View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by espace@Curtin

©2008 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

2008 IEEE International Conference on Signal Image Technology and Internet Based Systems

Semantic Web support for Open-source Software Development

Tharam S. Dillon[1], Gregory Simmons[2]
[1]Digital Ecosystems and Business Intelligence Institute, Curtin University of Technology, Australia
[2]School of Information Technology and Mathematical Sciences, University of Ballarat, Australia
t.dillon@cbs.curtin.edu.au, g.simmons@ballarat.edu.au

***Abstract*-Open-source software is unique in that the development of the product is performed in public over the Internet by developers who elect to contribute to the project and rarely if ever meet face-to-face. 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, leading to rework, confusion amongst developers and possibly deterring new developers from getting involved.

This paper develops an ontology based software development architecture for open-source software development. Such an architecture would enable better categorisation of information, communication, co-ordination and the development of sophisticated search agents.

I. 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 uncertain requirements, distributed development and loose management practices [1, 2]. 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 around a public World-Wide-Web site and communication conducted using mailing lists and discussion forums. 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. 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.

Open-source projects generate massive amounts of information; this information is usually housed in Internet repositories which typically provide little support for structuring it in a way that is meaningful to the heterogeneous needs of the open-source community.

This paper is concerned with knowledge management in open-source software development projects. Previous authors [3, 4, 5] have noted how open-source projects are poorly organised in relation to how they store their information, this research proposes a method for storing open-source related information more systematically without requiring developers to change their current habits. With this goal in mind this paper investigates the following question:

How can semantic web technology be leveraged to enable open-source software development to be more efficient?

In order to answer this question it is necessary to develop software which supports open-source software development whilst utilising semantic web technology. Before this can be realised however there must a common understanding of the structure of information among people or software agents, to enable reuse of domain knowledge and to make domain assumptions explicit. As a first step towards achieving this shared conceptualization this paper develops an ontology for Open Source Software Development. Such an ontology would enable better categorisation of information and the development of sophisticated search agents, providing the basis for a next generation open-source repository which better caters for the needs of different users. The paper also applies the ontology to a proof-of-concept semantic portal designed for use in open-source development in order to demonstrate its utility.

II. Open-source Development Methodologies

One of the first authors to describe the open-source development process was Eric Raymond in his paper "The Cathedral and the Bazaar" [6]. Raymond's contribution is particularly important due to its widespread adoption by the open-source community as the defacto manifesto for open-source development. In the paper Raymond contrasts

978-0-7695-3493-0/08 $25.00 © 2008 IEEE
DOI 10.1109/SITIS.2008.114

IEEE computer society

conventional software engineering practices utilising tightly co-ordinated, centralised teams, following rigorous development processes (which he labels cathedral-style development) to a bazaar-style development where no particular development approach is mandated and developers are free to develop what they wish in their own way. It is argued that this chaotic style of development leads to a greater exploration of the problem space in that it is consistent with an evolutionary principle of mutation and survival of the fittest, in so far as the best solution is likely to be incorporated into the evolving software product [7].

It is important to note that despite the incredible popularity of the Cathedral and Bazaar, a number of authors have been critical arguing the Bazaar metaphor is too simplistic.The black and white picture painted by Raymond (monolithic, authoritarian Cathedral model vs. democratic, distributed Bazaar model) is too simplistic. These metaphors for high centralization (Cathedral) and no centralization (Bazaar) do not account for the size of a given project; its complexity, timeframe and time pressures; its access to resources and tools; and, whether we are talking about core functionally (like Linux kernel) or peripheral parts of the system [8]

The success of open-source software has led to companies adopting it for use and developing open-source software themselves, companies such as Sun Microsystems, Netscape and IBM currently sponsor large open-source development projects. Two such projects, the office productivity suite OpenOffice.org and the Mozilla web browser, are notable in their size and complexity, and the development process for each differs significantly from the traditional "bazaar" style development described by Raymond. As the economic models for open-source are still relatively immature a number of companies have attempted to "borrow" the best features of open-source development whilst keeping proprietary control over their product's code, whilst not strictly open-source these "closed" imitations point to the significance of the open-source phenomenon. Amongst these hybrid development models are Microsoft's Shared Source [9] program and the Corporate Source program as applied at Hewlett-Packard [10]. The emerging phenomenon of corporate sponsored open-source project requires a development methodology which allows the sponsor to retain much of the control whilst continuing to elicit contributions and stimulate development from the open-source developer community. Striking a balance between corporate and community control is a difficult proposition and would be made easier if articulated in an appropriate software development methodology, however the literature is conspicuously absent in this regard.

