2015 48th Hawaii International Conference on System Sciences

Towards a Marketplace of Open Source Software Data

Fernando Silva Parreiras
Faculty of Business Sciences
FUMEC University
Belo Horizonte, Brazil
fernando.parreiras@fumec.br

Gerd Gröner
Ruhr Institute for Software Technology
University of Duisburg-Essen
Essen, Germany
gerd.groener@paluno.uni-due.de

Daniel Schwabe, Fernando de Freitas Silva
Departmnet of Informatics
PUC Rio
Rio de Janeiro, Brazil
{*fernd,dschwabe*}*@inf.puc-rio.br*

***Abstract*—Development, distribution and use of open source software comprise a market of data (source code, bug reports, documentation, number of downloads, etc.) from projects, developers and users. This large amount of data hampers people to make sense of implicit links between software projects, e.g., dependencies, patterns, licenses. This context raises the question of what techniques and mechanisms can be used to help users and developers to link related pieces of information across software projects. In this paper, we propose a framework for a marketplace enhanced using linked open data (LOD) technology for linking software artifacts within projects as well as across software projects. The marketplace provides the infrastructure for collecting and aggregating software engineering data as well as developing services for mining, statistics, analytics and visualization of software data. Based on cross-linking software artifacts and projects, the marketplace enables developers and users to understand the individual value of components and their relationship to bigger software systems. Improved understanding creates new business opportunities for software companies: users will be able to analyze and compare projects, developers can increase the visibility of their products, and hosts may offer plugins and services over the data to paying customers.**

***Keywords*-semantic web; open source software; linked data**

I. Introduction

In open source software projects, analysts, developers, architects, project managers and testers produce a large amount of data about software artifacts, from source code to license information. Usually, it is hard to track information crossing multiple artifacts [1], [2]. This lack of transparency results in technical problems, legal problems and market problems.

Technical problems comprise poor validation and tests of source code dependencies and low reuse rates of pieces of software. Legal problems include license decisions that are closely related to the business model. Market problems cause consumers to underestimate the value of a specific contribution or the expertise of small and medium enterprises.

Considering this context, in this paper, we address the following research question: what techniques and mechanisms can be used to help users and developers to understand artifacts by using links between information across open source software projects on the web?

The value of linking software engineering data comes from a veritable ecosystem of software data producers (e.g., analysts, developers, deployment engineers) data consumers (e.g., trainers, users and other stakeholders), and data hosts connecting producers and consumers and providing services to both.

For this ecosystem to work, participants must track dependencies between software artifacts crossing project boundaries. Although there exist multiple functional as well as non-functional dependencies between artifacts (or information sources) stored in various tools or databases, the data contained in the artifacts are not interlinked. Hence, producers and consumers currently do not have transparent access to inter-dependencies.

Contemporary approaches to increasing transparency (e.g., Krugle) provide unified search capabilities over multiple sources of software engineering data. However, these approaches are confined within company's boundaries where a limited and ad-hoc set of repositories is covered. Text-based search is still one of the main aspects (e.g., Snipplr and Koders) for assisting developers with finding meaningful pieces of code. However, none of the current approaches takes advantage of the semantic information in the hidden links between software artifacts; neither do they facilitate external sources to improve search results.

In this paper, we present a framework for a marketplace for open source software data. We build upon linked open data (LOD) techniques as a flexible representation means for software data and dependencies between software artifacts. These LOD principles offer easy access to software data on the marketplace for several marketplace actors. We outline our contribution as follows. Section II describes the idea of a marketplace for open source software and Section III derives the opportunities of such a marketplace. In Section IV, we present our solution for supporting the marketplace, describing roles in Section IV-A and the architecture in Section IV-B. In Section V, we present a proof of concept to validate the approach. Section VII concludes the paper.

II. Scenario

A marketplace is a (real or virtual) location where producers and consumers exchange their products and services.

1530-1605/15 $31.00 © 2015 IEEE
DOI 10.1109/HICSS.2015.439

Accordingly, a marketplace for software data is the place where software producers and consumers carry on their trade. In the following, we illustrate the marketplace and its main actors.

Software producers (P) act as supplier on the marketplace. They want to attract as many as possible customers that download and use the offered product, either as standalone or as companion product. In general, a prerequisite for distributing a product is to get attraction and interest of potential customers. Thus, software producers expect from a marketplace to get a high visibility of their software data.

The *consumers (C)* are looking for open source software data. For a certain need, they are searching for the best-suited software. The marketplace should offer means for searching and browsing for software according to certain individual preferences. Visualization services should provide graphical interfaces to help developers in making sense of the software data and allow a comparison of alternative software data. Consumers should also be able to analyze the context of software like license issues and activity logs.

The marketplace for open source software requires yet some *hosting service (H)* that provides the infrastructure for the software data. This includes additional features like facilities for software data analysis in order to compare software data, to recognize and explore patterns and dependencies between software. To remedy this, the software data themselves and the metadata of software have to be incorporated into the hosting service.

Hosting a marketplace requires complex software that covers a variety of features. A marketplace for open source software data should follow the good principles of open source software such that the marketplace is not set up by a single monolithic system but rather in terms of various loosely coupled subsystems and plugins. Thus, a hosting service consists of multiple plugin developers that offer plugins for dedicated purposes of the marketplace, e.g., for the analysis of software components.

III. Possible Outcomes

The scenario presented above, open opportunities for a marketplace of open source software engineering data as follows.

A. Improved Competitive Position by Providing Better Services to Businesses

Open source software has seen a massive growth during the last decade. Thus, open source software development is strategic for lowering the barriers to entry for new service producers and enabling them to develop and innovate faster.

A marketplace for open source software data facilitates the entry for open source software producers and providers by offering visualization services. It gives visibility, trust and opportunities to open source software producers for placing their products with all their dependencies into the open source software market.

