Weaving a Semantic Web across OSS repositories: 
unleashing a new potential for academia and practice

Olivier BERGER
Institut TELECOM, SudParis
9 rue Charles Fourier 
91011 Évry Cedex, France
olivier.berger
@it-sudparis.eu

Valentin VLASCEANU
Institut TELECOM, SudParis
9 rue Charles Fourier 
91011 Évry Cedex, France
ion_valentin.vlasceanu
@it-sudparis.eu

Christian BAC
Institut TELECOM, SudParis
9 rue Charles Fourier 
91011 Évry Cedex, France
christian.bac
@it-sudparis.eu

Stéphane LAURIERE
Mandriva S.A.
55 boulevard Saint-Martin 
75003 Paris, France
slauriere
@mandriva.com

Quang Vu DANG
Institut TELECOM, SudParis
9 rue Charles Fourier 
91011 Évry Cedex, France
quang_vu.dang
@it-sudparis.eu

Introduction
This article is an extended version of the position paper
Weaving a Semantic Web across OSS repositories (a spotlight on
bts-link, UDD, SWIM) presented by the authors at the 4th
International Workshop on Public Data about Software
Development (WoPDaSD 2009), co-located with the OSS 2009
Conference (Skövde, Sweden).
The HELIOS project1 is a joint project between french academics
and industrials in the frame of the Paris area System@tic cluster
(under the “Libre software” thematics group), to build an open
source Application Lifecycle Management (ALM) platform.
Among other goals, HELIOS aims at addressing bugtracker
synchronization issues, and bug traceability2. To that purpose, the
potential of using Semantic Web technologies for navigating
between the many similar bugs filed in the different bugtracker
systems has been experimented, together with implementing
REST protocols for manipulation of bug reports by different client
tools.
The first section of this article introduces several open source
tools illustrating the need for semantically interconnected
databases and interoperability standards: bts-link, and UDD
(Ultimate Debian Database) developed by the Debian community,
and the Eclipse-Mylyn bugtracker client. The second section
discusses new use-cases we foresee for researchers and open
source practitioners, with the advent of more semantics in open
source related software engineering facts repositories. It will be
illustrated by recent initiatives like the LD2SD methodology from
DERI, doc4 developed by Mandriva, the OSLC-CM specifications
and finally our FetchBugs4.me project.
This will not constitute a detailed analysis nor the presentation of
all results achieved during the HELIOS project. The objective is
mainly to attract attention to novel interesting projects, and
present ideas that may trigger the interest of the research
community, and maybe receive useful comments on the way the
work done in the frame of HELIOS can be further shaped, and on

1 http://helios-platform.org/ - both Institut TELECOM and
Mandriva are among the partners funded in project.
2 A more detailed description can be found at:
https://picoforge.int-evry.fr/cgi-bin/twiki/view/Helios_wp3/Web/

ABSTRACT
Several public repositories and archives of “facts” about libre
software projects, maintained either by open source communities
or by research communities, have been flourishing over the Web
in the recent years. These have enable new analysis and support
new quality assurance tasks.
This paper presents some complementary existing tools,
projects and models proposed both by OSS actors or research
initiatives, that are likely to lead to useful future developments in
terms of study of the FLOSS phenomenon, and also to the very
practitioners in the FLOSS development projects, provided that
interoperability is fostered at all places.
A goal of the research conducted within the HELIOS project,
is to address bugs traceability issues. For that, we investigate the
potential of using Semantic Web technologies in navigating
between many different bugtracker systems scattered all over the
open source ecosystem.
By using Semantic Web techniques, it is possible to
interconnect the databases containing data about open-source
software projects development, hence letting OSS partakers
dedicated properties, collectively designing a distributed semantic
resource functions, annotate them, and further interlink them using
Such links expressed with standard Semantic techniques
are paving the way to new applications (including ones meant for
“end-users”). For instance this may have an impact on the way
the research efforts are conducted (less fragmented), and could also
be used by development communities to improve Quality
Assurance tasks.

Keywords
RDF, forge, archive, bug, semantic, semantic Web, ontology,
database, repository of repositories, interoperability, bugtracker,
OSLC-CM, Debian.
how the use of such tools by the open source communities can be maximized.