III. Open-source lifecycle

Open source web portals such as Sourceforge[1] and Freshmeat[2] classifies an open source project into different stages of development as follows:

1. Planning - no code written, the project is just a proposal. Once code is written the project enters the next stage.
2. Pre-Alpha - Some source code available, the code is not expected to compile or run. The code might be confusing for outside observers and may lack coherence.
3. Alpha - Code works for some configurations, beginning to take shape. Development notes begin to appear. New features are rapidly being added. As soon as the volume of new features begins to decrease the project enters its *Beta* stage.
4. Beta - The code is deemed feature complete but is not error free. Once the number of faults is deemed low enough, the project releases a stable version.
5. Production/Stable - The software can be depended upon for daily use. Any changes are applied carefully, and the intent of changes is to increase the products stability not to add new functionality. If no significant changes are required over a long period of time the project enters the *Mature* stage.
6. Mature - There is little or no development occurring but the project continues to be maintained.
7. Inactive - The software ceases to be maintained.

Whilst Sourceforge might be happy to accept projects in the *planning* and *pre-alpha* stages it is clear that for an open source project to succeed it needs to be carried through to the *alpha* stage and produce a runnable prototype before the development community will get involved with the project. Eric Raymond observes "It's fairly clear that one cannot code from the ground up in bazaar style. One can test, debug, and improve in bazaar style, but it would be very hard to originate a project in bazaar mode Your nascent developer community needs to have something runnable and testable to play with." [6].

Thus most open source projects begin their life when a prototype is introduced to the community and made available under an open source complaint license and the *planning* and *pre-alpha* stages are replaced by the development of a prototype essentially in a closed fashion. Furthermore the Sourceforge classifications do nothing to describe the evolutionary nature of open source development. Figure 1 describes the lifecycle of an open source project illustrating how development and stable branches to the codebase are used to facilitate evolutionary development. When the development code is deemed stable it is packaged for release and split off into the maintenance branch where it becomes the current stable release, the newly released code is then used as the development codebase for the next set of features and therefore forms the initial "alpha" code for the next

[1] http://www.sourceforge.net
[2] http://www.freshmeat.net

production release. Any bug fixes required for a stable release are applied and merged back into the development branch as needed.

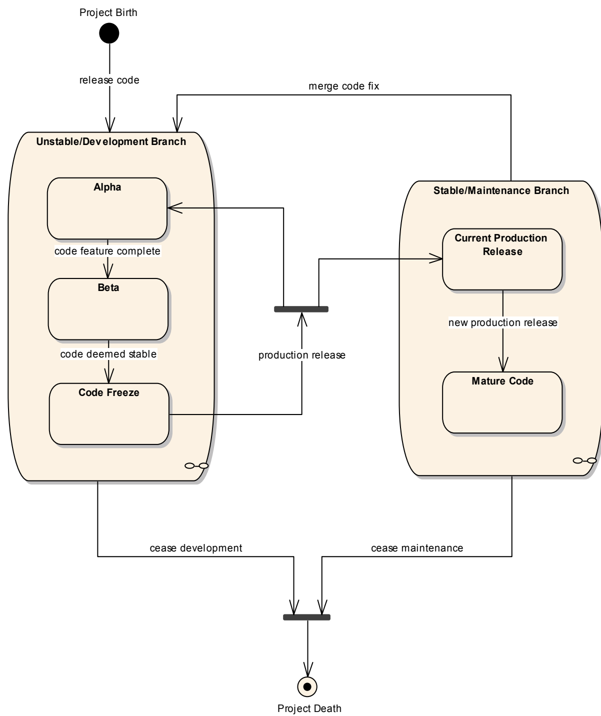

Fig. 1. Open source lifecycle

IV. Coordination in Open-source

Coordination in commercial software development is usually achieved through the use of a variety of mechanisms both explicit and implicit. Explicit mechanisms include such things as interface specifications, processes, plans, staffing profiles, and reviews. Implicit mechanisms include knowledge of who has expertise in what area, as well as customs and habits about how things are done. Ad-hoc or informal communication can also be used to overcome coordination problems, however as the complexity of the coordination increases the utility of informal communication decreases. When software development is performed using geographically dispersed developers the problem of coordination becomes more complicated. Mockus et. al. consider the Apache HTTP server project as an example of coordination in an open-source development project [11]. The Apache approach to coordination can be summarized as follows:

- A small core team responsible for creating the vast majority of new functionality coordinate their work using informal communication and implicit mechanisms
- In order to join the core group, candidates must demonstrate expertise in a needed specialist area and commitment to the project
- A larger group of people contribute bug fixes which are reviewed and acted upon by the core group
- A much larger group test the code through using it and submit problem reports when encountered
- There is no formal requirements process, developers are themselves the end users and feature selection is based on what the developers themselves deem appropriate
- Work is not delegated; individuals select what work they will do.

