in a POM are resolved during imports.
– *Documentation of Ontology Artifacts* - automated creation of *user documentation* and *technical documentation* from the artifact's metadata and analysis and inspection of the managed artifact's ontology (module).
– *Testing* - automated testing allows detecting inconsistencies, anomalies, improper design, as well as validation against, e.g., the intended results of domain experts' competency questions which are represented as ontology test cases.
– *Installation and Deployment of Ontology Artifacts* - install an ontology artifact into a (local) repository and deployment of artifacts and their documentations into an application (site).

The functionalities are implemented in Maven plug-ins which provide interfaces via so called *goals*. The execution order of goals can be organized into life cycle *phases*.

3 OntoMaven Plug-In Implementation

A Maven plug-in is a collection of one or more goals. The implementation of a Maven plug-in is done in an *Maven Plane Old Java Object (MOJO)*.In the OntoMaven approach, the phases and goals, which the plug-in implements, are defined by JavaDoc annotations in the source code of the Mojo class. Parameters are used to configure the plug-in execution. They can be declaratively configured in a POM.xml file or directly when calling a goal. An implemented plug-in can be installed using Maven `mvn install` and the plug-in goals can be declaratively called in a POM.xml of an OntoMaven project. Plug-ins provided by OntoMaven are, e.g.,

– the *OntoMvnImport* plug-in implements the imports of Ontologies into the Maven repositories. It is also checks if the import statements in the ontology including transitive imports can be resolved.
– the *OntoMvnSvn* plug-in provides ontology versioning support for OntoMaven. The plug-in computes semantic differences for the versioning of ontologies.
– the *OntoMvnReport* plug-in is implemented as Maven report plug-in. The goal `site` of this plug-in creates four different documentations about the ontology - a general *project documentation*, an *ontology report summary*, a *technical report*, and an *ontology visualization*.
– the *OntoMvnTest* plug-in implements functionalities for the test phase. The plug-in executes the configured tests using the goal `test` which parameterized by test suites such as the W3C OWL tests. It is also used internally in other phases such as the `package` goal.

OntoMaven can use all Maven compliant repositories. It follows a standard folder layout for its repositories; sources are in `$basedir/src/main/java`, resources in `$basedir/src/main/resource`, tests in `$basedir/src/test`, classes in

`$basedir/target/classes`, and packaged libraries in `$basedir/target/`. The goals for the OntoMaven plug-ins act as interfaces to the repositories and the installed ontology artifacts. For the OntoMaven proof-of-concept implementation we adapted the Apache Archiva Build Artifact Repository Manager as a managing tool providing a user interface for OntoMaven.

4 Conclusion

OntoMaven provides declarative goal and life-cycle based interfaces to plug-able design time functionalities and management functionalities for knowledge artifacts stored in distributed repositories. In this paper we have summarized the main concepts and plug-ins. For further details we refer to [4].

5 Acknowledgements

This work has been partially supported by the InnoProfile project "Corporate Semantic Web" funded by the German Federal Ministry of Education and Research (BMBF).

References

1. Gökhan Coskun, Mario Rothe, and Adrian Paschke. Ontology content "at a glance". In M Donnelly and G Guizzardi, editors, *Proceedings of the 7th International Conference on Formal Ontology in Information Systems*, pages 147–159, Graz, Austria, 2012. IOS Press.
2. Markus Luczak-Rösch, Gökhan Coskun, Adrian Paschke, Mario Rothe, and Robert Tolksdorf. Svont - version control of owl ontologies on the concept level. In Klaus-Peter Fhnrich and Bogdan Franczyk, editors, *GI Jahrestagung (2)*, volume 176 of *LNI*, pages 79–84. GI, 2010.
3. Markus Luczak-Rösch and Ralf Heese. Managing ontology lifecycles in corporate settings. In Tassilo Pellegrini, Sren Auer, Klaus Tochtermann, and Sebastian Schaffert, editors, *Networked Knowledge - Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 235–248. Springer Berlin Heidelberg, 2009.
4. Adrian Paschke. Ontomaven: Maven-based ontology development and management of distributed ontology repositories. In *9th International Workshop on Semantic Web Enabled Software Engineering (SWESE2013)*. CEUR workshop proceedings, 2013.
5. Adrian Paschke, Gökhan Coskun, Dennis Hartrampf, Ralf Heese, Markus Luczak-Rösch, Mario Rothe, Radoslaw Oldakowski, Ralph Schäfermeier, and Olga Streibel. Realizing the corporate semantic web: Prototypical implementations. TR-B-10-05:1–49, 02/2010 2010.
6. Ralph Schäfermeier and Adrian Paschke. Towards a unified approach to modular ontology development using the aspect-oriented paradigm. In *7th International Workshop on Modular Ontologies (WoMO 2013)*, 2013.