The marketplace enables open source software consumers to understand, reuse and extend pieces of open source software in a holistic way. Linking data across software artifacts allows software consumers to make sense of the impact of adopting a component from multiples points of view, e.g., technical and legal.

B. Reduced Development Costs and Shorter Time-to-Market

Empowering open software producers and consumers with high value added information improves visibility, increase reuse, support decision-making activities and management of software licenses, reducing development costs and shortening time-to-market.

An open source software marketplace enables software developers to produce software visible to the market. Even small projects have a chance to be found and selected for being used as a third party library, which is the basis of many business models of small software developing companies. Furthermore, developers are able to find products and analyses to help to improve their own software engineering processes.

IV. A Marketplace for Open Source Software Data

We follow the notion of large-scale distributed data representation defined in the linked open data (LOD) principles [3], where a LOD cloud is a distributed store of multiple data sources with inter and intra links between data items. Thus, a marketplace for open source data consists of a LOD cloud, which is fed with linked software engineering data that are generated by LOD-extractors. In the following, we describe the actors (roles) in such a marketplace, followed by a detailed presentation of the marketplace architecture and the underpinning technology.

A. Marketplace Services and Roles

To deal with data quality, the marketplace has to provide services for each role. Based on the software data cloud, services have to align artifacts, matching bug reports with commit messages, or linking forum entries and information about methods of a class with commonly used datasets (e.g., DBpedia).

Figure 1 illustrates the marketplace for open source software data and the involved roles, which are explained in the remainder of this subsection.

1) Software Data Producers: A marketplace allows open source software producers to expose the data they produce to the cloud, hence maximizing the visibility of their products and the sharing of data across open source software communities. The marketplace brings the following new functionalities to open source software producers: ability to share data using standard formats and advanced tools

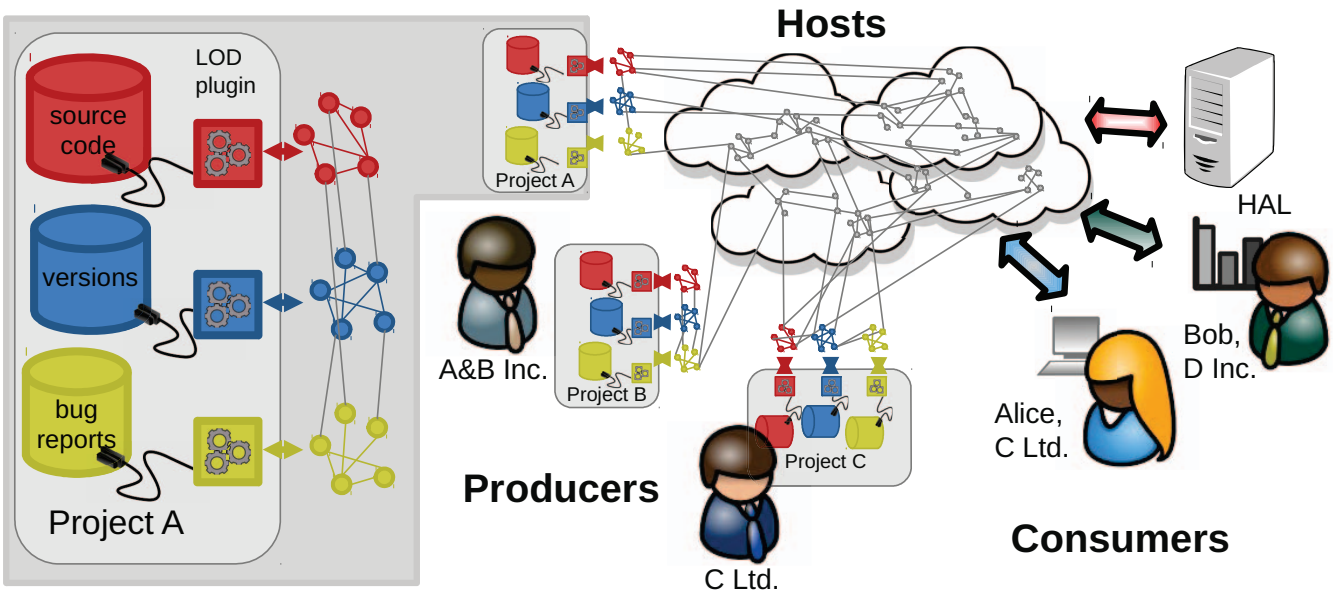

Figure 1. A marketplace for open source software data, including players (producers, consumers, and hosts), projects (A, B, C) and their artifacts (bug reports, versions, source code) and the open source software linked data cloud.

for extracting, storing, querying and visualizing the data about open source software; ability to follow projects clones and to interlink each project information system with the ones of other open source software producers. Typically, each large open source software integrator maintains its own database about the packages it comprises, and the associated bugs, patches, configurations, discussions, etc. A marketplace offers means for publishing this data into the cloud using standard formats, and for facilitating the creation of new links across distributed information systems. For example, standard mechanisms that allow for automating the tracking of bug reports across different bug trackers typically for notifying the upstream projects of changes in the bug statuses from within a downstream community.

2) Software Data Consumers: The usage of open source software components for creating complex software solutions within the industry is increasing. The marketplace eases the processes for creating, reusing, maintaining and evolving these solutions, by bringing the following capabilities to open source software consumers: ability to analyze the impact of new software releases, new security vulnerabilities, new bug reports, new hardware compatibility information, new features or new projects; ability to assess the activity of open source software projects by taking into account the clones, the downstream projects, the user communities, etc.; ability for open source software consumers to interlink the data produced internally around open source software with the public open source software linked open data thanks to the availability of data extractors and connectors; ability to collaboratively correct, enhance, rate, compare, integrate, export, the data provided by the marketplace.

Another challenge facing producers and consumers of open source software is to determine which software package providing specific functionality is actively maintained. In order to gather this information, currently, users typically need to read mailing lists, read commit logs, browse bug trackers and use word of mouth to determine whether a project is actively maintained. Providing metadata about project activity, use, similar projects and projects using specific code help consumers to choose which software is suitable in a given situation.