A number of limitations can be identified with this approach to coordination:

- This approach to coordination works well with a small core team but the reliance on informal communication becomes a liability as the size of the team increases.
- The size of the project is obviously also constrained by the size of the core team, as all changes must be approved and committed by a member of the team.
- It is also essential for developers to be users because of the absence of a requirements process.

Many open-source software projects are kept deliberately small with related functionality developed as independent open-source projects which interact through a well defined interface. This ensures the project does not become of a size that is unmanageable for the core team of developers. This small team requirement runs counter to Raymond's "bazaar" style development metaphor and is not appropriate for open-source projects which are sponsored by large organizations and to commercial variations on the open-source development model such as Hewlett Packard's corporate source model.

A. Roles in open-source development

Open source projects begin their lives when the project *founder* releases the project under an open source compliant license, at this time the project founder is entrusted with the *guardianship* of the project until such time as they hand over control to a new *guardian*.

Involvement in an open source project can be broadly catergorised as *passive* or *active*. Passive participants are consumers of the product and have no input into the development process, active participants contribute to the process in a variety of ways. Each participant in an open-source software project may assume many roles. Roles can be classified as *external* to the project (untrusted) or *internal* to the project (trusted). External active participants can contribute to the project by suggesting features, submitting bug reports, contributing to discussion forums, creating documentation or submitting patches. Internal active participants may maintain the project repository, be

responsible for reviewing contributions, prepare bug fixes and implement new features, manage new releases, build the development tree periodically, and generally make decisions about the direction of the project.

V. Open-source Repositories

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".

VI. Towards an Ontology Based Open-source Development

In order to better organise the information generated in an open-source project we need a conceptual framework that promotes agreement on how information should be organised, 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.

Gruber [12] 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.

McGuiness and Noy [13] provide five reasons for the development of an ontology, namely :to share common understanding of the structure of information among people or software agent; to enable reuse of domain knowledge; to make domain assumptions explicit; to separate domain knowledge from the operational knowledge; to analyse domain knowledge

As previously stated open-source repositories store vast amounts of information whilst providing little support for its categorisation and retrieval. It would seem obvious that a common understanding of the structure of information in open-source repositories is something desirable.

An open-source software development ontology would encompass diverse, complex, domain knowledge, technology and skills 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 organising the community knowledge contained within open-source repositories and provide the backbone for a next-generation semantic open-source development portal/repository [14]. The semantic portal would be responsible for parsing newly entered documents and generating associations/links by comparing the parsed document against the ontology. This dynamic link generation results in a more intelligent resultant hypertext [15].

VII. 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* [16].

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 organise 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 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 I. OSDO Base Classes

Class	Description
Participant	Any person who uses or contributes to the project.
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.

TABLE 2. OWL Definition

```
<owl:Class rdf:about="#Paticipant">
   <owl:disjointWith rdf:resource="#Tool"/>
   <owl:disjointWith rdf:resource="#Artefact"/>
   <owl:disjointWith>
     <owl:Class rdf:about="#Procedure"/>
   </owl:disjointWith>
   <owl:disjointWith rdf:resource="#Activity"/>
   <owl:disjointWith rdf:resource="#Role"/>
   <rdfs:subClassOf
rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:someValuesFrom rdf:resource="#Role"/>
       <owl:onProperty>
         <owl:ObjectProperty rdf:ID="assumesRole"/>
       </owl:onProperty>
     </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>
```

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 [17]. as it is a dedicated ontology language with large-scale semantic web community support. The ontology was constructed in OWL using the Protégé[3] 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). The base classes are further defined through a series of *restrictions*. Restrictions are used to restrict the individuals that may belong to a class and enable us to *reason* with the ontology (Falbo, Menezes et al. 1998). For example the class Participant is restricted with the existential restriction:

$$\exists \textit{ assumes Role}$$

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*.

$$(\forall a, s)\ (\mathit{produces}(a, s) \rightarrow \mathit{activity}(a,*) \wedge \mathit{artefact}(s))$$
$$(\forall a, s)\ (\mathit{requires}(a, s) \rightarrow \mathit{activity}(a,*) \wedge \mathit{artefact}(s))$$
$$(\forall a1, a2)\ (\mathit{preactivity}(a1, a2) \leftrightarrow (\exists s)\ \mathit{requires}(a2,s) \wedge \mathit{produces}(a1,s))$$

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.