Finally, a word of caution: this paper does not enter into the details of the Semantic Web approach, that some qualify as “the next revolution of the Internet”; it just focuses on the progressive adoption of Semantic Web concepts in various services and tools adopting interoperable representations of data through the use of standards such as RDF, RDFa, OWL, microformats and others. The reader unfamiliar with Semantic Web concepts and techniques is advised to read the gentle introduction presented by Howison in [1].

1. Introducing several tools and services

We'll start by introducing the reader to some key projects that have been developed by the FLOSS communities recently, that will illustrate the need for more interconnected databases, and more interoperability between tools, that may be addressed with the help of Semantic Web techniques.

1.1 Navigating the net of bugs: bts­-link, Debian's bug links watcher

Open-source GNU/Linux distributions such as Debian or Mandriva are composed of thousands of assembled packages (downstream) providing software which have been developed within hundreds of independent projects (upstream).

Each GNU/Linux distribution usually maintains a central bugtracker (for instance, Debian's debbugs which is available at http://bugs.debian.org/, or Mandriva's bugzilla running at https://qa.mandriva.com/) that is open to reports from its users in case of problems or requests for changes. Such bugtrackers are key to the quality assurance process of the distributions. Unlike forums or mailing-lists, they often are the only place where thousands of users and maintainers can coordinate on the technical problems in a semi-structured way (thanks to the workflows of bug reports imposed by the bugtrackers). Apart from these popular bugtrackers for the distributions (running bugzilla, debbugs or launchpad), in turn, each independent FLOSS project generally maintains its own dedicated bugtracker which is mainly used by its developers and a few “power users”. They may then be running their own instance of Bugzilla, Mantis, Trac, Jira, Roundup or other such tools, or also using co-hosted trackers on a shared service like a software forge at SourceForge.net or LaunchPad, for instance.

The many bugs reported “downstream”, by the end users of a distribution into its central bugtracker, are most of the time related to the few reports filed and actually monitored by the original developers “upstream", which are carefully handled in their own project's bugtracker. For a single problem that is mentioned in a single bug report being taken care of, as assigned to an upstream developer, there may be tens of duplicate reports filed by the users of many GNU/Linux distributions (often without upstream developers having any clue of these many reports).

It is the role of the Distribution package maintainers (using Debian terminology) to “triage” bug reports, and to maintain the correspondence between “their” bug reports (in the distribution) and the corresponding ones in the bugtrackers of the “upstream” projects, that will ultimately be assigned to the developers in search for a fix. And only once a fix is made available upstream, the package maintainers may eventually decide to add the corresponding patches to their Distribution package, so that the fix can be delivered to users.

Traceability of these bug reports, their duplicates (in various bugtracker instances), their states (in different tools' syntaxes), and eventual fixes (and patches) proposed for them (to be applied or not, depending on one's Quality Assurance workflow) is essential in the ecosystem of FLOSS development.

In the Debian bugtracker debbugs, such links between Debian bug reports and corresponding upstream bugs is tracked by manually setting a “forwarded-to” attribute on Debian bug reports. This free-form annotation (which is publicly available) used to be set to an email address of the upstream developer or community to which the problem was in turn reported, and is now usually set to the URL of a bug report in the upstream bugtracker.

Figure 1 shows the trend of such “forwarded” bugs from Debian to the upstream project, observed in debbugs. After 2003, most of the forwarded bugs in Debian have been linked to bug reports filed in bugtrackers, which corresponds to the tendency of all FLOSS projects to start using bugtrackers to facilitate collaboration.

![Figure 1: Trend of bugs from the Debian distribution forwarded “upstream” (source: UDD database)](image)

Companion to this annotation, the bts­-link tool [2] addresses the need for package maintainers in the Debian distribution to monitor status changes of the various “upstream” bugs that have been set as targets of forwarded-to links. Being alerted by bts­-link (by email) of bugs being “closed” by the developers in an “upstream” bugtracker, the Debian maintainers can then identify when it's the right time to prepare an updated package to be provided to Debian users in need of the corresponding fix.

