Release early and often

Developing software with Origo

Report

Author(s):
Bay, Till; Oriol, Manuel; Meyer, Bertrand

Publication date:
2012

Permanent link:
https://doi.org/10.3929/ethz-a-006820313

Originally published in:
Technical report / Computer Science Department, ETH Zürich 581

Release early and often: Developing Software with Origo

Till G. Bay
Chair of Software Engineering
Swiss Federal Institute of Technology in Zürich
CH-8092 Zürich
till.bay@inf.ethz.ch

Manuel Oriol
Chair of Software Engineering
Swiss Federal Institute of Technology in Zürich
CH-8092 Zürich
manuel.oriol@inf.ethz.ch

Bertrand Meyer
Chair of Software Engineering
Swiss Federal Institute of Technology in Zürich
CH-8092 Zürich
bertrand.meyer@inf.ethz.ch

ABSTRACT

Just as important as the technical activities of software development – requirements, design, coding, documenting, compiling, testing, debugging. . . – are the management and communication tasks: recording project events, managing project Wikis and web pages, sending out notifications, reconciling changes, and many others. These tasks become ever more delicate with the increasingly distributed nature of modern software projects, small as well as large. If not handled properly they can not only consume considerable time but also, just like bugs and other flaws in technical tasks, cause considerable damage.

Origo is a comprehensive platform for addressing such project needs by providing such facilities as project Web pages (both editable and generated), forums, mailing lists, bug tracking etc. All the facilities are also available through a program interface (API), allowing development tools and environments to invoke Origo mechanisms automatically upon completion of specified project events such as a compilation, a commit into the configuration management system, a failed test; environments for which a specific Origo plugin already exists include Visual Studio, Eclipse and EiffelStudio. Internally, Origo relies on a peer-to-peer middleware architecture supporting the integration of such application components as web and Wiki servers, database servers, business logic, configuration management, identification, access control, load balancing. The infrastructure is extendible both statically and at run time, allowing the integration of new functionalities by indepenedent providers.

Hosted by ETH Zurich, Origo is freely available to any project, open-source or closed-source, and designed for scalability. A few weeks after its introduction in August 2007, Origo is already hosting several hundred projects. The article describes the Origo services and architecture.

Categories and Subject Descriptors

K.6.3 [**Software Management**]: Software development, Software maintenance, Software process, Software selection

General Terms

Management, Human factors

Keywords

Software development process, software distribution and documentation, developer and user management, open- and closed-source development

1. INTRODUCTION

Modern distributed software development combines several applications and tools to allow developers to collaborate over a network. The combination of these applications forms the information systems used in software projects. Releasing software early and often requires that such an information system fits well into the development process. Often, development teams are the first adopters of emerging tools and technologies that support and accelerate collaborative work. As a consequence a software development information system has to be able to integrate new applications when they become important for a process.

The Origo platform provides a generic replacement for any ad-hoc combination of applications and improves the state of the art of existing development platforms. Released in August 2007, the platform already hosts more than 300 projects with over 1000 developers worldwide.

The novel services Origo offers are an application programming interface (API), an innovative display of events of project life and the possibility to host open- and closed-source project development.

Offering an API allows integration of Origo into any development process (see Section 2).

Origo relies on a network layer programmed using the JXTA [18] peer-to-peer protocol. This makes it possible to add new applications by extending them with JXTA libraries to handle the transport layer.

Section 2 presents the development platform, the IDE plugins and the API. Section 3 describes the architecture and implementation of Origo. Section 4 discusses related work and Section 5 draws the conclusions and provides an outlook on the future additions to the platform.

2. USING ORIGO

Origo (http://origo.ethz.ch) is an open-source information management platform for software projects. The platform enables a team of developers to track their own projects and those of other teams. For a developer, Origo encourages collaboration with several development teams and allows

working both on open and closed-source projects. Every Origo project has a web page that can be reached over a sub-domain (http://yourproject.origo.ethz.ch).

The platform does not impose any particular development model, technology or tool; the development proper happens outside Origo. The following paragraphs describe what Origo offers both for development teams and for users of the software projects that are hosted on Origo.

2.1 Projects and People

The software development platform manages projects, their development teams and user communities. Once registered as an Origo user, one can hold any of three roles for a given project: *project user*, *project member*, *project owner*.

Project *members* and *owners* are part of a project's development team. They both can modify the wiki pages of the project website, create releases on the download area and commit code into the Subversion repository. They can also post blogs, report issues and generally modify all content on the project pages. Adding new developers to a project team can be done by project *owners* only.