3) Software Data Hosting Services: Hosts of open source software data are able to provide advanced services for analyzing the dependencies of the projects they host, including external dependencies that go across hosting services. Currently, sites hosting open source software projects focus on providing the source code itself, along with collaboration services for producers of open source software.

In the future, the proposed approach for representing and linking open source software data will provide solutions for dealing with decentralized software repositories. In particular, version control software has seen a shift from centralized to decentralized systems over the years. As there is no longer a canonical version of the source code, it is hard to know which features have been added to other clones of a source code repository, or where a specific bug has been resolved. Information alignment services will simplify the process of identifying the correct links even for decentralized repositories. As a result, hosts will be able to

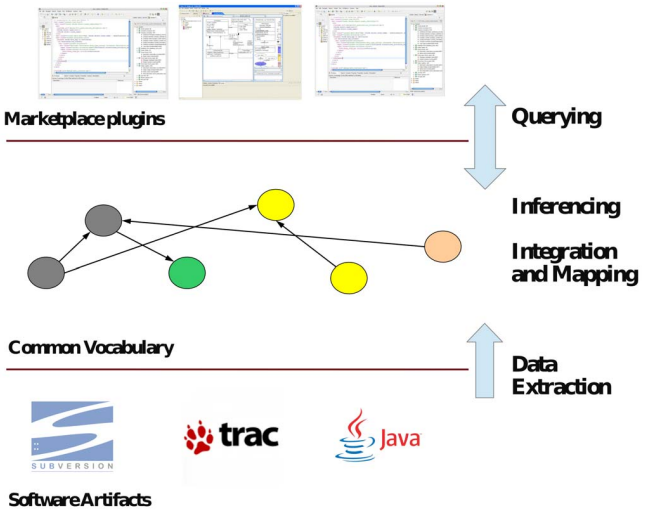

Figure 2. A three-layered architecture for the open source software data marketplace.

attract customers by offering suitable hosting services.

B. Conceptual Architecture

The conceptual architecture describes the components of the marketplace and their functionality. The marketplace relies upon the data produced from the LOD extractors. The ontologies serve as metamodel supporting the generation of LOD from a software artifact, e.g., an ontology that describes how to ask for method and class names. Here, we use common vocabularies for programming languages like Java and C#, for issue trackers and software repositories, as described below.

1) Representation Formalism and Data Extraction: The backbone of the marketplace is a flexible data representation formalism that allows managing software data from loosely coupled, interconnected software artifacts. The language and reasoning paradigm for flexible data representation is the Resource Description Framework (RDF) and the more expressive family of description logic languages covered by the W3C recommendation Web Ontology Language (OWL) [4].

Based on these representation formalisms, the next step is to extract software data from all kinds of software artifacts that adhere to a certain language or metamodel. We use a generic approach to extract the schema and the content of controlled natural languages that use Ecore-based models [5]. Ecore and OWL have many similar constructs, e.g., classes, attributes and references.

Based on these similarities, we use a generic transformation script to transform any Ecore-based languages into OWL TBox/ABox – The OWLizer [6]. Fig. 3 depicts the conceptual schema of transforming Ecore-based languages into OWL.

The four lanes, Actor, Ecore, Model Transformation and OWL show three modeling levels according to the OMGs

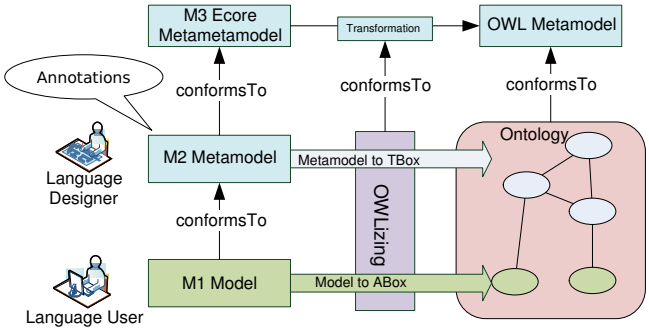

Figure 3. OWLizer: A generic extractor for transforming artifacts written in controlled natural language into OWL.

Four layered metamodel architecture [7]: the metametamodel level (M3), the metamodel level (M2) and the model level (M1). Vertical arrows denote instantiation whereas the horizontal arrows are transformations from the Ecore modeling space to the OWL modeling space. The ontology consists of the TBox that represents the metamodel and the ABox that expresses the model.

Thus, the language designer can create new attributes and references and define axioms for them in OWL by using reference properties. It is possible to add further modeling axioms in OWL for the model elements, e.g., expressing two artifacts as identical.

A model transformation takes the metamodel and the annotations as input and generates an OWL ontology where the concepts, enumerations, properties and data types (TBox) correspond to classes, enumerations, attributes/references and data types in the metamodel. Another transformation takes the model created by the language user and generates individuals in the ABOX of the same OWL ontology. The whole process is transparent for language users.

The structural mapping from Ecore-based metamodels and models to OWL makes MOF models available as federated, accessible and query-ready LOD resources. Extractors transform models into a common representation in OWL ontologies according to this structural mapping. Having models represented in OWL ontologies, one might connect these ontologies and process these ontologies in a federated way. Thus, the resulting OWL representations offers an integration management functionality (transforming and linking) within the marketplace architecture.

2) Inferencing: The inferencing layer is realized by reasoning services that operate over OWL vocabularies. Reasoning services are services provided by reasoning systems with respect to the ontology. Standard reasoning services are services available in all reasoning systems, whereas non-standard reasoning services are extensions of basic reasoning services.

The standard reasoning services for TBox are satisfiability and subsumption. A class C is unsatisfiable ($C \sqsubseteq \bot$) with

respect to an ontology $\mathcal{O}$ if C is empty (does not have any instances) in all models of $\mathcal{O}$. Satisfiability checking is useful for verifying whether an ontology is meaningful, i.e., whether all classes are instantiable.

