The Networked Forge: New Environments for Libre Software Development[1]

Jesus M. Gonzalez-Barahona[1], Andrés Martínez[2], Alvaro Polo[2], Juan José Hierro[2], Marcos Reyes[2], Javier Soriano[3], and Rafael Fernández[3]
[1] GSyC/LibreSoft, Universidad Rey Juan Carlos
jgb@gsyc.escet.urjc.es
[2] Telefónica Investigación y Desarrollo
[3] Universidad Politécnica de Madrid

Abstract. Libre (free, open source) software forges (sites hosting the development infrastructure for a collection of projects) have been stable in architecture, services and concept since they become popular during the late 1990s. During this time several problems that cannot be solved without dramatic design changes have become evident. To overcome them, we propose a new concept, the "networked forge", focused on addressing the core characteristics of libre software development and the needs of developers. The key of this proposal is to re-engineer forges as a net of distributed components which can be composed and configured according to the needs of users, using a combination of web 2.0, semantic web and mashup technologies. This approach is flexible enough to accommodate different development processes, while at the same time interoperates with current facilities.

1 Introduction

The libre (free, open source) software[2] development community, considered as a whole, is one of the largest cases of informal, globally distributed, virtual organization oriented to the production of goods. Hundreds of thousands of developers (working for companies or as volunteers) share a large base of source code (hundreds of millions of lines of code) and knowledge which they use to produce and improve software products. This collaboration has been possible only thanks to the intensive use of Internet-based tools, currently offered mainly by development forges such as SourceForge.

1 2This work has been funded in part by the European Commission, through projects FLOSSMetrics, FP6-IST-5-033982, and Qualipso, FP6-IST-034763, and by the Spanish Ministry of Industry, through projects Morfeo, FIT-350400-2006-20, and Vulcano, FIT-340503-2006-3

2 In this paper the term "libre software" is used to refer both to "free software" (as defined by the Free Software Foundation) and "open source software" (as defined by the Open Source Initiative).

Please use the following format when citing this chapter:

Gonzalez-Barahona, J.M., Martínez, A., Polo, A., Hierro, J.J., Reyes, M., Soriano, J. and Fernández, R., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 299–306.

In fact, the dawn of libre software development communities is linked to the spread of the Internet. Since their startup, many development teams used Internet-based tools for collaboration, and dissemination of the produced software. With the spread of the web they were integrated in 'project sites' which provided the infrastructure used for collaboration [7]. Around 1995 those sites provided mailing lists, download areas, issue tracking, source code management (usually CVS), and static HTML pages with information and documentation about the project.

During late 1990s and early 2000s some organizations started to offer those facilities to large collections of projects. The most known, and by far the largest of them, is SourceForge, established in 1999, but many others do exist. Many of them run different forks of the original SourceForge software, of which the most popular (specially for small sites) is GForge. In addition, large projects (GNOME, KDE, Apache, Eclipse, OpenOffice.org, Mozilla, etc.) maintain their own forges, usually with ad-hoc software, and similar systems are used as well in corporate environments for the development of non-libre software [3].

The basic architecture and services of all these forges have evolved only slightly during the last decade, meeting many of the needs of development teams, and proving to scale well to the tens of thousands of projects, hundreds of thousands of developers, and hundred of millions of lines of code. However, they also show several problems, rooted in their incomplete adaption to the extremely distributed, interrelated and flexible nature of the libre software community. To fix them, we decided to rethink the concept of forge.

2 Current forges and their problems

The architecture of current forges is similar [8, 1], with the needed considerations for scale, which may lead to the use of computer farms in large cases. Their main components are a web server (as front-end), a web application (e.g., GForge, providing the specific services of the forge), an SQL database (persistent storage for non-massive elements) and some specific components (such as source control or mailing lists management).

The web application provides services such as: information about projects and developers (including authentication), issue tracking, news and forums, wikis, scheduling services, and a common web interface to most of the functionality (including downloads of code). In addition, specific components usually found accompanying this application are: a download manager, a source control management system (CVS, Subversion, etc.), and a mailing lists management system with archiving facilities.