Project *users* can only report issues, write forum posts and comment content on project pages. The low number of different roles people can hold keeps the rights management for all actions concerning the project pages simple.

This simplicity also contributes to the usability of Origo: users and developers do not need to read documentation to start using the platform. While there is no built-in limit to project size, we have initially targeted Origo at fairly small projects; of the 300 projects hosted at the time of writing 80% have seven or fewer developers.

2.2 Basic features

The basic features of an Origo-hosted project are the usual services on which development teams rely today: a configuration management server for hosting the code; documentation and communication possibilities; ways to report and manage issues; a place to publish project releases. More advanced features are detailed in Section 2.3. A sample project page shows Figure 1. Every Origo project has the following features:

- Public and private wiki pages with WikiMedia syntax.
- Subversion repository with web user interface.
- Issue tracking with public and private issues.
- Blog, forums, comments, tags and screenshots.
- Release download area with mirroring.

The project web page, of which an example appears in Figure 1, has navigation links that lead to the home page, download section, screenshots, documentation, the forum page, the blog, the issue tracker and the development page. A project includes two kinds of pages: editable and generated. Editable pages which project members can freely create and update, use the wiki format; they include the home page, the screenshots, the documentation and the development pages. Generated pages are the download area, where all releases of a project are listed, the forum, the blog and the issue tracker.

Fundamental functionalities, repeated on every page (currently in the left-side menu, see Figure 1) include: the user and project settings, creation of new content, the request form for project creation and pages that allow tracking changes.

Also on every page are the search functions. There are several different kinds of searches; first, all projects hosted on Origo can be searched with Google Custom Search [14]. Every project page can also be searched separately. One can search for Origo users and the fourth search retrieves tagged issues. The searches are implemented using Generic Component Lookup (GCL) [2]. GCL is a search system that identifies different dimensions in the data to search separately, and weighs the results for each dimension. Sorting results according to an arithmetic combination of the weights fine-tunes and improves them. This way users can both search all the tags and the text of a reported issue separately but then receive the results combined.

2.3 Novel features

Together with the scalable, extendible and language independent design of the platform (see Section 3), Origo innovates through the following features:

- Besides the user interface already sketched, Origo provides a programming interface (API) enabling application developers to hook their processes and tools programmatically to the platform.
- Origo gives developers and users of a project a concise overview of the state of the projects (see Figure 3).
- Origo allows hosting both open and closed source projects.

Integration into the development process

The Origo API enables integration of the platform into any development process. It is implemented using XML-RPC [25]. XML-RPC is a simple, open-source specification and implementation for remote procedure calls between disparate and heterogeneous software systems. It uses HTTP/S as transport protocol and XML for message encoding. Messages are method calls with their argument data. Currently a wide range of languages provide XML-RPC libraries, including Eiffel, Java, C, Python, Perl, Objective-C, PHP. Since XML-RPC uses HTTP/S as transport protocol, it is easy to implement if no existing language binding can be used.

Every software project creates deliveries of its code. In some cases the delivery is an application that users can download, in other cases the delivery consists of a library that can be reused. All languages, operating systems and tools rely on the regularly recurring activity of building a delivery, driven by scripts. In spite of the wide diversity of tools and processes, the delivery step follows a common pattern. It involves a number of actions scheduled not manually but through a script like a *Makefile*, an *ANT-Script* or a *Visual Studio Solution File*. Once the script has been run, the next step is to publish the delivery online. Publishing a delivery can with the Origo API be integrated into the scripts. One of those scripts simply has to use the API call to publish a release for a project on its Origo page; that last step is also automated. There are many other API calls available for a project (see the complete list online[1]). This example illustrates how the Origo API enables hooking into a development process.