Subsumption is useful to organize hierarchically classes according to their generality. A class C is subsumed by another class D with respect to an ontology $\mathcal{O}$ if the set denoted by C is a subset of the subset denoted by D for every model of $\mathcal{O}$.

The standard reasoning services for ABox are instance checking, consistency, realization and retrieval. *Instance checking* proves whether a given individual i belongs to the set described by the class C. An ontology is *consistent* if every individual i is instance of only satisfiable classes. The *realization* service identifies the most specific class a given individual belongs to. Finally, the retrieval service identifies the individuals that belong to a given concept.

3) Data Integration and Mapping: Linking between difference sources of software data is an essential aspect of a data cloud in order to enable searching and exploration among data of different but related software artifacts. Accordingly, one requires services for community collaboration and system coordination as well as matching between data in the cloud.

Common Vocabulary: Open Services for Lifecycle Collaboration (OSLC) is an open community that aims to standardize data sharing between tools that are part of Application Lifecycle Management. Initially proposed by IBM in 2008, but currently involving other organizations and independent developers, OSLC specifications allow tools from different vendors to integrate their data and workflows to support the complete lifecycle of an application, from conception to implementation. These specifications are defined as RDF vocabularies and follow the principles of Linked Data. The following are the OSLC vocabularies used:

1) OSLC Automation Management Vocabulary: defines a vocabulary of terms that are often used in the automation of development tasks. This includes automation plan, automation request and automation result. This vocabulary was used to represent information from build definitions and build execution records that are extracted from continuous integration tools.
2) OSLC Change Management Vocabulary [1]: defines terms commonly used in this area, such as change request, activity, task, and the relationships between these resources. This vocabulary was used to represent information from bugs, tasks and activities present in issue trackers.
3) OSLC Asset Management Vocabulary: describes information present in Asset Management systems. One example of software asset is the executable generated by an automation plan, such as a Java Archive (JAR) or a dynamic-link library (DLL). This vocabulary was used to represent information extracted from project dependencies, such as library name, version and localization.
4) OSLC Quality Management Vocabulary [2]: models artifacts such as test plans, test cases and their results. This vocabulary was used to represent information from executed unit tests in continuous integration tools during the build of a project.
5) OSLC Source Control Management Vocabulary [3]: include resources related to Source Control Management Systems like Change Set, Change, Baseline and Snapshot.

The usage of a common vocabulary still requires the reconciliation of resources expressed using distinct controlled natural languages before usage. This reconciliation of concepts across controlled natural languages is the object of study in ontology matching discipline. For a deeper understanding of this topic, please refer to [8].

Ontology matching is the discipline responsible for studying techniques for reconciling multiple resources on the web. It comprises two steps: match and determine alignments and the generation of a processor for merging and transforming. Matching identifies the correspondences. A correspondence for two ontologies A and B is a quintuple including an id, an entity of ontology A, an entity of ontology B, a relation (equivalence, more general, disjointness) and a confidence measure. A set of correspondences is an alignment. Correspondences can take place at the schema-level (metamodel) and at the instance-level (model).

Matchings use multiple criteria: name of entities, structure (relations between entities, cardinality), background knowledge, e.g., existing ontologies or wordnet. Techniques can be string-based or rely on linguistic resources. Furthermore, matchings are established according to the structures: (i) Internal structure comparison: this includes property, key, datatype, domain and multiplicities comparison. (ii) Relational structure comparison: the taxonomic structure between the ontologies is compared. (iii) Extensional techniques: extensional information is used in this method, e.g., formal concept analysis.

4) Querying: Querying ontologies is a research field that comprises multiple techniques and languages. We limit the scope of our analysis to the SPARQL-like language SPARQL-DL. The reason for using SPARQL is that it is a W3C standard query language [9], and it includes the definition of graph pattern matching for OWL 2 Entailment Regime [10].

SPARQL 1.0 [9] is the triple-based W3C standard query language for RDF graphs. The semantics of SPARQL 1.0 is based on graph pattern matching and does not take into

[1]http://open-services.net/bin/view/Main/CmSpecificationV2

[2]http://open-services.net/bin/view/Main/QmSpecificationV2

[3]http://open-services.net/bin/view/Main/ScmSpecV1

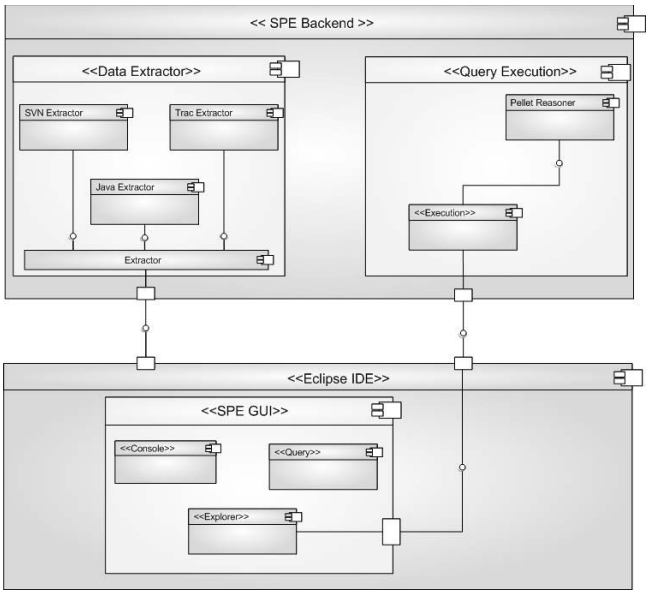

Figure 4. The UML Component Diagram of the semantic project explorer (SPE), complying with the proposed three-layered architecture (Fig. 2).

account OWL, although the specification allows for extending the SPARQL basic graph matching. SPARQL 1.1 [9] addresses this problem by specifying an OWL entailment regime for SPARQL.

