Carnegie Mellon University
Research Showcase @ CMU

Human-Computer Interaction Institute | School of Computer Science

2006

Supporting Online Problem-Solving Communities with the Semantic Web

Anupriya Ankolekar
University of Larlsruhe

Katia Sycara
Carnegie Mellon University

James Herbsleb
Carnegie Mellon University

Robert Kraut
Carnegie Mellon University

Chris Welty
IBM

Follow this and additional works at: http://repository.cmu.edu/hcii

This Conference Proceeding is brought to you for free and open access by the School of Computer Science at Research Showcase @ CMU. It has been accepted for inclusion in Human-Computer Interaction Institute by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

Supporting Online Problem-Solving Communities with the Semantic Web

Anupriya Ankolekar
University of Karlsruhe
76131 Karlsruhe, Germany
anupriya@aifb.uni-karlsruhe.de

Katia Sycara, James Herbsleb, Robert Kraut
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
{katia, jdh, kraut}@cs.cmu.edu

Chris Welty
IBM Watson Research Center
P.O. Box 704, Yorktown Heights, N.Y. 10598, USA
welty@us.ibm.com

ABSTRACT

The Web plays a critical role in hosting Web communities, their content and interactions. A prime example is the open source software (OSS) community, whose members, including software developers and users, interact almost exclusively over the Web, constantly generating, sharing and refining content in the form of software code through active interaction over the Web on code design and bug resolution processes. The Semantic Web is an envisaged extension of the current Web, in which content is given a well-defined meaning, through the specification of metadata and ontologies, increasing the utility of the content and enabling information from heterogeneous sources to be integrated. We developed a prototype Semantic Web system for OSS communities, Dhruv. Dhruv provides an enhanced semantic interface to bug resolution messages and recommends related software objects and artifacts. Dhruv uses an integrated model of the OpenACS community, the software, and the Web interactions, which is semi-automatically populated from the existing artifacts of the community.

Categories and Subject Descriptors

H.5.3 [**Information Systems**]: Group and Organization Interfaces; H.4.3 [**Information Systems**]: Communications Applications; H.3.6 [**Information Systems**]: Library Automation

General Terms

Design

Keywords

human-computer interaction, computer-supported cooperative work, open source software communities, semantic web applications

1. INTRODUCTION

Online professional communities have flourished in conjunction with the rise of the Web. Online professional communities are communities of people, who organize themselves and interact primarily through the Web, for work and knowledge sharing. Online professional communities constantly generate, share and refine information through active interaction over the Web as part of various community activities. Online professional communities can be either discussion-oriented communities, which are formed for knowledge sharing, such as Photo.net [24], or artifact-oriented communities that come together to create artifacts, such as online encyclopædias [26] and software programs [21]. In the former, interactions lead to the implicit capture of knowledge in the community Web discussion forums [14], whereas in the latter, interactions consist of decentralized, collaborative refinement of the artifacts being created. As people interact with each other in the community, their contributed information, knowledge and experiences get stored in the online community archive. Similarly, collaboratively built artifacts and the knowledge and experiences of the construction process also become a part of the community's content. The archived content thus becomes a valuable resource for current and future members of the community, a kind of collective good [20]. As the archive grows through community interactions, it becomes increasingly valuable for the community. For discussion-oriented communities, the archive is in fact the primary product of the community.

The most prominent example of online professional communities is probably open source software (OSS) communities, which form around the source code of a software program. Despite the complex and interdependent nature of software development [16], OSS communities have been remarkably successful, with several OSS software projects comparing very favorably with commercial offerings. Some of the most prominent OSS communities are the Linux operating system [18], the Apache software collection [1] and recently, the Firefox web browser [10]. OSS communities display characteristics of both discussion-oriented and artifact-creating communities, because the communities engage in a complex creation activity which requires a fair amount of coordination through discussion.

A major challenge for online professional communities is that they generate vast amounts of information as a result of their interactions, but that information is not well-linked on the basis of the meaning of its content. Thus, community members often find it difficult to coordinate and maintain awareness of other members' activities, leading to wasted

Copyright is held by the International World Wide Web Conference Committee (IW3C2). Distribution of these papers is limited to classroom use, and personal use by others.
WWW 2006, May 23–26, 2006, Edinburgh, Scotland.
ACM 1-59593-323-9/06/0005.

labor and reduced productivity.

The Semantic Web [2] has been proposed and promoted as an enhancement to the current Web. The Semantic Web vision adds a layer of machine-comprehensible *meta-data* over information on the Web and defines *ontologies* that describe the semantics of the meta-data. By thus enabling Web information to be automatically processed based on some representation of its content, Semantic Web technologies can make unstructured or semi-structured Web information meaningful. In the context of online professional communities, they facilitate the linking of community information and artifacts on the basis of their content and interpretation. This allows for intelligent support of online professional communities in terms of providing better awareness of community interactions and activities.

We believe that providing increased semantic support will be beneficial to online professional communities. By constructing a semantic model of the content, the interactions and the structure of the community, the activities of an online professional community can be supported. In this paper, we describe Dhruv, a prototype demonstrating the feasibility and utility of transitioning existing online professional communities to semantically-linked online professional communities. In particular, Dhruv supports bug resolution in open source software communities.

We first examine OSS communities and their three layers of content, interactions and community in detail in Section 2. We then discuss bug resolution as an example of problem-solving activities within OSS communites (Section 2.4) and describe some of its challenges, noting that the related work (Section 4) thus far in this area does not address the challenges for these communities. We then describe Dhruv as a prototype system supporting bug resolution in OSS communities (Section 3) and present an overview of its implementation (Section 5). Finally, we discuss some of the issues that arise for transitioning existing Web communities into the Semantic Web (Section 6) and end with a discussion of future directions (Section 7).

