Towards an Ontology for Open Source Software Development

Gregory L. Simmons[1] and Tharam S. Dillon[2]
1 School of InformationTechnology and Mathematical Sciences, Univeristy of Ballarat, Australia, WWW home page: http://uob-community.ballarat.edu.au/~gsimmons
2 Faculty of Information Technology, University of Technology Sydney, Australia, WWW home page: http://staff.it.uts.edu.au/~tharam/

Abstract. Software development is a knowledge intensive process and the information generated in open source software development projects is typically housed in a central Internet repository. Open source repositories typically contains vast amounts of information, much of it unstructured, meaning that even if a question has previously been discussed and dealt with it is not a trivial task to locate it. This can lead to rework and confusion amongst developers and possibly deter new developers from getting involved in the project in the first place. This paper will present the case for an open source software development ontology. Such an ontology would enable better categorization of information and the development of sophisticated knowledge portals in order to better organize community knowledge and increase efficiency in the open source development process.

1. Introduction

Open source software (OSS) development provides an alternative model of development to commercial systems developed by or for a single corporate entity. In this model of development, a variety of developers carry out development and distribute the source code associated with the product. This allows for incremental improvement by others or development of complementary products that can seamlessly interoperate with the open source products.

Open source projects can be broadly characterized by their distributed development, loose management practices and their uncertain requirements [1, 2], these are considered briefly below:

- Distributed development teams: Open source developers are potentially drawn from a global pool of talent using the Internet; developers do not typically meet face to face. Rather the development community for any one project is centered on a public World-Wide-Web site and communication conducted using mailing lists and discussion forums.

Please use the following format when citing this chapter:

Simmons, G.L., and Dillon, T.S., 2006, in IFIP International Federation for Information Processing, Volume 203, Open Source Systems, eds. Damiani, E., Fitzgerald, B., Scacchi, W., Scotto, M., Succi, G., (Boston: Springer), pp. 65-75

- Loose management: There are no time constraints in an open source project and no mechanism to insist that functionality is implemented. Management is less concerned with utilizing resources efficiently and more concerned with which contributions should be committed to the product and which should be discarded.
- Uncertain requirements: Open source projects are constantly evolving with developers choosing to contribute what they think the product needs rather than the solution to any problem they are assigned, requirements are therefore elicited rather than assigned.

The community around an open source software project usually interacts through asynchronous textual modes of communication, such as email and threaded discussions, which are logged in publicly browsable World-Wide-Web repositories. The merits of proposed changes, requirements for the product, any problems are all debated in the open and archived along with the source code for the product.

Open source repositories serve to advertise the product, document its use, provide help to end users of the product, capture feature requests and bugs from users and developers, support developer collaboration and provide the entry point for new developers to accustom themselves with the project. Repositories are also the means by which users and developers upload and download the product in source and binary form. It is therefore not surprising that these repositories typically contain vast amounts of information.

The information contained within an open source repository serves as a record of the community knowledge accumulated throughout the development process and as such represents an artefact of vital importance. It is therefore unfortunate that the current open source software repositories in widespread use provide little support in terms of their ability to structure information so that it is meaningful to different types of user. Much of the information contained within open source repositories is unstructured, meaning that even if a question has previously been discussed and dealt with it is not a trivial task to locate it, leading to rework, confusion amongst developers and possibly deterring new developers from getting involved. Ankolekar, Herbsleb and Sycara [3] sum up this problem succinctly "there is a need to get the right information to the right person for the current task, and to present it in an understandable, usable way".

One approach to better understand and organize the structure of information from a particular domain is to use ontologies. Ontologies explicitly define a structure of concepts from a particular domain and their relationships to one another. Next generation (semantic) World-Wide-Web applications rely on meaningfully annotated content and often use ontologies to define their annotation vocabulary; with access to the underlying ontology we understand how to process the annotated content, and we have a basis for organizing the information into a meaningfully navigable hierarchy of terms.

The remainder of this paper is organized as follows. Section 2 presents a short description of ontologies and why they can be useful in open source software development. Section 3 introduces an ontology to describe open source software development. Section 4 discusses how such an ontology could be applied by proposing a software architecture for semantic portal development. Finally section 5 presents a brief discussion and conclusion.