Sirin and Parsia [11] have done preliminary work on answering full SPARQL queries on top of OWL ontologies on SPARQL-DL. SPARQL-DL enables users to write queries relying on the expressiveness of OWL. Next, we describe the abstract syntax of SPARQL-DL and its semantics.

The semantics of SPARQL-DL extends the semantics of OWL to provide query evaluation. We say that there is a model of the query $Q = q_1 \wedge \ldots \wedge q_n$ ($\mathcal{I} \models \sigma Q$) with respect to an evaluation σ iff $\mathcal{I} \models \sigma q_i$ for every $i = 1, \ldots, n$.

A solution to a SPARQL-DL query Q with respect to an OWL ontology $\mathcal{O}$ is a *variable mapping* $\mu : \mathcal{V}_{var} \rightarrow \mathcal{V}_{uri} \cup \mathcal{V}_{lit}$ such that $\mathcal{O} \models \mu(Q)$.

In the next section, we illustrate the usage of SPARQL queries with an example of a software design pattern.

V. Validation

The realization of the marketplace for open source software has been the scope of PhD and MSc theses. In this paper, we illustrate some preliminary results of applying our approach with two plugins at the Querying level of the architecture (Section IV-B4).

A. The Semantic Project Explorer

We invite the readers to consider the following scenario. A new developer (*software data consumer*) joins the development team (*software data producer*) and needs to understand how the existing source code has been build. An efficient

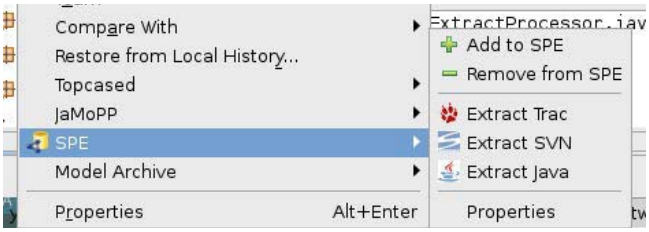

Figure 5. Screenshot of the Context Menu of the SPE Plugin for the Eclipse Framework.

way of learning about source code is by observing the occurrence of software design patterns [12]. Understanding of applying design patterns into the source code enables newcomers to learn how the source code is structured. In the marketplace, the new developer finds a plugin (*developed by software data hosting services*) to help software developers in verifying the quality of their source code – the semantic project explorer (SPE).

The SPE is an example of a plugin we have developed to test our approach. Figure 4 shows the components used by the SPE plugin, complying with the conceptual architecture explained in Section IV-B. The component Data Extractor comprises the set of components required to extract information from artifacts and upload them to the cloud (layer *representation formalism and data extraction*, Section IV-B1). The component Eclipse IDE groups the components the front-end representing interaction points with users (layer Querying, Section IV-B4). Finally, the component Query Execution comprises the set of components required for supporting inferencing and query execution (layer Inferencing, Section IV-B2)

In order to use the plugin, the new developer firstly needs to extract information from artifacts into a common vocabulary, by uploading source code and other related artifacts like bug reports and versioning information. Figure 5 depicts the context menu used for triggering this task. The artifacts are then extracted and transformed into OWL and linked to common vocabularies existing in the cloud.

Using our approach, we are able to extract information from a internal project as proof of concept. Table I presents the list of artifacts part of our development process and the corresponding metrics. For each artifact, we present the number of classes (C) on the metamodel, the number of instances (I), the number of non commenting source statements (NCSS) for source code and the number of RDF triples (T).

The developers are now able to use the plugin to assess the quality of the source code. The component Eclipse IDE (Figure 4) retrieves automatically from the cloud the updated list of software design patterns found in the source code. By clicking on the name of the design pattern on the left side, the console shows the list of the classes matching the

design pattern on the bottom (Figure 5). Listing 1 shows the example of the SPARQL query executed for finding the Singleton Design Pattern.

Table I
METRICS OF THE TWOUSE TOOLKIT PROJECT

Phase	Artifact	C	I	NCCS	T
Requirements	Requirements specification	24	212	-	-
	UML diagrams	261	174	-	-
Analysis	BPMN diagram	24	754	-	-
Design	Metamodel	23	5370	-	-
	Generator specification	20	3374	-	-
	Grammar specification	38	7611	-	-
	Model transformation	46	8043	-	-
Code	manifest	53	2824	-	-
	source code	345	-	454.350	616.643
Management	Versioning	22	7032	22366	

Listing 1. A SPARQL Querying that matches the occurrences of the Singleton Design Pattern

```
PREFIX rdfs:
    <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX java: <http://www.rdfcoder.org/2007/1.0/>
PREFIX java0: <http://www.rdfcoder.org/2007/1.0#>
PREFIX evo:
    <http://www.ifi.unizh.ch/ddis/evoont/2008/11/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?file ?LastAuthor ?Commitment
WHERE{
  ?clazz rdfs:subClassOf java0:JClass.
  ?clazz java:haslocation ?file.
  ?clazz java:containsattribute ?instance.
  ?instance java:attributetype ?type.
  ?type owl:sameAs ?clazz.
  ?clazz java:containscontructor ?constructor.
  ?constructor java:hasvisibility "private".
  ?file evo:message ?Commitment.
  ?file evo:hasAuthor ?person.
  ?person foaf:name ?LastAuthor
}
```

B. The Semantic Miner

SemanticMiner is a plugin addressing the challenge of Mining Software Repositories [13]. In order to demonstrate the role of SemanticMiner, we present the example of mining information from the source code of a project: its structure, versions, tests and dependencies.

In the following, we rely on field studies carried out with software practitioners [14], [15] that identified a set of questions that are recurrent among actual developers. These questions are difficult to be answered by the lack of support of existing tools. We focus specifically on the information related to the source code of a project: its structure, versions, tests and dependencies. The answers to these questions were reported by practitioners as being used as a basis in decision making during the project, for example, who should be allocated to fix a specific piece of code, or what changes should be postponed due to the impact it may cause on other systems.