About one decade of intense use of these forges have uncovered several problems:

- They are project-centric instead of developer- or user-centric. The service "unit" is the project, but both libre software developers and users are usually interested in many projects, hosted in several forges.
- Monolithic approach. Each forge offers a fixed set of services, each implemented by a specific system, which impedes that each project chose the software they prefer for every service, and slows down innovation (only administrators can add new modules).
- Isolation. For most practical purposes, each forge is isolated from others (even with federation facilities).
- Poor integration. Each separate component offer its own user interface.
- Lack of fine-grained coordination. Related elements in different subsystems are difficult to relate to each other (e.g., patch to fix a bug and its revision in source code management repository).
- Lack of support for views. A given project, or collection of projects, cannot offer multiple views to users.
- Little attention to collaborative knowledge sharing. Collaborative tagging, bookmarking and cataloguing, for instance, are missing.

There are also some concerns related to the extreme concentration in the most popular forges. Some of them are becoming single points of failure and potential control for libre software development as a whole [4, 5]. Therefore, they become a critical infrastructure that has to be maintained and defended something which is intensive in human resources, and difficult to scale.

A number of solutions to some of these problems have been proposed, either as improvements to existing forges, or as new systems (such as Launchpad), but they still remain open issues needing a comprehensive approach. In some reviews of future developments in the field [9], several scenarios that would address some of these problems are also described, but they do not propose detailed designs or implementations.

3 The networked forge

Given this situation, we found it reasonable to re-engineer the fundamentals of forges, under the following main lines: aggressive distribution (services located in different sites); easy relocation (backup and restores interfaces that allow for quick recovering of a service at another location); user-centric scenarios (which allow developers and users to access easily all the services they need, despite the project to which they are related); fine-grained links between different services (so that developers can access related information easily); federation and presence in different forges (so that a project can be supported by several sites); relationship with upstream and related projects (even when residing in different forges); composability of independent elements; and fostering the development and sharing of innovative

component. As an important side-target, we also wanted to maintain a high level of interoperability with legacy services, to easy the transition.

Our proposal, the networked forge, is based on the idea of considering forges not as single sites providing a monolithic set of services to host projects as isolated silos of knowledge, but as a collection of distributed components providing services, among which knowledge is shared. Each project decides on its own customized set of services, and users can configure their own working environment. This idea matches that of mashups [6] or semantic web 2.0 applications [2].

The main components of the architecture are (see figure 1**Error! Reference source not found.**):

- Integrated services, specifically built to feed into the proposed framework.
- Legacy systems, that have to be integrated in the networked forge.
- Client components provide the user interfaces.
- Connectors and adapters, connecting legacy components to the rest of the system (connectors are used with those components providing a semantic, RDF-like interface, adapters with non-semantic components).
- Aggregators, collecting RDF channels and processing them in several ways.
- Locators, used as name services, allowing for the registration of specific components.
- Catalogues, in which components available for a certain community are registered.

Several connectors or adapters can work with the same legacy service, providing different interfaces to it. Conversely, a given connector or adapter can work with several instances of the same kind of legacy service, providing the same interface to several sites. Client components can interact directly with the integrated services, with some semantic legacy services, and with connectors, adapters and aggregators. Aggregators can interact as well with all these components.

The architecture imposes little limitations to where the different components may reside. Usually, legacy services will be hosted in different sites in the Internet. Integrated services will also run somewhere in the Internet, but they could even be located in some cases in the user workstation. Client components will usually be hosted in some web server in the Internet, but will run in the user browser. However, other combinations are possible: they could also run in servers and be visualized in web browsers, or reside in the client side, for instance as modules for an IDE.

The communication between all the non-legacy components is performed with HTTP and WebDAV, with all the components providing RDF channels via REST interfaces. The architecture uses semantic web technologies for the exchange, handling and querying of the data between the different components of a forge network.

All connecting components feature a common REST interface, which simplifies composition. The information provided by the component is always an RDF channel, including semantic labeling, which is obtained via HTTP. Aggregators act as filters, also accepting RDF channels as information.