2. Ontologies

Gruber [4] defines an ontology as "explicit formal specifications of the terms in the domain and relations among them". An ontology includes definitions of basic concepts in a domain and relations among them, these definitions are expressed in a machine-interpretable way allowing for the development of artificially intelligent applications. More importantly ontologies denote a shared conceptualization, for the ontology to be useful its specification must be one that is accepted in its use by domain experts.

Ontologies broadly contain Instances, Classes and Properties. Classes represent important concepts of the domain (these classes may be arranged in a taxonomy indicating superclass-subclass relationships between classes), properties represent a type of association between the domain concepts (which may or may not have restrictions) and instances represent an observed instance of a concept.

For example: An ontology about animals may state that a subclass of the concept Domestic-Animal called Domestic-Dog requires the properties color, breed, age and name. Furthermore you can place restrictions on concepts governing what definitions are legal or not, for example Domestic-Dog could have a restriction stating that all instances are quadrupeds therefore preventing any two-legged Domestic-Dog subclasses being defined. There may then be many instances of a Domestic-Dog, each describing a different four-legged animal such as the bull terrier known as Max and the retriever known as Rover, who both belong to the class Domestic-Dog.

Noy and McGuinness [5] provide five reasons for the development of an ontology:

1. To share common understanding of the structure of information among people or software agents
2. To enable reuse of domain knowledge
3. To make domain assumptions explicit
4. To separate domain knowledge from the operational knowledge
5. To analyze domain knowledge

Ontologies have been developed to describe everything from pizza[3] to wine [5] to cataloguing artefacts from a museum as displayed by the Museum of Finland website[4].

[3] `http://www.co-ode.org/ontologies/pizza/2005/10/18/`
[4] `http://museosuomi.cs.helsinki.fi/`

2.2 Open source development – A case for ontologies?

Despite its popularity a number of challenges exist with the potential to reduce the perceived benefits of open source development. One key issue for open source development is its scalability with its high dependence on source code as project documentation and its lack of formal documentation.

> "Complexity and size effectively close source code for system programming projects like OSes compilers after, say, 100K lines of code without good higher level documentation or participation in the project from its early stages. This "binarization" of source code in large system programming projects may mean that there is little strategic importance to keep the source code of system programs closed after it reaches a certain level of maturity."[6]

Another issue facing open source development is the scarcity of developers, a number of authors [7-9] has noted a Pareto distribution in the size of the number of developers participating in open source projects with the majority of projects having only one developer and a much smaller percentage with larger, ongoing involvement.

There is also a high degree of conceptual dissonance exhibited between open source projects, development models, licensing, source-code structure, terminology all differ markedly from project to project. The badge open source might suggest a collection of homogeneous projects but the reality is quite different and projects can differ quite markedly from the apparent bazaar style development in the Linux project as documented by Raymond [10] to the Extreme Programming influenced development evident in the Zope project [1].

It would seem obvious that a common understanding of how to the structure of information in open source repositories is something desirable. A common vocabulary could help reduce conceptual dissonance and provide budding contributors with easier access to information about a project than is possible at present. If a potential developer could easily access information about the source-code structure, the tools employed, the development model and the software license easily then perhaps the "binarization" of source code becomes less of a problem and developers would find it easier to join a development effort mid-stream.

In order to better organize the information generated in an open source project we need a conceptual framework that promotes agreement on how information should be organized, without losing any of the flexibility of allowing people to express and view parts in their own familiar expression language. Understanding the meaning of shared information on the web can substantially be enhanced if the information is mapped onto a domain ontology.

An open source software development ontology would encompass diverse, complex, domain knowledge, technology and skills. It will ensure a common ground for distributed collaboration and interactions. It is envisaged that such an ontology could be used as a basis for better organizing the community knowledge contained within open source repositories by providing the backbone for next-generation semantic open source development portals/repositories [11, 12]

3. An open source development ontology

This section presents the top level of a preliminary Open Source Development Ontology (OSDO). The OSDO would provide definitions of relevant classes and properties providing a unified vocabulary and structure for open source development. Each open source project would take the ontology and create instances reflecting the individual circumstances for that project. For example one project might contain the instance CVS for the class Version-Control whilst another project might have the Version-Control instance Subversion.

As with all ontologies the OSDO is a work in progress and the authors welcome any feedback. Due to space limitations it is not possible to present the entire ontology, rather the base concepts are presented along with some restrictions to demonstrate how the ontology could be reasoned with. A full version of the ontology is available from the author's website[5].