[1] http://origo.ethz.ch/wiki/origo_api

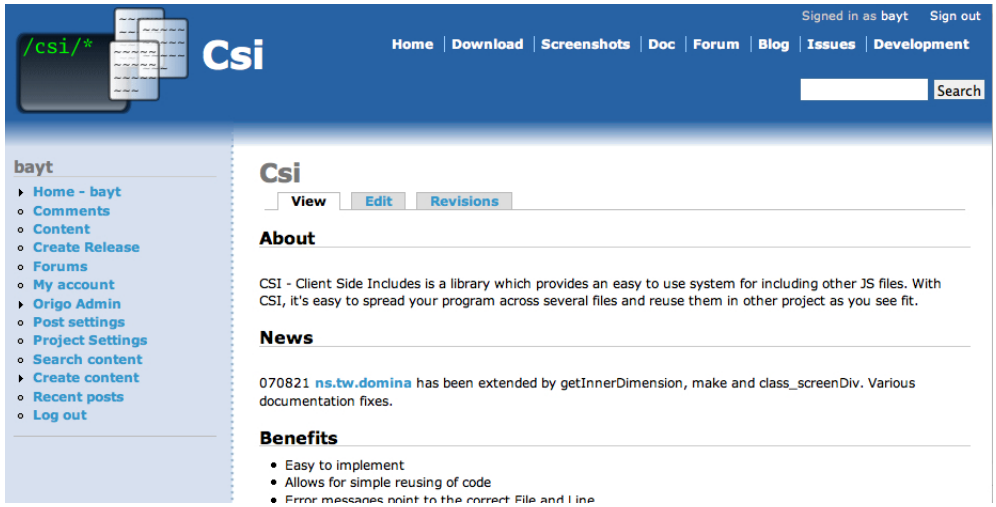

Figure 1: Typical Origo project page - http://csi.origo.ethz.ch

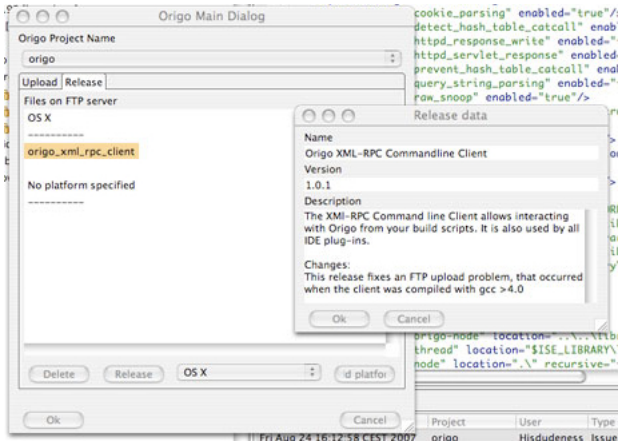

Figure 2: Release creation dialog in the Origo Eclipse plugin

IDE Integration

Similarly to build script integration, the platform can be used directly from integrated development environments. Currently, plugins for EiffelStudio, Eclipse and VisualStudio exist.

Figure 2 shows the Origo Eclipse plugin. Using the plugin for an IDE, users can choose files to upload to the project page, specifying release and platform information. All three IDE plugins offer the same functionality and the release creation dialog looks the same for all three IDE plugins. They all use the XML-RPC API to send calls from the IDE to Origo.

Besides integrating release creation, the IDE plugins also offer access to the work item overview that is discussed in the next section.

Work item overview

A member of a collaborative project needs to know what other team members are doing. The most important resources are the source code files; projects will typically rely on configuration management such as Subversion to enable concurrent access by multiple developers. Apart from managing the code with such a tool, developers typically also like to be informed of changes to the source code in other ways. An important functionality of such systems is the ability to send mail to project members on occurrence of specific events such as commits.

Origo generalizes this concept by enabling projects to hook up various actions to many possible events of project life. In Origo, all resources that can experience modifications are tracked and the changes summarized in the work item overview of a project page (see Figure 3).

The five resources tracked by Origo for each project are changes to source files, issues that are reported or modified, wiki pages that are edited, blogs that are posted and releases that are published. Figure 3 shows the work item overview page. In the figure, the tab for the Origo project itself is active and it shows the work items of the project at that point in time. Each listed work item links to a corresponding page that shows the changed resource. When a new resource is created (for example a new blog posted) - the link points to that resource. If an existing resource is modified, the link points to a page showing the differences between the old and the new version of the resource (for example when a typo on an existing wiki page is corrected). The figure shows also other tabs for other projects. The tabs on the left are all the projects of which the user (here: bayt) is a member. After the projects that a user belongs to, the tabs for bookmarked projects are enumerated. Every project on Origo can be bookmarked by a user and then the work items for that project are shown on the overview page accordingly. Whenever new work items for a project are published, its corresponding tab is highlighted.

For each project showing on the work item overview page one can configure both the mode of notification (receiving a mail for an update on a resource or listing the updated work item on the overview page) and it can be set which of the five work item types should be tracked. This allows setting the level of information a user or developer desires individually for every project. For all work items and projects an RSS feed is generated and can be used to get information on work

item updates as well.