2. OPEN SOURCE SOFTWARE COMMUNITIES

Open source software development typically consists of a series of activities around the software code [22]. These include the discovery that a bug exists or that new functionality is needed, determining who among the pool of active developers will work on the issue, identifying a solution to the issue, developing and testing the solution, (if needed) presenting the code changes to the core committers for review, and committing the code and documentation to the repository. In fact, it has been suggested that most open source software projects operate in what would conventionally be regarded as the software maintenance and evolution phases of the software lifecycle [14]. The original code that seeds an OSS community is typically developed by individuals or by commercial software development teams and then contributed to the open source domain. It is rare for an OSS community to form without any seed code.

OSS communities can be viewed as having three layers (see Figure 1): a content layer of software code, bug reports, documentation etc.; an interactions layer that builds on the content, as people interact through bug tracking systems and web forums to participate in activities around the code, such as software development and bug resolution; and finally a community layer that is formed through the interactions that take place around the content. The community layer contains people and their various, dynamic roles in the OSS community.

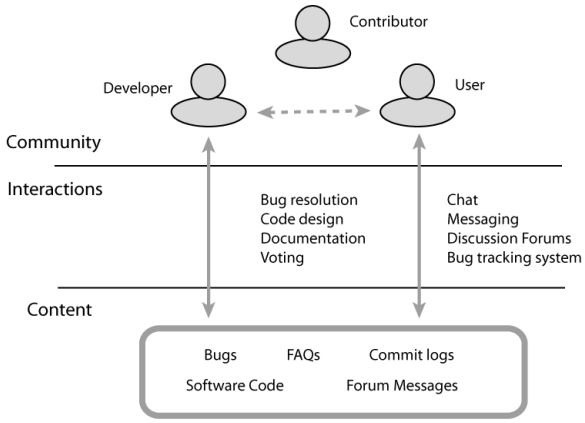

Figure 1: The Three Layers of Community, Content and Interactions in an OSS Community

In the following, we discuss each of the three layers of community, content and interactions in detail.

2.1 Content

The primary content of an OSS community is its software code. OSS communities typically form around an open source software program, a software program whose source code is publicly available. The communities' activities and interactions then revolve almost entirely around the program code, working to develop the program further, to fix bugs (defects in the software), to provide support on using the program, discuss future evolution of the software and so on. Essentially, the community performs the whole gamut of software development activities. In addition to the software code, OSS communities also create content in the form of bug reports about the software code, documentation and change logs.

2.2 Interactions

OSS communities are virtual communities and accordingly, interact primarily through the Web. They use a variety of communication tools, including mailing lists for technical discussions and support, a bug tracking system for monitoring and fixing bugs, a CVS code repository for storing a common version of the source code. In addition, communities often use chat for more real-time communication. Within the OSS community, there is a strong culture of 'making it public', i.e. conducting all community interactions, such as answering questions, discussing plans and design details and reporting on project status, in public locations, such as the community mailing lists and other discussion forums [13]. Most such explicit interactions in an OSS community are discussions about some bug or a design detail or the best way to implement a new feature and any person can freely participate in such discussions. Archives of all past activity is usually available through the community website and can be browsed. Thus, the OSS communi-

ties use the Web primarily as a communication and storage medium.

2.3 Community

OSS communities typically bring together a loose collection of volunteers: the active developers of the software, the end-users of the software, and anyone who has an interest in the software. They organize themselves as a community, creating roles for themselves (such as core developer, software contributer, end-user, etc.) and performing administrative functions, such as voting, marketing, etc., in addition to the main software development activities, such as source code design, code implementation, program maintenance etc. Open source software communities face the challenge of maintaining awareness of other developers so that they can coordinate their own work with others. Coordination becomes particularly important when developers are to work together over a long term. An activity-centric view then becomes crucial to providing adequate support, particularly since OSS communities have severely restricted communication compared to co-located software development: they communicate primarily through artifacts and email discussion.

2.4 Bug Resolution in Open Source Software Communities

In order to participate effectively within the community, OSS community members need *group awareness* [7], i.e. they need to understand who is working on what in the community and how their work affects other community members. Such knowledge allows people to coordinate work effectively, anticipate other members' actions, discuss tasks and locate help [12]. There are three primary ways that an OSS community maintains such group awareness [13]: reading developer mailing lists, reading real-time chat logs and watching commits from the code repository.

The group awareness problem is most apparent during bug resolution. Bug resolution is a very important activity for OSS communities since it determines the quality of the code produce. It is often also a precursor to an official release of the source code. Bugs and feature requests are usually tracked by means of an issue tracking system such as Bugzilla [4] or the OpenACS Bugtracker [23]. Bug fixes or patches are sometimes also submitted through the bug tracking system. The community is encouraged to use the bug tracking systems, since it then becomes a single access-point for all modifications to be made to the code. In most OSS bug tracking tools, each such bug becomes a message board centered on the issue [14]. However, bug fixing is not as exciting as code design and development. Consequently, it often tends to get neglected.

Part of the problem is that since the community has collective responsibility for fixing bugs, no person is explicitly assigned to any bug. Every now and then, for example, before a software release, people will go through the bugs posted on the bug tracking tool, looking for bugs they can fix. This places the burden of finding bugs that can be fixed squarely on the people who can fix them. Thus, there can be a long delay before the bug gets to the attention of developers who can fix the bug. Of course, if a community member is really keen to get a bug fixed, they can always specially request help with the bug on the discussion forums. Community members are generally quite responsive and try to help out as far as possible.