The bts­-link tool, running on a Debian distribution's server, periodically navigates such forwarded-to links, analyzes the target bugs status by querying the various bugtrackers' interfaces, and eventually notifies the maintainers (and interested subscribers) whenever the “upstream” reports in these bugtrackers change state (from “open” to “closed”, or “closed” to “re-open”, etc.).

The current way these bugs are interlinked with this “forwarded-to” annotation is somehow specific to the debbugs tool³, and error-prone, as wrong URL may be set by package maintainers. Also, to make it worse for the programmers of bts­-link, the interfaces and Web services to access various bugtrackers contents are not standardized, and sometimes some “screenscraping” is necessary on the Web pages.

³ Other tools like launchpad have another format for a similar feature, and ongoing developments in Bugzilla will eventually deliver yet another implementation.
1.2 Taming bugs without standard interfaces: the Mylyn case

Similar challenges are faced by implementers of other tools that interact with many different bugtracker tools, like the Mylyn\(^4\) task manager for the Eclipse development environment.

Mylyn offers a common interface inside Eclipse, to manipulate lists of tasks, coordinate work on these tasks (which are eventually bug reports or feature requests), reschedule them, collaborate on work related to these, and no matter where such tasks may be registered: local individual storage on the developer's computer, or on shared remote servers, among which many of the bugtrackers used by collaborative development projects are supported. Developers “triaging” bugs or working on their resolution are then able to manipulate them, without changing tools, i.e. without opening a Web browser and connecting to the bugtracker Web interfaces.

This is a great feature for programmers using Mylyn, which allows them to keep focused on code maintenance without juggling with different tools, keeping working in Eclipse. But this implies a huge effort for Mylyn implementers, as Mylyn currently includes tens of different client connectors for the various bugtracker APIs.

There would be a lot of benefit of all bugtrackers adopting a single standard for communication with different client tools like Mylyn of bts-link, at least in terms of bug reports retrieval (if not manipulation of the bug attributes).

As will be presented later, Mylyn is about to implement the OSLC-CM protocol that should solve this interoperability challenge and provide some means of Semantic interoperability. In HELIOS, we are investigating the possibility of improving the bts-link tool to make it less Debian-specific and to use the potential of Semantic Web techniques brought by OSLC-CM. We particularly hope to demonstrate benefits of LinkedData's best practices [3] to track links between such bugs in various bugtrackers, and the use of standard bug representations for interoperability. We hope that this will in the end prove beneficial for the Quality Assurance work in the whole open source ecosystem, as tools like Mylyn and bts-link could become more widely used in FLOSS development.

1.3 Estimating order in the bazaar: the Ultimate Debian Database (UDD)

Bugtrackers and their companion tools are not the only applications that are key to Quality Assurance practice in FLOSS Distributions.

In order to address some needs for more understanding of the process operating in such a loosely coordinated project as Debian, mainly for Quality Assurance (QA) concerns, some Debian Developers\(^5\) have developed a repository called UDD, the “Ultimate Debian Database”, for use inside the Debian community. This database, accessible to Debian contributors, groups facts about the Debian project, to ease the creation of (SQL) queries on what’s happening in the Distribution. This is for instance very helpful for QA tasks, like counting bugs with certain characteristics, comparing packages in various ways, or spotting “Missing In Action” (MIA) developers. Such queries are helpful in order to provide some kind of business intelligence reporting to back decisions in a volunteer-only community, where thousands of developers are generally bound by only a minimal set of technical requirements.

Even if started as a grassroots effort of Debian developers, and as a repository of facts about the Debian process, UDD is quite similar to the Flossmetrics\(^6\) or Flossmole databases. These repositories of repositories [12] are well known to the academic community; they are collecting facts about many libre software projects, by extracting contents of the project data from the hosting forges (including bug-related figures).

We imagine UDD could be helpful to researchers through its integration with Flossmole and similar archives, as it contains facts about the packaging phases (downstream) of the libre software development process for the many software developed (upstream) in the forges (that have already been crawled for RoRs). We imagine that analyzing links between such upstream and downstream activities of FLOSS actors can lead the way to new research.