Components provide RDF channels, but no other interface intended for end users (such as HTML pages). Therefore, the user interface is usually provided in the client side. When the client application is a web browser, AJAX techniques are used to display the data in useful ways. In fact, the most natural way of producing an application interface for a networked forge will be via a mashup (including those implemented using Google Gadgets or Yahoo Pipes). But it is important to notice that any kind of application could interface to the system from the desktop, including for instance RDF client-side aggregators, or components for an IDE such as Eclipse plugins.

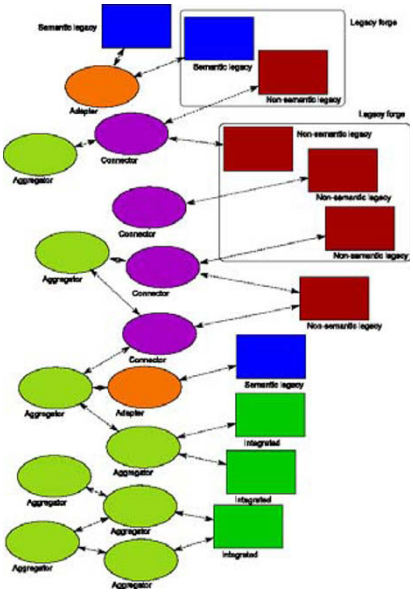

Fig. 1: General architecture. Clients, locators and catalogues are not shown to improve visibility.

Locators and catalogues are optional but important components. Both accept, at configuration or run time, information about the location of the different components (that is, their url), and provide upon request that information (either for all the components registered, or for a certain subset of them fulfilling some property).

Catalogues maintain information about available components to compose forges, while locators are configured to 'build' a networked forge: the components of the forge will be those registered with the locator. Therefore, components will be a part of several networked forges just by being registered with different locators.

To be functional, the services offered by the components of the networked forge are integrated in flexible and diverse ways. This integration can be accomplished either in the server side (using aggregators) or in the client side (using mashups, standalone RDF aggregators, or plugins in an IDE).

4 Implementing the concept

We have implemented a first version of the networked forge. A proof-of-concept setup using it is composed of several connectors and adapters to integrate legacy components (GForge, Trac, Subversion, etc.), several widgets (using Google Gadgets technology) to implement a client-side application to access the networked forge, and several locators. Figure 2**Error! Reference source not found.** shows a screenshot of this implementation.

Each widget reacts to changes in other widgets. For instance, the "My Vulcano" gadget (which shows a list of projects) controls the content of the "Project Details", "Project Tickets", "Project's Wiki" and "SVN Log" widgets. Widgets can also be configured by the user, pointing them to different urls: "My Vulcano" widget shows a different list of projects if pointed to a different locator.

The current implementation is written in PHP, Python and Java (standalone adapters or connectors on server side), and in JavaScript (client side adapters and connectors and gadgets logic). It is still minimalistic, but even so it shows the great potential of the concept of a networked forge.

Other proof-of-concept scenarios mimic the functionality of a legacy multi-project forge; the forge for a company, including its collaborations in libre software projects; the forge for a software distribution, including the development of upstream (original) products and their packaging activities; and a personal forge.

5 Conclusions and further work

Forges are an increasingly important tool for collaborative software development. However, the traditional model for implementing them has some shortcomings that we address with a new design: the networked forge. Networked forges allow for the easy integration of components residing in many different sites and administrative realms, while at the same time provide a great deal of flexibility.

Given a networked forge infrastructure, users, developers, projects, companies and other actors can configure the forges they need, share their configurations, and select the exact services they prefer. Interoperability with legacy systems is a part of the model, as well as a seamless integration with many different client-side environments, from web-based mashups to traditional IDE applications. The proposed architecture has been tested in proof-of-concept implementations.