The bugs that do get submitted to the bug tracking tool may themselves not be genuine bugs. A common example of an invalid bug is one that cannot be reproduced. Determining that such bugs are invalid wastes developer time and clutters the bug tracking system.

Receiving duplicate or invalid bug reports is unfortunately quite common. In the Apache project, a few dedicated developers would usually go through the bug reports, mark duplicate bugs, remove mistaken bugs, fix simple bugs quickly, review and commit patches, and forwarding reports to the developer mailing lists if the bugs are considered critical [22]. However, not all OSS projects can afford dedicated developers to triage the bug reports and sorting through the bug reports remains a major problem.

The responsibility for fixing specific bugs can sometimes bounce between several developers or groups of developers before eventually being accepted. Sometimes developers will themselves bring bug reports to the attention of people, who can fix the bugs. Once the bug has been fixed or the enhancement developed, the bug tracking tool is searched for similar reports, so that those can also be closed.

To support community members in the normal process of bug resolution, the Semantic Web should support answering the kinds of questions that normally arise in in the minds of developers in the midst of bug resolution. Two kinds of questions that often arise when developers attempt to understand a program are 'what' and 'why' questions [17]. The 'what' question represents questions of the form 'What is this software object' or 'What does this software object do?'. The 'why' question represents questions about the purpose and rationale for sections of source code, such as 'Why is this fragment of code implemented in this particular way?'. Dhruv can support both these kinds of questions in the context of bug report comments. The former by presenting the definition of the object and its cross-links. The latter by providing cross-links to the discussions about the software object.

Both 'what' and 'why' questions are posed as soon as the unfamiliar software object is encountered [17]. Individual references to software objects in a bug report message are captured as code terms, as described in Section 3. To support the immediate answering of 'what' and 'why' questions, the extracted code terms and noun phrases themselves should be immediately clickable to get more information.

During the course of normal activities and interactions within the community, community members also tend to refer to other artifacts that explain a point or provide background information for it. Such links should be captured by Dhruv and presented meaningfully to community members. Inter-artifact linkage takes many forms:

- CVS commit logs refer to the bug reports that the commit fixes. They also often refer to a patch report whose patch is being committed. Less often, they may refer to forums discussions that contain the discussion that prompted the commit.

- Bug reports in the OpenACS Bugtracker have a slot for the patch reports that address the corresponding bugs. In addition, bug reports may refer to other bug reports that are either duplicates or report similar problems. Bug reports also refer to forum discussions, for example, when the bug was discussed on the forums prior to

Figure 2: A bug report in Dhruv, displaying the highlighted links and artifact and people recommendations for each bug report message.

filing an official bug report. Similarly, bug reports also tend to refer to code files when describing the problem experienced and the suspected offending files.

- Similarly, forum discussions may refer to files, bug and patch reports, other forum discussions and CVS commit logs, as the need arises.

- Patch reports and files have significantly less inter-artifact linkage, but they may also link to files, bug and patch reports, forum discussions and CVS commit logs.

3. DHRUV

In this section, we describe the creation of the community Semantic Web prototype, Dhruv. Dhruv performs two functions with respect to OSS bug resolution. First, it provides an *enhanced semantic interface* to messages posted during bug resolution. The enhanced interface allows community members to click on selected highlighted terms within the message, taking them to a *cross-links page*, which furnishes greater detail on the clicked term. The cross-links page primarily presents semantically related information about the term in the system and suggests related artifacts. Second, Dhruv provides a number of *message recommendations* of people, source code files, bug reports and discussions for each bug report message. These recommendations are determined by taking into account the semantic cross-links of each of the highlighted terms in the message.

Dhruv has been constructed for the OpenACS/dotLRN [23] OSS community. The OpenACS/dotLRN community has formed around the open source OpenACS (Open Architecture Community System) toolkit, for building scalable, community-oriented web applications. OpenACS provides the foundation for many web applications, including the open source dotLRN e-learning platform [6], and many websites, including Greenpeace [11].

Figure 2 shows a bug report in the Dhruv system with

Figure 3: Cross-links page for a code term

highlighted terms in the text of a bug report message and artifact and people recommendations for each bug report message. The links highlighted gray-blue lead to more information on the highlighted text, namely the information in the cross-links page for the selected term. The lightly colored links indicate that this term is a special kind of link, so that community members can click on it and delve deeper into that concept if they wish to. However, the color is muted, so that people reading the bug report may ignore them and are not distracted by the links.

The cross-links page, as shown in Figure 3, lists various kinds of information about a code term, `ad_page_contract`, both the semantic in the knowledge base and the cross-links of the term in the community semantic web. Since there are two kinds of terms: code terms and noun phrases, we have two separate kinds of pages for each. The noun phrases capture technical keywords used by the community, such as `folder hierarchy`, whereas the code terms capture software object references, such as the software procedures `db_transaction` and `lang::util::localize`. Figure 3 displays the cross- links page for a code term, where Figure 4 shows the cross-links page for a noun phrase. Other kinds of extracted metadata, such as file names, which have corresponding semantic information in the knowledge base, are treated in the same way as code terms.