A. *Exploring the base classes*

This section further describes the open source ontology showing the proposed subclasses for each of the base classes. A number of figures have been produced displaying a hierarchy of concepts using the OWLViz[4] plugin for Protégé. OWLViz presents its concepts as ellipses with relationships marked as lines with hollow-headed arrows. The top-level concept in all OWL ontologies is defined as owl:Thing and all base classes presented in Section 3.2 are defined as immediate subclasses of owl:Thing.

1) *Role/Participant*

The Participant and Role classes define respectively who participates in an open source project and what parts they play in the development. Figure 2 displays a partial view of the Participant/Role hierarchy. Role can be further broken

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

[4] http://www.co-ode.org/downloads/owlviz/

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), 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 specialised classes namely Administrator, Developer, Manager, Documenter, and QualityAssurance roles.

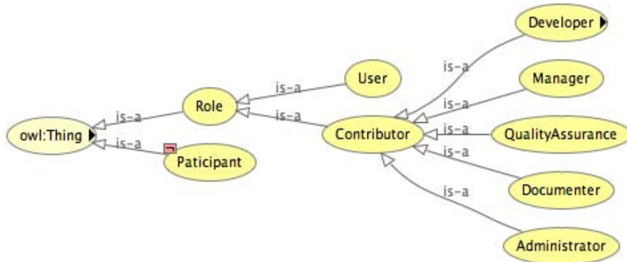

Fig. 2. Roles (concepts are ellipses, the "is-a" indicates indicates inheritance with all concepts derived from the top level concept owl:Thing)

Furthermore Developer can be broken down into those developers whose capabilities are trusted and have designated areas of development responsibility (InternalDeveloper) and those whose capabilities are unknown to the project (ExternalDeveloper). Some internal developers are granted extra responsibilities and may become a ModuleOwner .

2) Activity

The Activity class defines the tasks performed during the development of an open source project. Figure 3 displays a partial view of the Activity hierarchy. Activities can be broken down into different categories namely: AdministrativeActivity, ManagementActivity, ConsumeProduct, GenerateCode, GenerateDocumentation, or QualityControlActivity.

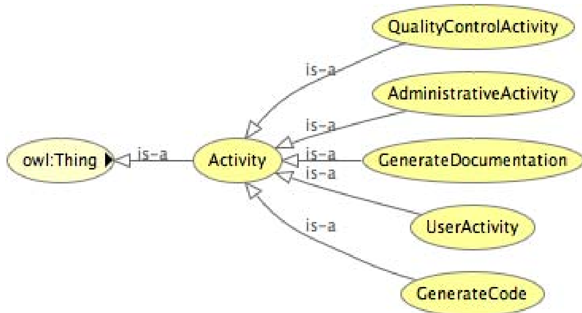

Fig. 3. OSDO Activities

3) Procedure

The Procedure class defines how activities are to be performed. Figure 4 provides a partial view of the Procedure hierarchy. Procedures can be broken into the following categories: AccessControlProcess, BackupProcess, DefectManagemenrProcess, CodeApprovalProcess, DocumentationProcess, ReleaseManagementProcess, RequirementsProcess, TestingProcess, or VersionControlProcess.

4) Artefact

The Artefact class defines what the development process produces. Figure 5 provides a partial view of the Artefact hierarchy. Artefacts are organised into Code and Document classes. The Code class is further broken down into Build, Module or Patch. Build can then be broken down again into Promotion or Release. Finally Release can be categorised into DevelopmentRelease or StableRelease. The Document class can be categorised into DefectReport, HelpDocument, License, Vision, or ReleaseDocument. Help documents can be further categorised as FAQ, Tutorial or HowTo. Whilst ReleaseDocument can be broken down into API, DefectList, ReleaseNotes, UserGuide, DevelopersGuide, or InstallGuide.