However, a general criticism that we can make on these databases (just like for bugtrackers) is that their schema (the tables & columns layout, as well as the eventual relations) and the code of the data “harvesters” used to populate them, are the only means to understand the real semantics of the data collected there. There’s not much explicit semantics (unlike in RDF documents for instance), which diminishes the possibility to cross-link facts between different databases, or replay some analysis\(^7\). Surprisingly, sometimes the contents are even ambiguous between tables of the same UDD database, for known reasons, because, as explained by the UDD developers, there’s actually much incoherence in some of the Debian tools already (although it still happens to deliver a distribution anyway!).

More generally, as proposed in in By Howison [1], using Semantic Web techniques would allow access to contents of databases of facts such as UDD using standard ontologies. That would help and convey some bits of commonly agreed semantics, hence fostering interoperability between these databases. The development of the data acquisition tools used by academic researchers would be facilitated and much more reuse could happen with the use of standards such as RDF and ontologies.

For “simple” tasks like retrieving bug status, and observing bug fixing regimes in large scale analysis, the advent of standard bugtracker interfaces and standard Semantic models of bug properties will be key to sustainable study programs. Such semantic interoperability would ultimately benefit to all parties, be they Debian developers, Mylyn users, researchers monitoring bug fixing regimes, upstream developers, QA responsible persons, etc.

Let’s examine how the gap between tools and techniques of FLOSS development communities and those of the Research community may diminish in the future, with help of a few current initiatives.

\(^4\) http://www.eclipse.org/mylyn/
\(^5\) Lucas Nussbaum, Stefano Zacchioli and Marc Brockschmidt supervising the development made by Christian von Essen.

\(^6\) http://melquiades.flossmetrics.org/
\(^7\) This critique, which is common for all databases made available on the Web though web applications, is the main driver leading the advent of the Semantic Web, in other fields like egovernment.
2. Fostering interoperability through more Semantic tools

The advent of the Web 3.0, i.e. a Semantic Web of Linked Data [3] brings new perspectives to the field of software engineering in general, and to the tools used within HELIOS for the traceability of bug reports in particular.

The LinkedData initiative is described as follows on the reference site: "LinkedData is about using the Web to connect related data that wasn't previously linked, or using the Web to lower the barriers to linking data currently linked using other methods. More specifically, Wikipedia defines Linked Data as "a term used to describe a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF"."

Like in many other fields of application of the Semantic Web (like e-Government for instance), the study and the management of open-source software engineering is likely to benefit tremendously from the interlinking of distributed structured data that the Linked Data paradigm is bringing. Its main realization could be the shift from closed silos of facts collected about open source projects (as in current databases and repositories of repositories like UDD, Flossmetrics or even any bugtracker's database, as explained above) to future semantically-described Web resources interlinked within an overlay graph of machine-processable data on top of existing open source software infrastructure (forges, bugtrackers, wikis, mailing-lists etc.).

2.1 The LD2SD research project: Linked Data Driven Software Development

Linked Data Driven Software Development (LD2SD) is a lightweight Semantic Web methodology proposed by DERI to turn software artifacts such as data from version control systems, bug tracking tools or Java source code into linked data [11].

The use cases for which it was developed initially can be separated into two main categories:

- search persons for a task or issue who have the required skills to do it or solve it;
- search for software artifacts according to user's criteria;

The methodology consists mainly of the following steps: first, one must assign URIs to all entities in software artifacts and convert them to RDF representations based on the linked data principles and thus producing LD2SD data sets. In the next step, the RDF datasets are processed in order to find RDF fragments that are describing the same entity. On a positive match they are linked together using the property owl:sameAs, indicating that these URIs actually refer to the same entity.

Next, by use of semantic processors, such as the DERI Pipes[8], one is able to integrate, align and filter the LD2SD data sets. The DERI Pipes are an open source project used to build RDF-based mashups. They allow to fetch RDF documents from different sources (referenced via URIs), merge them and operate on them. In LD2SD methodology they are used to:

- Fetch the RDF representation of the repository log, bugtracker, source code, etc.;
- Merge the datasets;
- Query the resulting, integrated dataset with SPARQL;

The last step is to deliver the information to end-users, in a way that is integrated in their preferred environment (through AJAX mashup techniques, for instance over/inside an unmodified Jira bugtracker).