Structured metadata drawn from the knowledge base is displayed in the left column, while the right column displays related artifacts of various kinds. The related artifacts are categorized into different kinds of artifacts, such as code files

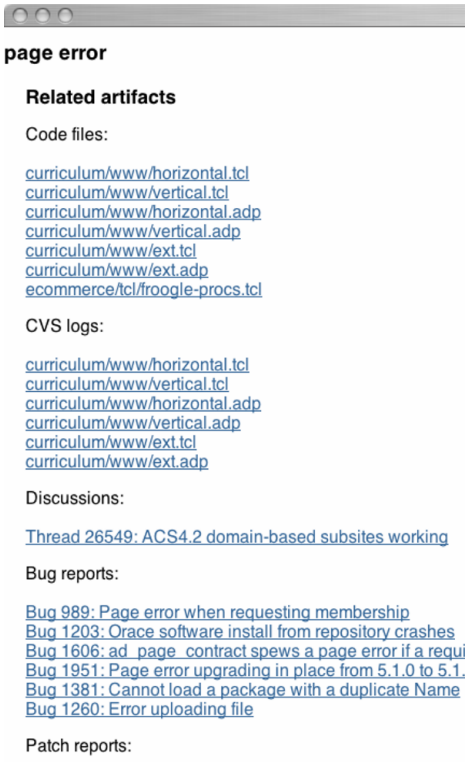

Figure 4: Cross-links for a noun phrase

and bug reports, because different artifacts carry different kinds of information about a given code term that a bug fixer might need.

Together, the highlighted links and cross-links pages comprise an enhanced interface to Dhruv's semantic data for individual terms. In addition, Dhruv also presents artifact recommendations for each message. When a community member comments on a bug report, Dhruv produces recommendations of related artifacts for the comment and appends them to the message, as depicted by Figure 2. Message recommendations represent key artifacts for the message as a whole, based on the cross-links of individual terms. They can represent a best guess for new developers, who are unsure where to start with the bug report. If a artifact is clearly related to the entire message, then the message recommendations ought to capture it and present a short-cut for developers looking at the bug report.

Figure 2 shows the message recommendations given by Dhruv. These recommendations are classified into several categories, just like the related artifacts for a term. The rationale is the same: to let people easily get to the different kinds of information represented by different artifacts.

4. RELATED WORK

One of the first systems to explore the idea of expressing additional semantics of software code to assist software developers was the Software Information System (SIS), developed by Brachman et al. [3]. A SIS indexes the software source code and stores relationships that are frequently searched for by software developers during software maintenance. Such a system, while useful, would be inadequate in the context of OSS communities. Supporting bug resolution in OSS communities also requires a great understanding of the semantics of the content of developer messages and the context of the project and the actions of developers. The community semantic web essentially performs the same functions as an SIS, but is much broader in scope, encompassing a greater variety of information, such as bug reports and discussion messages.

An example closer to the open source software development context is Hipikat [5]. Hipikat is an Eclipse plug-in [9], which builds a group memory from all the artifacts in an open source project. Then, someone viewing the code or bugs is presented with information considered similar using information retrieval techniques. Hipikat differs from the approach presented here in that it focusses on the single-developer process of solving a bug. Our focus is on the multi-developer bug resolution processes that take place beyond the actual fixing of the bug, in particular the coordination required of multiple developers to fix the bug.

From the Semantic Web research area, there are a few applications of the Semantic Web variously to email and personal information [19], personal information management [25] and for browsing through marked-up Web content [8] and e-learning. Haystack [25] is an end-to-end Semantic Web application for personal information management. The concept is similar to that of the community semantic web, but it focusses on managing the personal information of a single user. Magpie [8] is a browser that exposes the semantic metadata of a Webpage to the end-user through right-clicks. Mangrove [19] is a 'semantic email system'. By augmenting email with simple metadata (e.g. yes/no for replies), an agent can process multiple emails and summarize the replies. To our knowledge, there are no Semantic Web applications that address problem-solving in Web communities.

In addition, most of the above related work (save Haystack) assumes the existence of domain ontologies and present tools to process information on their basis. However, there is a research gap in the application of Semantic Web to a specific domain, especially a dynamic domain constantly changing due to interactions of the Web community. Although individual components of the Semantic Web are well-developed, there is a need for more application knowledge of the Semantic Web. How can the Semantic Web be used by a Web community? What does it take to create a Semantic Web for an existing community? How does the introduction of the Semantic Web change the interactions of the community? How do community interactions enrich the Semantic Web itself? This paper attempts a first shot at addressing some of these questions.

5. IMPLEMENTATION OF DHRUV

There are several requirements to realize the community semantic web sketched thus far. To begin with, we need a way to describe the semantics of information on the Web, namely the bug reports, discussion messages, documentation, source code files and commit logs. This is achieved by means of ontologies to describe community artifacts and metadata to populate the ontology TBoxes. Given metadata about information in the OSS community artifacts, we then need a way to relate metadata about different artifacts to each other. In other words, the Semantic Web needs to be able to express the relationships between various artifacts

through their metadata. By expressing metadata and their relationships in a machine-processable manner, we enable the automatic processing, classification and presentation of the artifacts. In addition, in order to interpret the semantics of community interactions, we require natural language processing to process the communication artifacts of a community. Finally, we need information retrieval techniques to retrieve communication artifacts when supporting bug resolution.

5.1 Ontologies

To begin with, Dhruv requires an ontology that describes the structure of the project and provides a basis for determining how artifacts are related. We model all three layers of content, interactions and community of OpenACS. The content layer is modeled in two ontologies, the code ontology and the bugs ontology. These ontologies enable us to determine the location and context of given software objects and related bug reports. Essentially, these ontologies help identify the semantic context of a bug report message. The interactions ontology describes the structure of interactions around bug reports, files and discussions in an OSS community. Using the interactions ontology, Dhruv can identify people who are experts in the area of the bug. Finally, the community ontology describes the various roles in the community. Knowing the roles of people in the community, Dhruv can recommend people appropriate to their roles in the community. Thus, someone who has expertise in the area of the bug, but has never modified a file or submitted a patch, is likely to have the in-depth knowledge of the code required for fixing a bug. In the following, we give an overview of each of the ontologies. The complete ontologies can be found at the indicated URL.