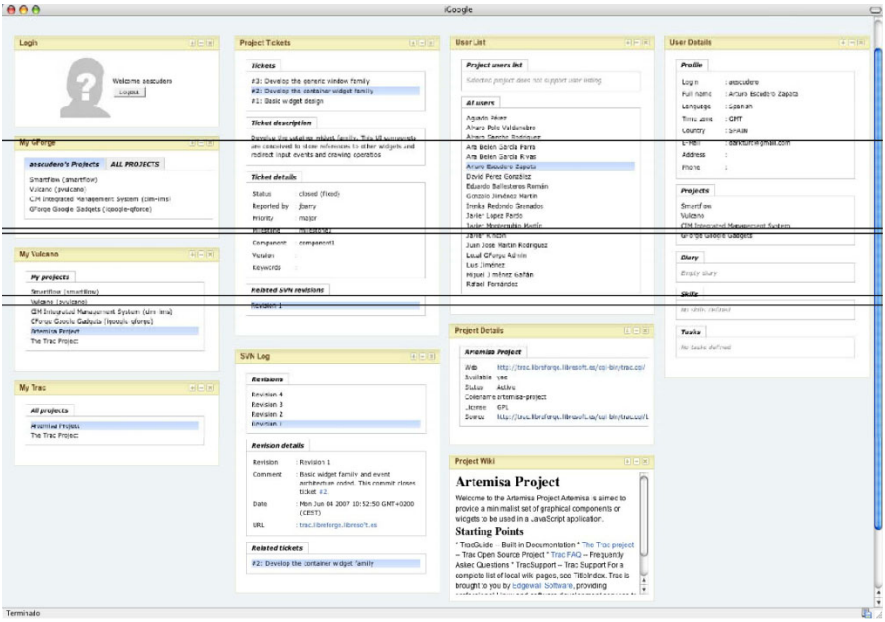

Fig. 2: Proof-of-concept networked forge. User interface implemented with Google Gadgets.

Some aspects of the design still remain open, while we explore several options for them. Security and authentication, for which we are studying single sign-on or common authentication infrastructure technologies, are two of them. Selection of ontologies for the exchange of information, and formats for specifying connections between components are also an open field. We are also developing more complete implementations in the context of the Vulcano and Qualipso projects.

In summary, the networked forge is a concept that suits well the practices of the libre software development community, and can also be applied in other more traditional environments.

Acknowledgements This work has benefited from many discussions in the context of the Vulcano, Morfeo, OSOR, FLOSSMetrics and Qualipso projects. We thank all the persons who contributed with ideas, comments and criticisms.

References

[1] Olivier Abdoun, Bernard Lange, Stéphane Laurière, Irenka Redondo, and Christian Rémy. Collaborative development environment: A state of the art. Technical report, Qualipso Project, 2007 (to be published).

[2] Anupriya Ankolekar, Markus Krötzsch, Thanh Tran, and Denny Vrandecic. The two cultures: mashing up web 2.0 and the semantic web. In *Proceedings of the 16th international conference on World Wide Web*, pages 825–834, New York, NY, USA, 2007. ACM Press.

[3] Grady Booch. Introducing collaborative development environments. Technical report, IBM, December 2006.

`http://www.alphaworks.ibm.com/contentnr/cdepaper`.

[4] Loïc Dachary. SourceForge drifting, 2001.

`http://fsfeurope.org/news/article2001-10-20-01.en.html`.

[5] Jesus M. Gonzalez-Barahona and Pedro de las Heras Quirós. *Future Trends in Distributed Computing. Andre Schiper, Alex Shvartsman, Hakim Weatherspoon, Ben Zhao eds.*, chapter Hosting of Libre Software Projects: A Distributed Peer-to-Peer Approach, pages 207–211. Number LNCS 2584 in Lecture Notes in Computer Science. Springer Verlag, 2003.

[6] Anant Jhingran. Enterprise information mashups: integrating information, simply. In *VLDB'2006: Proceedings of the 32nd international Conference on Very Large Data Bases*, pages 3–4. VLDB Endowment, 2006.

[7] Tim O'Reilly. Lessons from open-source software development. *Commun. ACM*, 42(4):32–37, 1999.

[8] Patrice-Emmanuel Schmitz, Abdelkrim Boujraf, Rishab Aiyer Ghosh, Emidio Stani, Juan José Amor Iglesias, Julia Anaya González, and Álvaro del Castillo San Félix. Feasibility study. Technical report, Open Source Observatory and Repository, IDABC, 2007 (to be published).

[9] Jim Whitehead. Collaboration in software engineering: A roadmap. In *Future of Software Engineering (FOSE '07)*, pages 214–225, 2007.