3.1 Ontology design

When designing a new ontology one needs design principles to guide development and provide a basis for evaluation, Gruber [13] identifies five design principles which should guide the development of ontologies:

1. Clarity – does the ontology effectively communicate its intended meaning?
2. Coherence – is the ontology logically consistent? "If a sentence that can be inferred from the axioms contradicts a definition or example given informally, then the ontology is incoherent."
3. Extendibility – ontologies should be designed in a way that allows for the definition of new terms for special uses without needing to redefine existing terms.
4. Minimum Encoding Bias – ontologies should be designed at the "knowledge level" rather than committing the ontology to a particular implementation language and its specific limitations.
5. Minimal Ontological Commitment – ontologies should make as few claims as possible about the domain being modeled without sacrificing the usability of the ontology.

3.2 Overview of the ontology

The first activity to be performed in any engineering activity is to decide upon the system's purpose and its intended uses, ontology engineering is no different in that we begin with specifying a number of *competency questions*, and *scenarios of use* [14].

[5] http://uob-community.ballarat.edu.au/~gsimmons

By establishing a series of competency questions we can determine the ontology's scope, and its applicability, competency questions also provide a means to evaluate an ontology.

An open source ontology designed with the intention to better organize community knowledge would need to be able to answer questions like; who performs the different tasks? how are the tasks performed? what tools are used? and so on. The following key competency questions can be identified:

1. What output is produced?
2. What activities are performed?
3. Who is responsible for performing the different activities?
4. What procedures need to be followed?
5. What tools are used?

These questions are by no means exhaustive but as they are used to initially scope the ontology and may be revised if later found to be missing. Once the scope of the ontology and its competency questions are identified relevant concepts and relations should be identified. This task can initially be performed using a *top-down* approach, where the most general concepts are identified and then broken down into specializations, or a *bottom-up* approach, which begins by defining specific concepts and groups them into related classes.

Using the competency questions as input, a top-down approach is used to discover the base classes (concepts). Table 1 presents the resultant six base classes for the OSDO along with their respective descriptions.

Table 1: OSDO Base Classes

Class	**Description**
Participant	Any person who uses or contributes to the project. Some participants may remain anonymous such as those that download and use the product but do not contribute in any other way.
Role	Represents in what capacity a participant was acting when they performed an activity in the project. There are some roles that may be assumed by any participant whilst only certain participants may assume other roles.
Activity	Any action that results in a contribution to the project or where the projects resources have been used in some way.
Procedure	Any established and well defined behaviour for the accomplishment on some activity.
Artefact	Any storable input to or output from an activity.
Tool	Any software resource used by a procedure in order to accomplish some activity.

Once defined these classes can be represented in a formal ontology language (such as RDF, DAML+OIL or OWL). We have chosen to implement our ontology using OWL-DL [15] as it is a dedicated ontology language with large-scale semantic web community support. The ontology was constructed in OWL using the Protégé[6] application.

The full ontology specification in OWL is omitted from this paper for sake of brevity but an example is provided as a means of illustration providing the OWL definition for the "Participant" class (Table 2).

Table 2 - OWL Definition

```
<owl:Class rdf:about="#Participant">
  <owl:disjointWith rdf:resource="#Role"/>
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="assumes"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#Role"/>
    </owl:Restriction>
  </owl:equivalentClass>
  <owl:disjointWith rdf:resource="#Procedure"/>
  <owl:disjointWith rdf:resource="#Tool"/>
  <owl:disjointWith rdf:resource="#Artefact"/>
  <owl:disjointWith rdf:resource="#Activity"/>
</owl:Class>
```

The base classes are further defined through a series of *property restrictions*. Restrictions are used to restrict the individuals that may belong to a class and enable us to *reason* with the ontology [16]. For example the class Participant is restricted with the existential restriction:

$\exists$ *assumes Role*

This states that any individual of the Participant class *assumes* at least one Role. Restrictions can be used to express complicated logic. The following restrictions define an Activity (a1) to be *preactivity* of Activity (a2) iff (a1) *produces* an Artefact (s) which (a2) *requires*.