1. *Code Ontology*: The code ontology is the core component of the OpenACS ontology and models the software source code. Given that all activities within an open source software community are primarily centred around the software source code, the code ontology is the most critical portion of the OpenACS ontology. All bugs occur somewhere in the source code and the ensuing discussion around the bug also takes place in referential context of the source code. The code ontology is therefore key to interpreting the semantics of other artifacts, especially message-based artifacts.

   The code ontology[1] models the software source code in terms of the contained software objects, such as modules, packages, functions, namespaces, variables and database objects. In addition, code packages and procedures are usually distributed across several files. In order to connect commits performed on files, to actual blocks of code, the ontology models storage structure of the code, i.e. the file(s) in which a procedure (or package) is located.

   Many of the concepts, such as `code:Function` and `code:Table`, do not model the real-world objects, such as functions and tables, completely. For example, they do not model columns of the table and return values of functions. However, this is inconsequential, since comprehensive modeling is not required by any of the reasoning tasks and is in fact detrimental to efficient use of the concepts. The purpose of the code ontology is therefore primarily to model the location, type and partof relations for significant code objects.

2. *Bugs Ontology*:

   The bugs ontology[2] is the other portion of the content layer. The bugs ontology models the information in the OpenACS Bugtracker: the bug reports, their attributes and the discussions around them. The bugs ontology is much simpler than the code ontology, containing 11 concepts and 15 properties. The core classes in the bugs ontology are `bugs:Report` and its two direct subclasses are `bugs:BugReport` and `bugs:PatchReport`. These represent a bug report and a patch report respectively, filed in the OpenACS Bugtracker. Other classes model the various attributes of bug and patch reports, such as status, priority, submitter and so on.

3. *Interactions Ontology*: On top of the code ontology and the bugs ontology, lies the interactions ontology[3], which models community interactions around artifacts. The interactions ontology therefore models people's actions on bug reports and on discussion forums. In the interactions ontology, interactions are modeled as follows: individual artifacts or interaction items may have `interaction:Message`s made by instances of the class `community:Person`. Each message is differentiated into several types, for example, the subclass `interactions:OpenMessage`, which refers to the action performed on the artifact by posting the message.

   An interaction item represents an artifact, an instance of `community:Resource`, around which an interaction may take place. The class `interactions:InteractionItem` has three direct subclasses `interactions:DiscThread`, `bugs:Report` and the class `code:File`. The classes `bugs:Report` and `code:File` have been discussed previously and support interactions in the form of discussions around bug reports and commit sequences on files respectively. The class `interactions:DiscThread` represents the set of all discussion threads in the OpenACS Forums.

   Bug report comments are modeled because they contain valuable information that elaborates on the bug, its symptoms, possible fixes and trade-offs. Such information is very useful, for example, to a developer wishing to fix a similar bug. In addition, bug comments indicate the people who participated in the resolution of the bug and are therefore likely to have some expertise in the area of the bug. Similarly, comments or messages posted on web-based discussion forums are also modeled in the interactions ontology, since they may contain information that a developer needs to be aware of while fixing bugs.

4. *Community Ontology*: On top of the interactions ontology, lies the community ontology[4], which models the online community in terms of the people taking part in the community interactions around artifacts

[1] `http://www.cs.cmu.edu/~anupriya/code.owl`

[2] `http://www.cs.cmu.edu/~anupriya/bugs.owl`

[3] `http://www.cs.cmu.edu/~anupriya/interactions.owl`

[4] `http://www.cs.cmu.edu/~anupriya/community.owl`

and their roles. The community ontology therefore captures the semantics of the community layer. There are three key classes in the community ontology: the class `community:Community`, which contains numerous `community:Resource`s and a number of instances of class `community:Person`.

The relatively simple structure and limited number of defined classes is intentional. Initial versions of the ontologies were much richer and more fine-grained, but the large number of instances that arise naturally from the modelling details of the code structure and the numerous interactions made reasoning computationally demanding. We therefore adopted a simpler ontological structure in order to be able to reason over the entire community corpus.

5.2 Generating Metadata

Having defined classes and roles in the OpenACS ontology, namely the TBox, we can now consider how to define the instances that populate these classes, namely the ABox. These instances are gathered from the artifacts, namely source code files, CVS commit logs, bug and patch reports and discussion threads. The defined ontologies now need to be populated by metadata, for which Dhruv needs to identify meaningful terms and concepts within the community artifacts, namely the source code and the interaction messages of the community. There are two kinds of metadata: (a) references to code, files, packages, error traces, other bug reports and discussions, and (b) semantic concepts expressed as a technical vocabulary or jargon that is meaningful to community members. These extracted terms are highlighted by Dhruv within the message.

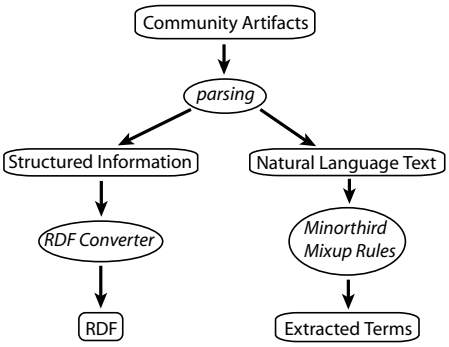

Figure 5: An overview of the metadata generation process

A schematic of the metadata generation process is shown in Fig. 5. The metadata generation process uses a combination of hand-written parsing rules and information extraction patterns. We attempted to generate the instance data automatically as far as possible. There are several reasons for this. Firstly, it would be infeasible to define all the instances in the ontology manually. This is true for any ontology that aims to capture any sizable domain. Although it requires some upfront effort to teach a system like Dhruv to acquire data automatically, once the system is functioning, the incremental effort of data generation on the ontology developers and users is negligible.