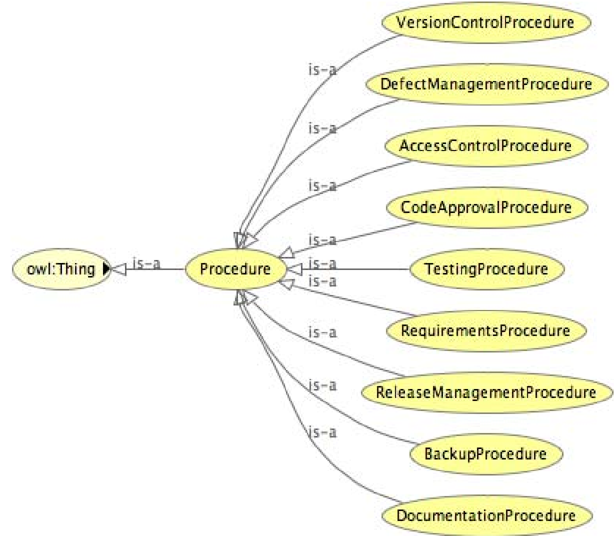

Fig. 4. OSDO Procedures

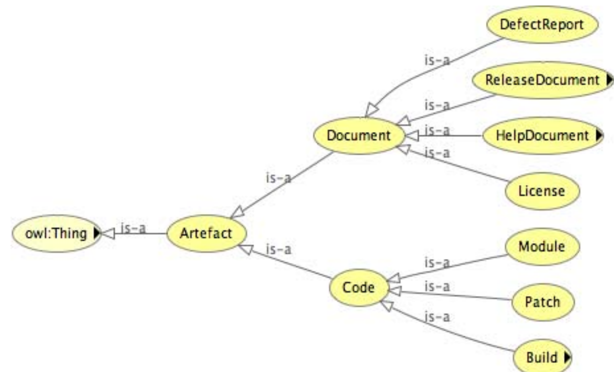

Fig. 5. OSDO Artefacts

5) Tool

The Tool class organises the different types of tool required to support development of an open source project. Figure 6 provides a partial view of the Tool hierarchy. Tools can be categorised as BackupSystem, AsynchronousCommunicationTool, SynchronousCommunicationTool, , ContentManagementSystem, DefectManagementSystem, SoftwareConfigurationManagement, or TestFramework.

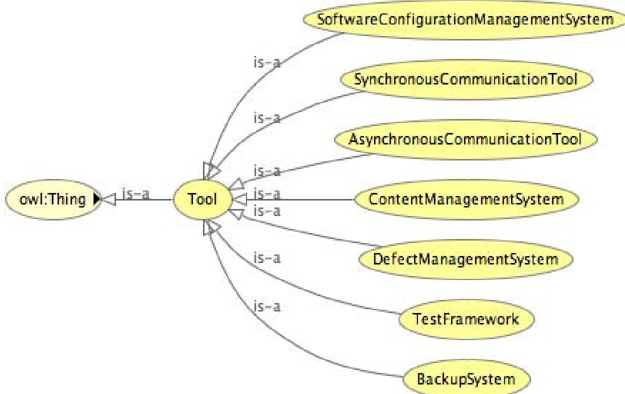

Fig. 6. OSDO Tools

VIII. AN ONTOLOGY DRIVEN ARCHITECTURE

Whilst ontologies are useful things in themselves, their real power can only be realised when applied to a broader application framework. In the case of the OSDO our motivation was to better organise 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 [18] and OntoViews [19]. In this section we propose an architecture (depicted in Figure 7) 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 which 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.

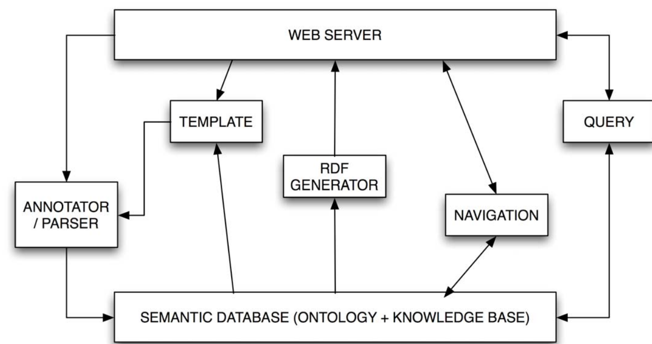

Fig. 7 Ontology Driven Architecture

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 [4], 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.