[6] http://protege.stanford.edu/

$(\forall a, s)\ (produces(a, s) \rightarrow activity(a, *) \wedge artefact(s))$

$(\forall a, s)\ (requires(a, s) \rightarrow activity(a, *) \wedge artefact(s))$

$(\forall a1, a2)\ (preactivity(a1, a2) \leftrightarrow (\exists s)\ requires(a2, s) \wedge produces(a1, s))$

Once appropriate restrictions are defined for each of the base classes, defining sub-classes for each of Role, Activity, Procedure, Artefact and Tool can further extend the ontology. For example Role can be further broken down into either a *Consumer* or a *Contributor*. Consumers typically use the product but do not actively contribute to its development (other than promoting the product through its very use) and may often be anonymous; contributors however contribute directly to the product through source code development, project support, documentation, administration and so on. The Contributor role can therefore be broken down into a number of further specialized classes.

4. Putting it to work – An ontology driven architecture

Whilst ontologies are useful things in themselves, their real power can only be realized when applied to a broader application framework. In the case of the OSDO our motivation was to better organize open source project repositories. It is proposed that the OSDO could provide the basis for the development of a semantically aware project repository (or portal).

A number of semantic portals have been described in the literature including SEAL [11] and OntoViews [12]. In this section we propose an architecture (depicted in Figure 6) for a semantic portal based on the SEAL project.

The architecture consists of the following components:

- Semantic database – provides storage of semantic content and inferencing capabilities.
- Semantic query – querying facilities that exploit the inferencing capabilities of the semantic database and provides facilities such as semantic ranking.
- RDF generation – a facility to enable remote applications to interact at the RDF level.
- Template services –form generation for user input based on the reference ontology.
- Navigation – provides semantic linking and a dynamically generated portal structure.
- Annotation / Parsing – all new content is parsed against the reference ontology and semantically annotated before being stored in the database.

Each of the components of the architecture with the exception of the Annotator/Parser is present and well described in the SEAL project. To adopt a semantic portal for use in an open source project the addition of some form of automatic/semi-automatic annotation is a necessity because of the high likelihood of developers rejecting the requirement to manually annotate their contributions.

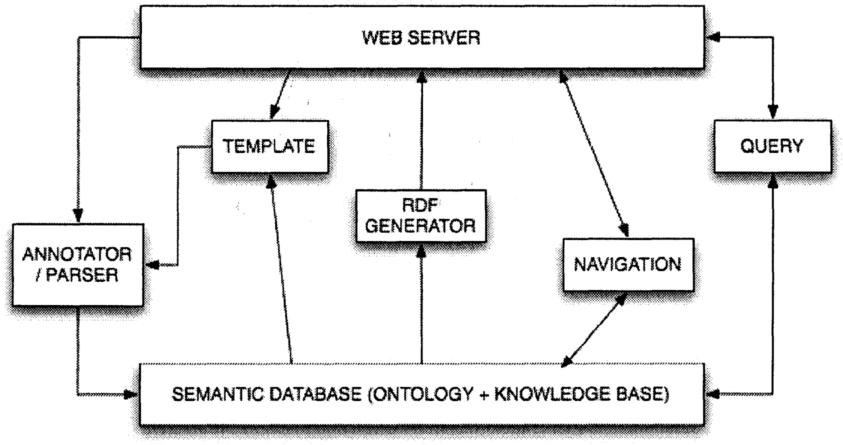

Figure 1: Ontology Driven Architecture

Take for example a bug report. Typically bugs are entered using a web form that requires the user to enter a bug description in free form text (perhaps a binary dump or screen shot) and some metadata (which may or may not be optional). The free form text can be parsed to identify terms known to the ontology and annotated accordingly whilst the metadata could be checked for consistency using the inferencing capabilities of the semantic database and if consistent annotated before being stored in the database for future reference. The problem of identifying duplicate bug reports and resolving incorrectly classified reports has been identified previously in the literature [17], semantically annotated bug reports could suggest possible duplicates via semantic query and ranking mechanisms thus aiding in this (largely manual) time consuming task. Semantic annotation could also allow bug reports could also be automatically emailed (or stored in a pigeon hole) to the responsible module maintainer or allow developers to identify a relevant discussion from a mailing-list archive, there are numerous possibilities for such a system.

5. Conclusion

Software development is well established and well understood in practice. However, distributed open source software development spread over multiple sites using open software for collaboration is a new challenge. The challenge is to develop a conceptual meta-model that will provide the architecture for the collaboration of distributed software teams and better supports the software development.