The novelty brought by the LD2SD methodology is that it allows to exhibit the previously implicit links between software artifacts found in software development (like version control systems, bug trackers, forums) by using the RDF representation.

Due to the extensible nature of resource description ontologies used in such RDF applications, and their associated reasoning, the benefit is much more versatile. Large scale observation of process happening in many different projects can be conducted. Contrary to database like UDD that are only conveying very low semantics as hard-coded by their makers, here, there's an open field for much wider and richer studies, using non-ambiguous standard data models.

This approach conducted at the DERI is quite similar to a recent development on doc4.mandriva.org, this time developed in a FLOSS community.

2.2 A semantic distribution portal: doc4.mandriva.org

doc4.mandriva.org is a recently rolled-out application[9] developed in Helios as a follow-up to the Nepomuk[10] project. It stems from the fact that, as stated by Henry Story[11]:

"Open source software is creating a global software space, with dependencies between projects, is meshing software from many different sources. But we are not meshing the data about the software!"

Doc4 aims at storing semantic statements pertaining to software engineering process happening inside the Mandriva community. For instance, it enhances the Mandriva bug descriptions by providing both automatically extracted data from a set of bug repositories (Mandriva, KDE, Gnome and others) and manual annotations enhancing the bug descriptions by knowledge that could not be inferred programmatically. The semantic database can then be queried either from a dedicated KDE tool or from a dedicated Web interface. A KDE annotation tool features the capability of storing the annotations either locally (in case they relate to private tasks or in case of offline work) or publicly on the doc4 server.

Using the RDF standard and chosen ontologies, the semantic information is stored in a specialized data store, which maintains the relationship between the data. The structure of the data stored is described by an ontology based mostly on the data model[12] proposed by EvoOnt [4]. A similar approach is followed in [6], by applying in addition Natural Language Processing technologies for extracting further information from the text data. In the near future, doc4 will be enhanced as well by text mining components, in particular in the context of the SCRIBO project [7].

http://pipes.deri.org/

http://doc4.mandriva.org/

http://nepomuk.semanticdesktop.org

http://www.ifi.uzh.ch/ddis/evo/
The work in Helios focus mostly on two parts: containers" expressed by Mark Shuttleworth and services that will allow to glue together various bug trackers OSLC-CM compatible bugtrackers adopted for bug description (currently drafted for OSLC-CM V2).

studies in a much easier way, due to the common semantics individual bug reports description, all bugtrackers offering public. Moreover, since an RDF description is specified in OSLC-CM for by FLOSS projects implement this API standard, it will eventually adoption by bugtracker implementers of bugtrackers, as the popularity of Mylyn should foster its need to implement custom connectors for tens of bugtrackers. This OSLC-CM V1 first consequence is the implementation of its specification for interacting with a bugtracker.

OSLC-CM V1 first consequence is the implementation of its standard protocol in new versions of Mylyn, which will reduce the need to implement custom connectors for tens of bugtrackers. This makes it the first credible candidate standard for interoperability of bugtrackers, as the popularity of Mylyn should foster its adoption by bugtracker implementers.13 When all bugtrackers used by FLOSS projects implement this API standard, it will eventually help fulfill the needs of projects such as bts-link for large scale bug traceability using standard APIs.

Moreover, since an RDF description is specified in OSLC-CM for individual bug reports description, all bugtrackers offering public OSLC-CM interfaces will become defacto contributors to the Semantic Web and potential Linked Data participants. This will open all these (public) bugtrackers for new QA work and research studies in a much easier way, due to the common semantics adopted for bug description (currently drafted for OSLC-CM V2).

2.4 Fetchbugs4.me: tracking bugs across information systems

The R&D efforts conducted in the frame of HELIOS on bug traceability, together with the progressive roll-out of RDF / OSLC-CM compatible bugtrackers allow the creation of models and services that will allow to glue together various bug trackers by information pipelines, realizing the vision of "porous federated containers" expressed by Mark Shuttleworth14 [9].