We illustrate how SemanticMiner is able to answer such questions by giving two representative examples. For each natural language question, we show it is translated into a SPARQL query that can be executed against a specific project's mined data, extracted through the SemanticMiner.

The selected project was Apache Mahout [4], a scalable machine learning library maintained by the Apache Foundation. The choice of this project was made due to its characteristics: it is written in Java, uses Apache Maven for dependency management, its continuous integration is executed in Jenkins and uses Atlassian Jira as its issue tracker tool. The project currently contains about 1860 commits and 1220 issues created over six years.

1) Who to assign a code review to? / Who has the knowledge to do the code review?: This question was translated as a query to search all the owners of commits that caused an impact on one or more source entities that are part of the code in focus (Listing 2). The rationale is that if a developer committed changes that alter that specific piece of code, s/he should be qualified to review it.

Listing 2. A SPARQL Querying that matches the occurrences of the Singleton Design Pattern

```
PREFIX sm:<http://semanticminer/>
PREFIX
    rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX oslc_scm:<http://openservices.net/ns/scm#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX dcterms:<http://purl.org/dc/terms/>

SELECT ?developer ?element (COUNT(?impact) AS
    ?count)
WHERE { ?developer a sm:Person .
?developer ^sm:commitAuthor ?changeSet .
?changeSet a sm:ChangeSet .
?changeSet sm:commitDate ?impactDate .
?changeSet ^sm:contextTo ?impact .
?impact sm:impactOfAfter ?element.
FILTER NOT EXISTS { ?impact sm:impactType
    'NOTHING' }
FILTER ( ?element IN ( $element ) )}
GROUP BY ?developer ?element
ORDER BY DESC(?count) DESC(?impactDate)
```

As input to this query we have chosen the two most invoked methods in the project: `AbstractJob.addOption(String)` and `AbstractJob.getOption(String)`. Listing 3 presents the results.

[4]http://mahout.apache.org

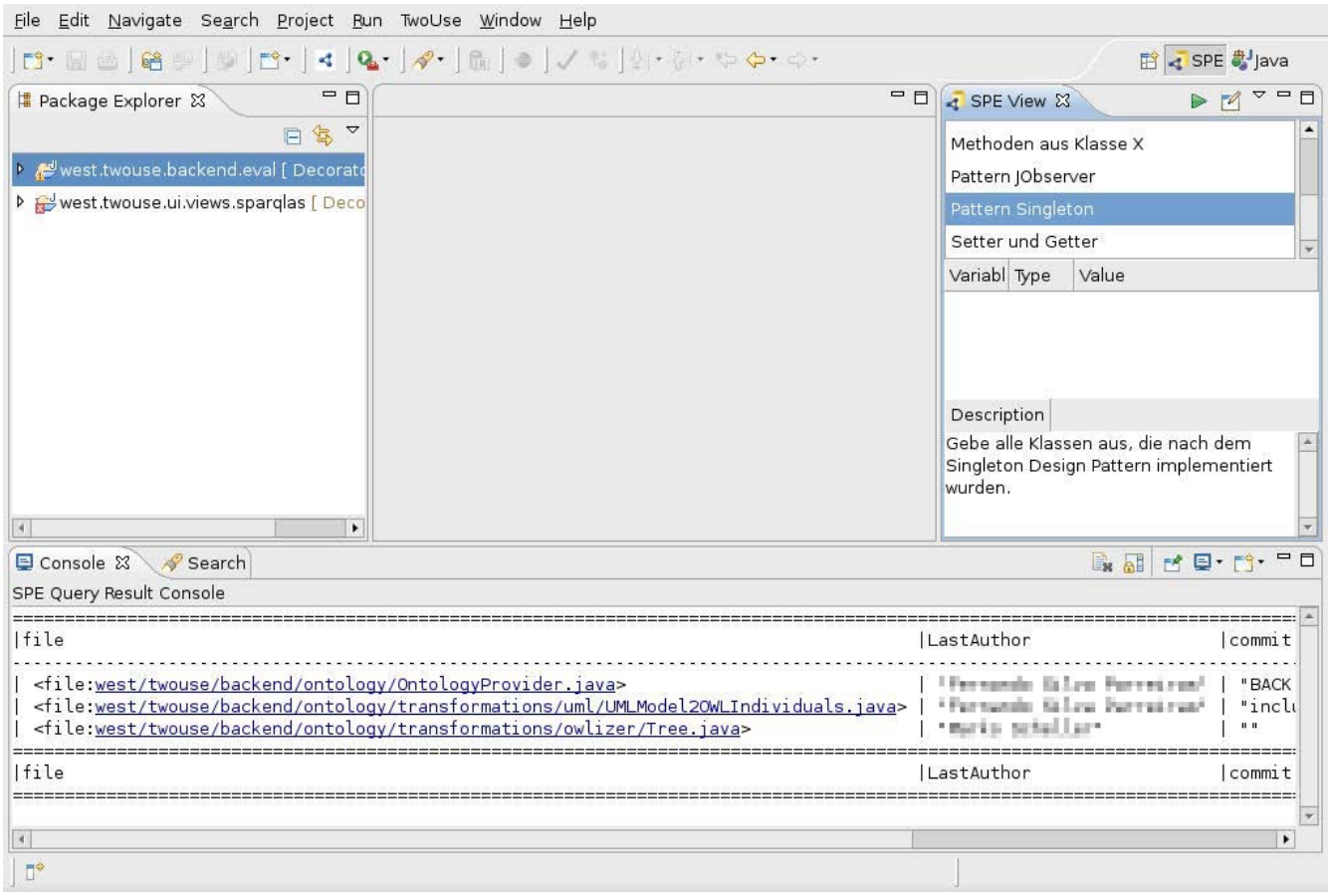

Figure 6. Screenshot of the User Interface of the SPE plugin: the updated list of software on the right-hand side; after choosing a design pattern, the console shows the list of classes in the source code that matches the design pattern on the bottom.