By restricting ourselves to automatic instance acquisition, we also limit ourselves to capturing information that is high in structural semantics, but may be relatively shallow in its interpretational semantics. For the purposes of this exploratory study, though, this restriction is not severe. Important instances not captured automatically can always be added manually to the ABox. Naturally, the decision to acquire instances automatically has influenced the design of the TBox to an extent, in that only those concepts and properties were retained, whose instances could be determined automatically.

5.3 Cross-Links

Having generated metadata for various kinds of software objects and community artifacts, Dhruv can now use the ontology relations created in the first step to identify the semantic context of the metadata generated in the previous step. This context consists of metadata from the ontologies and related artifacts as determined by text similarity. We call the artifacts identified by the context of an extracted term the cross-links for the term. The cross-links for each extracted term in the message is presented in a cross-links page. The highlighted terms and cross-links pages together represent the enhanced semantic interface provided by Dhruv. We used various heuristics for generating cross-links for terms. In particular, we identified artifacts and objects from noun phrases, from code terms and from references to other artifacts using information extraction techniques. A code term

Given a code term identified through information extraction, we build cross-linkages in two ways:

1. by identifying the type of software object it refers to and
2. by treating it as a noun phrase and using textual similarity to identify related resources.

An example of cross-linkages for a code term is shown in Figure 3 and for a noun phrase in Figure 4. Different kinds of cross-linkages are created for a code term depending on the type of software object in question, using numerous heuristics. We used heuristics such as the file a software object has been found in is an important part of the context for that software object; software objects that are semantically related are often given similar names; connecting namespaces to the functions they contain and functions to the variables they contains are valuable indicators of their function. For example, the names of variables often provide more insight into what a function does short of actually reading the definition of the function.

In order to identify people with related expertise, we examine their co-authorship of artifacts. If people tend to participate in the same bug reports and discussions and modify the same code files, albeit across different periods of time, then it is quite likely that they will have similar expertise. Thus co-authorship of artifacts has implicit in it shared expertise. If we consider every two people who co-author an artifact to have interacted, then the co-authorship heuristic essentially determines the professional 'social networks' of the community.

5.4 Recommendation Heuristics

Given the semantic context or cross-links for message terms, Dhruv can now generate message recommendations. Dhruv

gathers all the cross-links for a message and then prunes and ranks the list of artifacts using a number of heuristics. Dhruv utilizes the cross-links determined in the previous section to generate recommendations for each message in a bug report. The recommendations for each message are generated on the basis of the content of the message, i.e. its extracted metadata, namely code terms, noun phrases and artifact references. The artifacts referred to by the artifact references and by cross-links from the extracted metadata are accumulated and then pruned to generate the recommendations. Each type of cross-link is given a recommendation weight to reflect its importance in determining the recommendations. Artifacts are weighted according to the cross-link that picked them. Two different cross-links may suggest the same artifact. In this case, the artifact weights are summed up to give a new recommendation weight to the artifact. Finally, the recommendations are ranked according to descending weight and the top n recommendations are finally presented to the user.

In addition to the cross-links determined in the previous section, Dhruv uses a couple of heuristics specifically for generating bug report message recommendations. For example, artifacts that are cross-linked to by metadata captured from the bug report summary are given a higher recommendation weight, since the summary of a bug report is often the most informative text in the bug report. Human-endorsed artifacts that appear within a bug report message itself, are likely to be more reliable than the artifact references generated by Dhruv. Therefore, such within-message artifact references are given a high recommendation weight. Each bug report message adds a certain modicum of information to the bug report. The semantics of the bug report can only be determined by examining the entire bug report. Thus, in order to give recommendations for the most recent bug report message, Dhruv also utilizes the metadata of all previous messages in the bug report.

6. DISCUSSION

Dhruv is an initial prototype of a community semantic web for OSS communities. The creation of Dhruv relied on light-weight processes that parsed existing web content and transformed it into semantic web content without interfering with the natural activities of the community. Making the transition from the Web to the Semantic Web as seamless as possible is an important requirement for Semantic Web applications.

A major obstacle in the creation of Dhruv is the large amount of data that is generated by the OSS community. The reasoners we tried to use within this work, namely Racer and Pellet, were unable to reason efficiently for large data. If the metadata input to the reasoners is inconsistent, then identifying which statement in the hundreds of statements caused an error to be flagged is itself an non-trivial task. This points to a huge gap for the Semantic Web. Reasoning about information on the Web necessarily brings with it the specter of huge data. Due to this, Dhruv could not take advantage of the full expressivity of description logics provided by OWL DL. Dhruv relied on cached information to generate its pages.

There is one system that does address the problem of efficient reasoning for large numbers of individuals: instance Store (iS) [15]. The iS system stores assertions about individuals and their types in a database, reducing reasoning over individuals to terminological reasoning. However, the current version of iS is limited to role-free reasoning of individuals, i.e. the ABox may have no axioms asserting role relationships between individuals. During the creation of Dhruv, this was deemed to be a major limitation. However, ultimately the primary use of ontologies in Dhruv is for the description, annotation and retrieval of large numbers of individuals. In hindsight, iS was probably the most appropriate system to use in Dhruv.