The work in Helios focus mostly on two parts:

- Improving bts-link like tools, so that the notification of linked bugs status changes can be used by more teams: not only only limited to Debian's debbugs on one side and some other few bugtrackers on the other side;
- Creating a Web application and service, called Fetchbugs4.me, which can act as a proxy, aggregating all bugs facts from various sources, in a similar way as the RSS aggregators, and then providing this information both on site or desktop remotely. Starting from the model proposed by the doc4 project, a more advanced tool will be created, which will support use of data standardized in RDF formats such as EvoOnt's BOM, OSLC-V2 and other ontologies.

At the beginning of the Helios project we developed the Helios_bt ontology to complement the BOM (Bug Ontology Model) and similar ontologies like Baetle15 (or the very basic one proposed in V1 of OSLC-CM), in order to better describe a generic bug model, as well as adapt to technical requirements of practical tools for open-source developers. We designed the ontology so that it includes bugtracker-specific classes for some attributes values of the bugs which define the life-cycle of a bug in a particular bugtracker. The goal was to render Helios_bt "neutral" on the semantic analysis of the equivalences that may exist or not for different bugtrackers having different bug life-cycle models. For instance, what are the equivalences between severities in two different bugtrackers? They are probably dependent on the context of the projects hosted in these bugtrackers and not only of the bugtracker developer's choices. Sometimes, these values can be configured also by the bugtracker admins, rendering such equivalences impossible to define once and for all. We believe that such correspondence is not absolute, but domain specific, in the context of a particular application (Quality, Software Engineering, etc.). So we decided not to offer such reasoning possibilities in a generic ontology as ours, and leave it to other ontologies to propose such equivalences. Still, we tried and define which states are terminal or not, as an exception to the previous rule.

The novelties brought by Helios_bt are the relational models between an issue/bug and software (packages, distributions, versions), persons and other issues. For example, it extends BOM by adding semantic relations between bugs (relations to describe duplicates, merges and dependencies) and by adding bugtracker-specific severities, priorities and states (for Bugzilla and debbugs). In EvoOnt's VOM and BOM ontologies, bugs are linked to versions of source files that have been committed in the source repositories in order to fix them. Instead, our approach is to create links between issues and the released versions of software packages that are reported as including a fix for a bug (in the changelog) and thus because a component of a product is made of software packages, which are packaged in distribution packages (source or binary).

Some widely used ontologies are still used, to describe facts related to entities linked to the bugs in the Helios_BT ontology (or its underlying BOM base):
- FOAF (Friend Of A Friend): describe relations between people
- DOAP (Description Of A Project): describes software development projects and adds the project's bugtracker relation
- SIOC (Semantically-Interlinked Online Communities): describes persons, accounts on online services and discussions

13 In Helios we'll work on implementing a reference OSLC-CM demo server as well as Mantis OSLC-CM interface connectors.
14 the creator of the Ubuntu Linux project
15 http://code.google.com/p/baetle/
The mixing of our novel ontology and these existing accepted standards, together with the roll-out of more OSLC-CM compatible tools will facilitate the construction of rich and standardized interfaces for semantics-aware bug tracking tools.

On an architectural level, once a generic data model is usable, the data conforming to it can be managed in two ways: either gather all the collected data into one giant (public or semi-public) RDF database (graph) providing a SPARQL endpoint for answering queries, or keep the data in the leaves of the network, connecting distributed semantic databases using the LinkedData model, hence embracing fully the Semantic Web vision.

The main idea behind Fetchbugs4.me is that users can access a service (i.e. a Web application) which allows them to subscribe to information about bugs filed in many bugtrackers. Then, it will act as a (read) proxy aggregating all bugs from various sources, in a similar way as the RSS aggregators (Google reader, etc.) collecting news feeds from various Web sites for users.

An immediate use of this is to monitor the list of all bugs one has reported, in many bugtrackers, in a single list, with status and other meaningful attributes.

Facts about these bugs, and changes to these, will be collected automatically (cached and refreshed if needed) by FB4Me, so that they can be presented to the user when he/she connects to the Service according to a set of personal preferences.

All elements collected by a user may then be published in the form of a semantically enhanced RSS feed (i.e. RSS 1.0 or ATOM with RDF payloads containing bug ontologies and other Semantic Web facts), to be consumed by other applications, tools, including desktop tools like Mylyn or KDE (for offline display for instance). Through such feeds, the FB4.me may provide other sources of bug facts, available for other applications (i.e. acting as a Web Service).