As mentioned above, also the IDE plugins show the work item overview as part of their user interface. Just like the work item overview page, the IDE plugins link the displayed work items back to Origo, or show the updated differences directly inside the IDE. For screenshots of the IDE plugins see the screenshot page online[2].

Open- and closed source projects

The third contribution of Origo is the hosting of closed-source projects. Origo helps its users produce and maintain better software, faster, more effectively. It is not its role to push a political agenda for either open- or closed-source development. There are all kinds of reasons for a project to prefer a closed-source model. This is not only true in the software industry, but even in a purely academic environment where some work on projects needs to be kept closed until time for publication is suitable. The open or closed-source nature of an Origo project determines the visibility of its work items for its non-members; source code commits for example can only be seen and accessed by developers that are members of a closed-source project.

3. ARCHITECTURE

The key design goals for Origo were scalability, extendibility and language independence. To achieve these aims, Origo uses a peer-to-peer (P2P) back-end that relies on widely supported communication protocols. Figure 4 shows the architecture of the platform. Origo is running on multiple servers and is built following the model view controller (MVC) pattern. In the figure, the P2P back-end represents the controller and the applications represent the different views that are available. One of those views is the project page of a given Origo project; another would be the work items that are displayed in the Eclipse plugin for Origo[3]. The views and the controller communicate using XML-RPC. For communication inside the controller, Origo uses the P2P framework VamPeer [23] that is itself based on JXTA [18]. Some nodes of the back-end provide access to databases that are themselves representing the model in the MVC pattern.

3.1 Back-end

The back-end forming the controller of Origo is built using different peer-to-peer nodes. Besides acting as controller for the platform, the nodes of the back-end also provide access to several collaborating services that are used in Origo. Services include the database server, the Subversion servers, the FTP and the mail server; all of them exist as nodes of the back-end. The JXTA protocol is the infrastructure that they use to communicate. The back-end relies on four main notions, *nodes*, *node types*, *messages* and *use cases*, to be detailed now.

Nodes and Node Types

Each *node* represents a service used in the back-end (see Figure 5). A *node type* regroups a set of *nodes* that all provide the same service. Each *node type* has its own policies for *message* processing. Each *node* of a given *node type* is equal and any incoming *message* for a *node* can be treated by any other *node* of the same type.