Dhruv does not make use of the open world assumption nor does it make use of ontologies distributed over multiple sites. As a small, self-contained community semantic web, Dhruv does not require them. It is simpler to implement Dhruv as a closed world with stable ontologies for individual communities. This also simplifies the ontology descriptions and reasoning. An ontology language like OWL then becomes primarily useful in linking up the ontologies of individual communities and enabling interoperation among them.

Within the current implementation of Dhruv, the inferencing and classification capabilities of the Semantic Web are underutilised. This is purely because current reasoners do not scale well to the numbers of objects dealt with by Dhruv. There are several possible uses for classification within Dhruv. A prime example is using inference to classify people into roles, such as bug fixer or core developer. The people recommendations can then use this role information to suggest appropriate people. For example, inactive members or people who have never participated in a bug resolution are unhelpful people recommendations for a bug report.

Inter-artifact links provide surprisingly useful information to Dhruv. Inter-artifact links are formed when the community explicitly links various artifacts together during the natural course of their interactions. These semantic cross-links are also rich enough to be useful in predicting the artifacts that are likely to be useful for a bug report. Inter-artifact links can be easily captured and represented by the Semantic Web beyond OSS communities, such as in discussion-oriented online communities. Exposing semantic information to people such that they can browse the semantic links between artifacts is likely to be meaningful to people, rather than merely using the semantic information to compute related artifacts. The strength of the Semantic Web is its power to provide explanations for why certain artifacts are related or recommended. Within collaborative work contexts, such explanations can be as useful as the final answer.

A key feature of the Dhruv community semantic web prototype is that it is built from the 'outside'. In other words, it represents a way to take an existing online OSS community and transform it into a semantic community. This contributes to the ease of deployment for the Semantic Web and reduces the adoption barrier for online communities.

The ontologies in Dhruv are fairly general and domain-independent. They can be used for a community semantic web by any OSS community built around an OpenACS website. Modeling other OSS communities will require some modification of Dhruv, primarily in the processes used to gather semantic metadata in OpenACS. Various components of Dhruv, such as the metadata extraction and the heuristics for determining cross-links are independent modules that can be extended depending on the need. The recommendations of Dhruv can potentially be improved by plugging in

state-of-the-art tools and techniques for these modules.

The current version of Dhruv represents a initial step in the realization of a community semantic web. By modeling individual communities and community processes more closely and by developing more specific ontologies, we can improve on the current version to provide comprehensive and tailored support to online professional communities.

6.1 Lessons Learnt

The most important lesson learnt as a result of this work is that the Semantic Web needs ways to handle large amounts of data. The Semantic Web reasoning infrastructure will need to deal with significant amounts of data on the Web and current reasoners for OWL are easily overwhelmed with data. For this reason, we were not able to make full use of the semantic inferencing possibilities offered by the Semantic Web. Instead, we opted for a simpler, less expressive set of ontologies for modeling the OpenACS community. The fact that this simpler modeling also proved to be so valuable to the OpenACS community attests to the value of semantics and the lack thereof in current systems.

The Semantic Web also needs more domain-specific ontologies. Although there has been an effort to develop upper ontologies for several domains, these need to be linked to more specific ontologies or categories that are used within actual work domains. Thus, for Dhruv, we could not make use of any existing ontology. Instead, we needed to construct specialized ontologies for use in the context of the community work artifacts.

Contact with the OpenACS community members suggested they were keen on being involved in the improvement and extension of Dhruv. The community is technically sophisticated enough to understand how Dhruv works and to handle and maintain the explicit semantics used within Dhruv. This is an audience unlike that of typical ontology-based systems, where there is a sharp distinction between the users of the system and the knowledge engineers who design the system. Furthermore, systems that do not require developers to change their system or ways of working are well-regarded. The community appreciated the lightweight and non-intrusive nature of Dhruv in comparison to other knowledge management systems, citing it as a major factor in their enthusiasm for Dhruv.

7. CONCLUSIONS AND FUTURE WORK

The primary contribution of this research work has been to demonstrate the feasibility of capturing the semantics of a comprehensive set of artifacts to support bug resolution in OSS communities. In particular, we developed:

1. A model and knowledge base for the OpenACS community, their software content and interactions. The model can be generalized to any OSS community and to other online professional communities.

2. Tools and techniques to automatically extract and annotate Web information with respect to the ontologies mentioned above. In addition, we developed rules and heuristics to generate cross-links between OSS artifacts.

3. A framework for the use of the Semantic Web in the context of OSS development and more generally, for collaboration in web communities as well as in software development. We demonstrated the potential of a community Semantic Web for OSS communities.

With respect to research in the Semantic Web community, our work is the first to focus on supporting problem-solving in Web communities. Given that thriving Web communities have been integral to the success of the Web, it is imperative that the evolution of Semantic Web communities from Web communities be explored. However, this area of research has remain essentially unexplored until now. In this work, we demonstrated the transition of the content of an existing Web community, OpenACS, to Semantic Web content. In addition, we explored how the Semantic Web can support problem-solving interactions within Web communities by providing supporting information from the existing archive of the community.

Another contribution of this research in the Semantic Web area is to develop several domain-specific ontologies to model OSS communities. The ontologies describe various interaction artifacts of OSS development, such as bug reports, discussion threads and commit log information, and associated web community processes. The code ontology describes OpenACS code and can be used with slight modifications for other OSS community websites.