Ontology engineering is a highly collaborative process; it is of no use to develop an ontology which is mathematically precise but not accepted by domain experts. Due to the public nature of open-source software development the knowledge acquisition process can consist largely of the analysis and retrieval of the existing information stored in open-source repositories. Nevertheless it is vital that this ontology be promoted to the open-source and semantic web communities throughout stages of the development process in order to elicit feedback and agreement about the methodology employed for development of the ontology and the ontology itself, this will be done through the publication of journal

papers, conference presentations and the publishing of the results as an open-source project in itself.

IX. Conclusion

This paper presents an ontology for open source software development. The proposed ontology is intended to be work in progress for discussion and adaptation. All ontology engineering is iterative and collaborative and the authors welcome any comment on what is presented herein.

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 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] G. Simmons & T.S Dillon, "Critical Comparison of Agile Methods and Open Source Development through a Case Study," *Proceedings of the international conference on software and systems engineering and their applications*, Paris, France, 2003a.

[2] G. Simmons & T.S Dillon, "Open Source Development and Agile Methods," *Proceedings of the 7th IASTED international conference on software engineering and applications*, Marina del Rey, CA, USA, 2003b.

[3] A. Ankolekar, J. Herbsleb & K. Sycara, "Addressing Challenges to Open Source Collaboration With the Semantic Web Taking Stock Bazaar", *Proceedings of the 3rd workshop on open source software engineering*, 25th ICSE, Portland , USA, 2003.

[4] L. Gasser, W. Scacchi, G. Ripoche & B. Penne, "Understanding Continuous Design in F/OSS Projects," *Proceedings of the international conference on software and systems engineering and their applications,* Paris, France, 2003.

[5] W. Scacchi, "Understanding Requirements for Developing Open Source Software Systems," *IEEE Proceedings - Software*, vol. 149, no. 1, pp. 24-39, 2002.

[6] E.S. Raymond, *The Cathedral & the Bazaar* (2 ed.), Sebastapol, CA: O'Reilly, 2001.

[7] K. Kuwabara, "Linux: A Bazaar at the Edge of Chaos," *First Monday*, vol. 5, no. 3, 2000.

[8] N. Bezroukov, "A Second Look at the Cathedral and the Bazaar," *First Monday*, vol. 4, no. 12, 1999

[9] Microsoft, *Shared Source Licensing Programs*. Retrieved 1/2/2005, 2005

[10] J. Dinkelacker & P. Garg, "Corporate Source: Applying Open Source Concepts to a Corporate Environment," *Proceedings of the 1st workshop on open source software engineering, the 23rd international conference on software engineering*, Toronto, Canada, 2001.

[11] A. Mockus, R.T. Fielding & J. Herbsleb, "A Case Study of Open Source Software Development: The Apache Server," *Proceedings of the 22nd international conference on software engineering*, Limerick, Ireland, 2000.

[12] T.R. Gruber, "A Translation Approach to Portable Ontology Specification," *Knowledge Acquisition*, vol. 52, no.6, pp. 1111-1133, 1993.

[13] N. F. Noy & D. McGuinness, *Ontology Development 101: A Guide to Creating Your First Ontology (No. KSL-01-05)*, Stanford Knowledge Systems Laboratory. 2001

[14] G. Simmons & T.S. Dillon, "Towards an Ontology for Open Source Software Development," *Proceedings of the second international conference on open source systems*, Como, Italy, 2006.

[15] C. Goble, S. Bechhofer, L. Carr, D. Roure & W. Hall, "Conceptual open hypermedia = the semantic web," *Proceedings of the WWW2001, Semantic Web Workshop*, Hongkong, 2001.

[16] M. Grunninger & M. S. Fox, "Methodology for the Design and Evaluation of Ontologies," *Proceedings of the IJCAI-95 workshop on basic ontological issues in knowledge sharing*, Montreal, August 19-20th 1995.

[17] D.L. McGuinness & F. V. Harmelen, "OWL Web Ontology Language Overview," W3C, 2005

[18] A. Maedche, S. Staab, N. Stojanovic, R. Struder & Y. Sure, "Semantic portal - the SEAL approach," Institute AIFB, University of Karlsruhe, Germany, 2001.

[19] E. Mäkelä, E. Hyvöne, S. Saarela & K. Viljanen, "OntoViews - A Tool for Creating Semantic Web Portals," The Semantic Web - ISWC 2004, Hiroshima, Japan, 2004.