Listing 3. The result of the execution of the Query in Listing 2

```
Developer          | API Method
-------------------------------------------------------------
Grant Ingersoll    | AbstractJob.addOption(String)
Grant Ingersoll    | AbstractJob.getOption(String)
Sean Owen          | AbstractJob.addOption(String)
Sean Owen          | AbstractJob.getOption(String)
```

While we cannot directly determine the actual correctness of these answers, there is evidence that it is reasonable, given by the GitHub. Indeed, looking at the Github page of the Class that contains these two methods, these two developers are shown as its main contributors.

2) Who should you to talk to if you have to work with libraries you havent worked with?: This query takes as input a specific version of a library and returns the developers who made changes to code that includes adding references to entities present in the library (Listing 4). The rationale is that if a developer added a reference to the library in question to the code, s/he must be familiar with it.

Listing 4. A SPARQL Querying that matches the occurrences of the Singleton Design Pattern

```
PREFIX sm:<http://semanticminer/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX oslc_scm:<http://openservices.net/ns/scm#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX dcterms:<http://purl.org/dc/terms/>

SELECT ?developer (COUNT(?impact) AS ?total)
WHERE { ?changeSet a sm:ChangeSet.
?changeSet sm:commitCommiter ?developer.
?changeSet ^sm:contextTo ?impact.
?impact sm:impactOfAfter ?relationship.
?relationship rdf:value ?entity.
?entity ^sm:elements $libraryVersion.
FILTER NOT EXISTS { ?impact sm:impactType
    'NOTHING' } }
GROUP BY ?developer ORDER BY DESC(?total)
```

Since this query needs a Library as input, we chose a version of the Apache Lucene [5] library, since it is one of the libraries used by Apache Mahout. This library is used for the creation of applications that require "full text search" functionality. It is highly scalable and provides various search algorithms in its distribution. Listing 5 presents the result of this query with Lucene Core 3.5.0.

[5]http://lucene.apache.org/core/

Listing 5. The result of the execution of the Query in Listing 4

```
1 Developer
-----------
  Paritosh Ranjan
  Jeff Eastman
5 Robin Anil
```

VI. Related Work

The integration of software artifacts has been the topic of works including [16], [17]. These approaches present dedicated extractors for specific systems, e.g., for bug tracking and version control systems. Domain-specific declarations of rules for data integration are presented in [18]. For the model integration, there are also techniques based on traceability [19] in order to capture relationships of elements across model boundaries. In contrast to our work, neither of these approaches presents formats for publishing data suitable to the linked-data approach, i.e., they do not share the principles of interoperability for connecting federated software models across the Web.

Kiefer et al. [20] and Iqbal et al. [21] explore semantic web approaches for transforming software artifacts such as data from version control systems, bug tracking tools and source code into linked data. Both approaches use artifact-specific extractors and thus work only for a fixed number of software artifacts. Witte et al. [22] use semantic web technologies for the joint representation of source code and documentation in OWL. Based on the OWL representation, SPARQL queries are used to analyze software repositories. Our work proposes a more generic approach for transforming and managing controlled natural languages in an OWL representation.

A Linked Data platform for publishing data sets of software repositories is presented in [23]. The authors use RDF as representation formalisms and restrict OWL representations to object-oriented source code. The Linked Data framework in [24] provides source code representation in RDF. The source code repository is connected to Linked Data sources like DBpedia, Freebase and OpenCyc. Our approach presents a complementary solution that provides generic extractors and enables the development of services for visualization and analysis of software data.

Related to our research is the work on code search, where search engines like Google code search[6], Krugle[7], Koders[8] and the search engine of the sourcerer project[9] offer dedicated search facilities over large code bases. The proposed code search engine in [25] exploits an OWL representation of the source code. Like in our approach, the source code is extracted from repositories and afterwards transformed to OWL ontologies. A framework for querying for source code based on sample code snippets is presented in [26]. Query snippets are transformed to XPath queries. The source code is represented in the repository as abstract syntax trees. We use an approach based on expressful vocabularies using the OWL language.

Trustability for code search is presented in [27]. They specify a metric for trust of source code that incorporates the trustworthiness of persons (e.g., of the developer) that are associated with a program code. The Sourcerer search engine is extended by reputation of source code developers [28]. Thus, users might search for source code and chose their preferred source code by incorporating social characteristics like their opinion of others. We propose a marketplace that comprises the foundations for developing plugins that might also include trustability.

[6]http://code.google.com
[7]http://www.krugle.com
[8]http://www.koders.com
[9]http://sourcerer.ics.uci.edu

VII. Conclusion

In this paper we outline a marketplace for open source software engineering data and propose its architecture. The marketplace (1) maximizes the reuse of pieces of software by making them easy to find, analyze and integrate; (2) creates incentives for open source software producers, consumers and hosts and provide mechanisms for stimulating the creation of additional services; and (3) develops services demanded by software consumers that create value (e.g., mining, statistics, analytics, visualization and operation) over the linked software engineering data cloud. Its architecture is realized using semantic web technologies for the representation and management of loosely coupled software artifacts and their dependencies. Currently, we build on top of the Linked Data repository of source code proposed by [23] and focus on helping developers to make sense of massive data volumes, as described in Section V. For the future, we are working on plugins for fostering awareness and collaboration in global software engineering.

VIII. Acknowledgments

This work is partially supported by the Brazilian funding agencies FAPEMIG, CNPq and CAPES.

References

[1] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. T. Devanbu, "Fair and balanced?: bias in bug-fix datasets," in *Proceedings of the ESEC/FSE 2009, Amsterdam, The Netherlands, August 24-28, 2009*, H. van Vliet and V. Issarny, Eds. ACM, 2009, pp. 121–130.