The primary next step for this work is performing a rigorous evaluation of Dhruv. There are several components to this: an evaluation of the metadata generation methods, of the recommendation heuristics and an evaluation of the functionality in the context of the OpenACS community. The most promising direction of future work is actual deployment of Dhruv in the OpenACS community. Initial contact with the community revealed that they are highly enthusiastic about Dhruv and ready to help to bring about the eventual use of Dhruv in the community. Deploying Dhruv in the OpenACS community will allow the community to get involved in the use, maintenance and future evolution of Dhruv.

Real-world deployment will also give us the opportunity to improve individual components of Dhruv. In this work, we focused on simple techniques to demonstrate a proof-of- concept. To take the prototype and transform it into a working system, we need to use more sophisticated state-of-the-art techniques for various components of Dhruv, such as metadata extraction and the generation of cross-links.

Dhruv is likely to be particularly useful for novice developers, who wish to participate more substantially in the community. By making it easier for newcomers to understand the context of a bug report and to explore the source code related to the bug report, Dhruv helps newcomers to participate in bug resolution more effectively and perhaps even develop fixes for the bug. This is likely to help attract new developers to the community and help compensate for the typical lack of documentation in OSS communities that turns away novice developers from a community and a code base. Future work to extend Dhruv to support newcomers more thoroughly and explicitly can have significant effect on OSS communities.

The concept underlying Dhruv is fairly simple and general: identify a structured portion of the semantics of interactions and attempt to support interactions by making the semantics explicit. There is huge scope for applying this concept to other contexts beyond bug resolution in OSS communities. The enhanced semantic interface provided by

Dhruv is likely to be directly useful in the task of code comprehension, which is a pre-requisite for bug resolution. The task of trying to understand the code is less tightly focused than bug resolution and involves a high degree of exploration of the links between software objects. Dhruvs enhanced semantic interface supports precisely this type of exploration and is therefore likely to be useful for code comprehension too.

Beyond OSS communities, there are other online professional communities that have a core of relatively structured content which is high in semantics. Obvious examples are educational communities and communities conducting scientific research. Both types of communities are likely to benefit from a system that supports current interactions in the community by making the interaction history of the community more transparent. By extending Dhruv to support both these types of communities, we can demonstrate the generality of the concept underlying Dhruv as well as make Dhruv itself more domain-independent.

8. REFERENCES

[1] Apache software foundation. `http://www.apache.org/`.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[3] R. J. Brachman, P. Devanbu, P. G. Selfridge, D. Belanger, and Y. Chen. Toward a software information system. *AT&T Technical Journal*, 69(2):22–41, 1990.

[4] The bugzilla bug tracking system. `http://bugzilla.mozilla.org/`.

[5] D. Cubranic and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings, International Conference on Software Engineering*, pages 408–418, Portland, OR, May 2003.

[6] dotLRN. `http://openacs.org/projects/dotlrn/`.

[7] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW)*, pages 107–114. ACM Press, 1992.

[8] M. Dzbor, J. Domingue, and E. Motta. Magpie – Towards a semantic web browser. In K. S. Dieter Fensel and J. Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 738–753, Sanibel Island, FL, USA, October 2003. Springer Verlag.

[9] Eclipse.org. `http://www.eclipse.org`.

[10] The firefox web browser. `http://firefox.mozilla.org/`.

[11] `http://www.greenpeace.org/international/footer/software-copyright`.

[12] C. Gutwin and S. Greenberg. A descriptive framework of workspace awareness for real-time groupware. *Journal of Computer-Supported Cooperative Work (JCSCW)*, 3-4:411–446, 2002.

[13] C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *Proceedings of the Computer Supported Cooperative Work (CSCW)*, Chicago, Illinois, USA, November 2004.

[14] T. J. Halloran and W. L. Scherlis. High quality and open source software practices. In *Meeting Challenges and Surviving Success: 2nd Workshop on Open Source Software Engineering, International Conference on Software Engineering*, Orlando, FL, May 2002.

[15] I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The instance store: DL reasoning with large numbers of individuals. In *Proceedings of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40, 2004.

[16] R. Kraut and L. Streeter. Coordination in software development. *Communications of the ACM*, pages 69–81, 1995.

[17] S. Letovsky. Cognitive processes in program comprehension. In E. Soloway and S. Iyengar, editors, *Proceedings of the First Workshop on Empirical Studies of Programmers*, pages 58–79, Washington, DC, USA, June 1986. Ablex Publishing Corporation.

[18] The linux operating system. `http://linux.org/`.

[19] L. McDowell, O. Etzioni, S. Gribble, A. Halevy, H. Levy, W. Pentney, D. Verma, and S. Vlasseva. Mangrove: Enticing ordinary people onto the semantic web via instant gratification. In K. S. Dieter Fensel and J. Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 754–770, Sanibel Island, FL, USA, October 2003. Springer Verlag.

[20] D. R. Millen. Community portals and collective goods: Conversation archives as an information resource. In *Proceedings of the 33rd Annual Hawaii International Conference on Systems Sciences (HICSS)*, Maui, Hawaii, USA, January 4-7 2000.

[21] A. Mockus, R. T. Fielding, and J. Herbsleb. A case study of open source software development: the apache server. In *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering*, pages 263–272, New York, NY, USA, 2000. ACM Press.

[22] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3), July 2002.

[23] OpenACS: Open architecture community system. `http://openacs.org/`.

[24] Photo.net. `http://www.photo.net/`.

[25] D. Quan, D. Huynh, and D. R. Karger. Haystack: A platform for authoring end user semantic web applications. In K. S. Dieter Fensel and J. Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 738–753, Sanibel Island, FL, USA, October 2003. Springer Verlag.

[26] Wikipedia. `http://wikipedia.org`.