The user will also be able to add annotations to the bugs to add more knowledge, and such user-contributed added information can be shared by users interested in the same bugs. Such information may be semantic, relating to bug properties (duplicates, etc.), or custom tags.

We can even imagine very advanced features: by taking advantage of these collaborative annotation of the bugs, it may be possible to extract new facts about bugs, to be presented to other users interested in the same bugs, and have them confirm/note/vote in order to validate such recommendations. Above a certain threshold, or based on specific additional preferences, these annotations could in turn be notified to the original bugtrackers to which the bug was reported to.

Fetchbugs4.me will contain three major components: 1) a bug facts collectors running for every known bugtrackers (that may collect only those bugs for which users are interested). They would use export formats if available (mainly OSLC-CM if it succeeds as a standard) or do some Web scraping if necessary; 2) an internal engine for data processing, that would store the facts about the bugs, and manage the caching/expiration algorithm; 3) interfaces including Web pages for humans and RDF/RSS 1.0 feed provider to be consumed by other Web apps, or desktop clients, RSS readers, etc.

The advantages of using such a Web service are the possibility of tracking multiple related bugs across several bugtrackers through one single RSS feed. It reduces the effort/time to check for changes on “favorite” bugs, offers a centralized history system (may be added later as a feature) and most important by using tags, users can share bug information when configuring their tags.

But there are also disadvantages, because it may create uneven load on bugtrackers (a high number of update requests at the same time) and it implies dedicated team for the Web service maintenance.

Nevertheless, we hope we can address the technical and computational challenges of such ambitious goals, and help foster interoperability between bugtracking services and client tools. We hope this can improve traceability of bugs and effort sharing between projects and distributions, hence improving the general quality of open source software.

2.5 Increased usability and data interchange

The use of Semantic Web techniques will become more widespread in the GNU/Linux systems in the coming years for the availability of novel desktop tools. Thanks to the outcome of the Nepomuk project, the recent Linux desktop include metadata management capabilities across applications, both in KDE (>= 4.3) and Gnome. Modern Linux desktops like Mandriva One 2010 already rely on standard components like Soprano or Tracker which act as building blocks of new end-user applications harnessing the potential of the Semantic Web approach, both at the personal (PIM applications, files and bookmarks, tasks and logs of activity) and at the Web levels. Such applications may include for instance a BOM/OSLC-CM aware bug triage client tool, a bug annotation tool, a distributed semantic search engine across software issues in remote forges, etc.

The progressive adoption of semantic enabled databases on the GNU/Linux distributions servers [10] like doc4, and within archives at research facilities, together with LD2SD-like methodologies for Software Engineering will increase the interoperability of all these services with the tools used by the end-users, developers or researchers in their preferred environments.

One of the challenges is the way standards can emerge from the various designs by researchers, developers and users, of ontologies modeling the realm of open source software development. We hope that OSLC-CM can help play such a role through massive adoption by Mylyn users.

Another challenge relies on the capacity of handling efficiently very large volumes of data, or other computational issues. Relational databases prove that the currently available data can be processed and can lead to useful knowledge and metrics assisting effectively the engineers in their production tasks, whereas the scalability of the Semantic Web approach in the field of software engineering remains to be experimented.

Conclusion

Our efforts conducted in the frame of the HELIOS project, together with other above-mentioned initiatives like OSLC or LD2SD, aim at widening the use of developer-friendly applications harnessing the potential of Semantic Web aware services. Such applications may include an improved bts-link, and other tools linking information across dedicated information systems like doc4 or Fetchbugs4.me, contributing to generic bridges federating OSS information systems into a giant knowledge base distributed over the (Semantic) Web.

The use of Semantic Web techniques is likely to improve the processes at stake in open-source software engineering and maintenance, both at the individual level through a deep integration of production and monitoring tools (like Mylyn) with the contributor’s desktop and own mental vision of the processes,
and at the collective level through better connections between distributed engineering facts in the open-source ecosystem, which are changing at rapid pace on the Web. Not only will the Semantic Web techniques adoption ease the distributed engineering processes, but also the reuse of common data by both researchers and software developers or users, hence changing the way research on open-source software is conducted.

References