[2] A. Bachmann, C. Bird, F. Rahman, P. T. Devanbu, and A. Bernstein, "The missing links: bugs and bug-fix commits," in *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*, G.-C. Roman and K. J. Sullivan, Eds. ACM, 2010, pp. 97–106.

[3] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data – The Story So Far," *International Journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22, 2009.

[4] W. O. W. Group, "OWL 2 Web Ontology Language Document Overview," W3C Working Draft 27 March 2009, 2009. [Online]. Available: http://www.w3.org/TR/2009/WD-owl2-overview-20090327//

[5] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework*. Pearson Education, 2008. [Online]. Available: http://books.google.de/books?id=sA0zOZuDXhgC

[6] T. Walter, F. S. Parreiras, G. Gröner, and C. Wende, "Owlizing: transforming software models to ontologies," in *Ontology-Driven Software Engineering*, ser. ODiSE'10. New York, NY, USA: ACM, 2010, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/1937128.1937135

[7] "Meta object facility (MOF) 2.0 core specification," OMG, Specification, 2003, version 2.

[8] J. Euzenat and P. Shvaiko, *Ontology matching*. Springer Berlin, 2007, vol. 18.

[9] S. Harris and A. Seaborne, "SPARQL 1.1 Query Language," W3C Working Draft 1 June 2010, 2010. [Online]. Available: http://www.w3.org/TR/2010/WD-sparql11-query-20100601/

[10] B. Glimm and C. Ogbuji, "SPARQL 1.1 Entailment Regimes," W3C Working Draft 1 June 2010, 2010. [Online]. Available: http://www.w3.org/TR/2010/WD-sparql11-entailment-20100601/

[11] E. Sirin and B. Parsia, "SPARQL-DL: SPARQL Query for OWL-DL," in *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions, Innsbruck, Austria, June 6-7, 2007*, ser. CEUR Workshop Proceedings, vol. 258. CEUR-WS.org, 2007.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994.

[13] F. de Freitas Silva, "A new approach for mining software repositories with semantic web tools," MSc Thesis. PUC Rio.

[14] T. Fritz and G. C. Murphy, "Using information fragments to answer the questions developers ask." in *ICSE (1)*, 2010, pp. 175–184.

[15] T. D. LaToza and B. A. Myers, "Hard-to-answer questions about code," in *Evaluation and Usability of Programming Languages and Tools*, ser. PLATEAU '10. New York, NY, USA: ACM, 2010, pp. 8:1–8:6.

[16] G. Antoniol, M. D. Penta, H. Gall, and M. Pinzger, "Towards the Integration of Versioning Systems, Bug Reports and Source Code Meta-Models," *Electr. Notes Theor. Comput. Sci.*, vol. 127, no. 3, pp. 87–99, 2005.

[17] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *Proceedings of the 22rd International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, FL, USA*. ACM, 2002, pp. 503–512.

[18] A. Königs and A. Schürr, "MDI: A Rule-based Multi-document and Tool Integration Approach," *Software and System Modeling*, vol. 5, no. 4, pp. 349–368, 2006.

[19] R. F. Paige, N. Drivalos, D. S. Kolovos, K. J. Fernandes, C. Power, G. K. Olsen, and S. Zschaler, "Rigorous identification and encoding of trace-links in model-driven engineering," *Software and System Modeling*, vol. 10, no. 4, pp. 469–487, 2011.

[20] C. Kiefer, A. Bernstein, and J. Tappolet, "Mining Software Repositories with iSPARQL and a Software Evolution Ontology," in *Proceedings of the 29th International Conference on Software Engineering Workshops (ICSEW '07), Minneapolis, MN, USA, May 20-26, 2007*, 2007.

[21] A. Iqbal, O. Ureche, M. Hausenblas, and G. Tummarello, "LD2SD: Linked Data Driven Software Development," in *Proceedings of SEKE 2009, Boston, MA, USA, July 1-3, 2009*. Knowledge Systems Institute Graduate School, 2009, pp. 240–245.

[22] R. Witte, Y. Zhang, and J. Rilling, "Empowering Software Maintainers with Semantic Web Technologies," in *4th European Semantic Web Conference (ESWC)*, ser. LNCS, vol. 4519. Springer, 2007, pp. 37–52.

[23] I. Keivanloo, C. Forbes, A. Hmood, M. Erfani, C. Neal, G. Peristerakis, and J. Rilling, "A Linked Data platform for mining software repositories," in *9th IEEE Working Conference on Mining Software Repositories (MSR)*, 2012, pp. 32–35.

[24] I. Keivanloo, C. Forbes, J. Rilling, and P. Charland, "Towards sharing source code facts using linked data," in *Proceedings of SUITE 2011*. ACM, 2011, pp. 25–28. [Online]. Available: http://doi.acm.org/10.1145/1985429.1985436

[25] I. Keivanloo, L. Roostapour, P. Schugerl, and J. Rilling, "SE-CodeSearch: A scalable Semantic Web-based source code search infrastructure," in *Proceedings of ICSM 2010*. Washington, DC, USA: IEEE Computer Society, pp. 1–5. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2010.5609533

[26] O. Panchenko, J. Karstens, H. Plattner, and A. Zeier, "Precise and scalable querying of syntactical source code patterns using sample code snippets and a database," in *19th IEEE International Conference on Program Comprehension, (ICPC)*, 2011, pp. 41–50.

[27] F. S. Gysin and A. Kuhn, "A trustability metric for code search based on developer karma," in *Proceedings of SUITE 2010*. New York, NY, USA: ACM, pp. 41–44. [Online]. Available: http://doi.acm.org/10.1145/1809175.1809186

[28] R. E. Gallardo-Valencia, P. Tantikul, and S. E. Sim, "Searching for reputable source code on the web," in *Proceedings of the 16th ACM international conference on Supporting group work*, ser. GROUP '10. ACM, 2010, pp. 183–186. [Online]. Available: http://doi.acm.org/10.1145/1880071.1880102