The problem of knowledge management in open source software development has been identified in the literature by a number of authors [3, 17, 18], however we note there has been no previous attempt at using an ontology based approach to address knowledge management in open source software development.

This paper presents the case for an ontology for open source software development, the proposed ontology is intended to be a starting point for discussion and adaptation rather than precise definition. All ontology engineering is iterative and collaborative and the authors welcome any comment on what is presented herein.

There are many possibilities for further research. The authors intend to further refine the ontology and to validate it using data from live open source projects. The architecture proposed needs to be implemented and validated using real data. Indeed the use of semantic portals in applications such as the one proposed and the continuing evolution of web portal technology provide numerous potential research opportunities.

Importantly the proposed ontology will provide practitioners with a basis for developing semantic web services in order to better organize community knowledge in open source development projects. Such web services have the potential to increase the efficiency of open source development and to make open source projects more accessible to those developers who would like to contribute to a project but are discouraged by the high barriers to entry.

References

1. Simmons, G. and T.S. Dillon. *Open Source Development and Agile Methods.* in *The 7th IASTED International Conference on Software Engineering and Applications.* 2003. Marina del Rey, CA, USA: ACTA Press.
2. Simmons, G. and T.S. Dillon. *A Critical Comparison of Agile Methods and Open Source Development through a Case Study.* in *International Conference on Software and Systems Engineering and their Applications.* 2003. Paris, France.
3. Ankolekar, A., J. Herbsleb, and K. Sycara. *Addressing Challenges to Open Source Collaboration With the Semantic Web.* in *Taking Stock of the Bazaar: The 3rd Workshop on Open Source Software Engineering, the 25th International Conference on Software Engineering (ICSE).* 2003. Portland OR, USA.
4. Gruber, T.R., *A Translation Approach to Portable Ontology Specification.* Knowledge Acquisition, 1993. **52**(6): p. 1111-1133.
5. Noy, N.F. and D. McGuinness, *Ontology Development 101: A Guide to Creating Your First Ontology*, S.K.S. Laboratory, Editor. 2001, Stanford Knowledge Systems Laboratory.
6. Bezroukov, N., *A Second Look at the Cathedral and the Bazaar.* First Monday, 1999. 4(12).

7. Hars, A. and S. Ou. *Working for free? - Motivations of participating in Open Source Projects.* in *The 34th Hawaii International Conference on System Sciences.* 2001.

8. Hunt, F. and P. Johson. *On the Pareto Distribution of SourceForge Projects.* in *Open Source Software Development Workshop.* 2002. Newcastle, UK.

9. Madey, G., V. Freeh, and R. Tynan. *The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory.* in *American Conference on Information Systems.* 2002. Dallas, TX.

10. Raymond, E.S., *The Cathedral & the Bazaar.* 2 ed. 2001, Sebastapol, CA: O'Reilly.

11. Maedche, A., et al., *Semantic portal - the SEAL approach.* 2001, Institute AIFB, University of Karlsruhe, Germany.

12. Mäkelä, E., et al. *OntoViews - A Tool for Creating Semantic Web Portals.* in *The Semantic Web - ISWC 2004.* 2004. Hiroshima, Japan: Springer.

13. Gruber, T.R., *Towards principals for the design of ontologies used for knowledge sharing.* Internation Journal of Human-Computer Studies, 1995. **43**: p. 907-928.

14. Gruninger, M. and M.S. Fox. *Methodology for the Design and Evaluation of Ontologies.* in *IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing.* 1995. Montreal.

15. McGuinness, D.L. and F.v. Harmelen, *OWL Web Ontology Language Overview.* 2004, W3C.

16. Falbo, R.A., C.S. Menezes, and A.R. Rocha. *Using Ontologies to Improve Knowledge Integration in Software Engineering Environments.* in *World Multiconference on Systemic, Cybernetics and Informatics / 4th International Conference on Information Systems Analysis and Synthesis.* 1998. Orlando, USA.

17. Gasser, L., et al. *Understanding Continuous Design in F/OSS Projects.* in *International Conference on Software and Systems Engineering and their Applications.* 2003. Paris, France.

18. Scacchi, W., *Understanding Requirements for Developing Open Source Software Systems.* IEE Proceedings - Software, 2002. **149**(1): p. 24-39.