[2]http://www.origo.ethz.ch/wiki/screenshots
[3]http://origo.ethz.ch/download

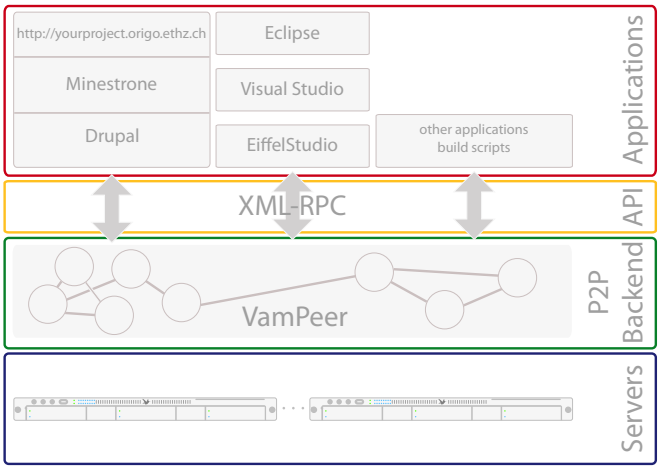

Figure 4: Origo architecture

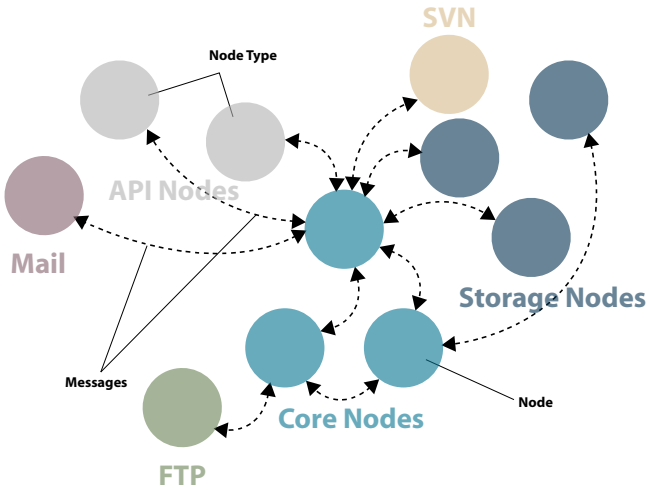

Figure 5: Nodes and node types in Origo

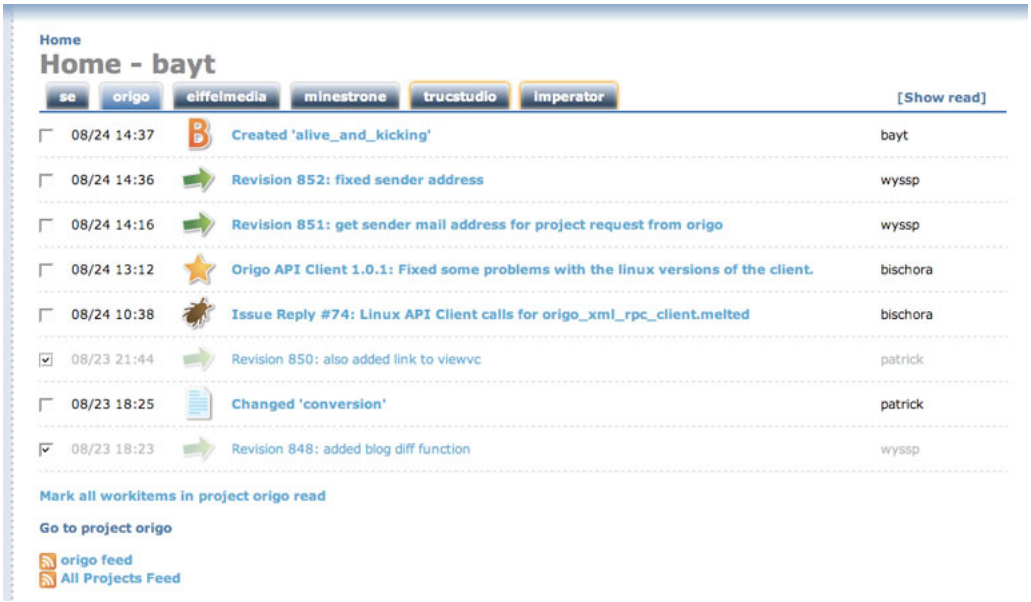

Figure 3: Work item overview page for multiple projects

Messages and Use Cases

The core nodes contain the code representing the *use cases*. A *use case* is the description of how the Origo back-end reacts to an incoming XML-RPC API call. After the API nodes receive a call, *messages* are sent to the core nodes that then send *messages* to all *nodes* taking part and performing some action in that particular *use case*.

Core Nodes

The core nodes are the controller of Origo and contain the code of the *use cases*. They manage all interactions between other nodes. Load balancing for all *node types* is implemented in the core nodes. The core nodes are aware of all *node types* existing and know how to use them to perform each of the *use cases*.

API Nodes

The API nodes are the interface to access the Origo back-end. API nodes listen to incoming XML-RPC calls as a daemon on a port on the server machines of the platform. The API nodes expose parts of the *use cases* that are encoded in the core nodes. They are not load balanced by the core nodes. For load balancing incoming XML-RPC calls, traditional web-server load balancing techniques like round robin IP address resolution, or LVS systems should be used.

A list of the API calls currently offered can be found online[4]. The calls offered allow access to all information stored in an Origo project. There are calls to enumerate the work items for a project, calls to upload and publish project releases, calls for issue management as well as general calls handling access and rights management, login processing and management of user information.

Internal API Nodes

To avoid possible denial of service attacks to API nodes, parts of the exposed API have to be hidden to outside applications. The API call for creating a new Origo user is an example for a call that has to be hidden, because bots would be able to create uncontrollable numbers of Origo users if it were available. Hidden calls are only available to certain views - in the case of the user-creation call, the view that has access to it is the Origo project web page (in Figure 4 the web page of an Origo project is the stack Drupal / Minestrone / http://yourproject.origo.ethz.ch in the applications layer).

Storage Nodes

The storage nodes maintain connections to database servers. All data used by Origo is stored in the databases managed by them. The storage nodes handle all communication and authentication with the databases. The databases store both data on users (user ID, name, password hash, email address and an application key) and on projects (name, description and a logo) as well as the association of users and projects, associations between users (development teams), access policies for all resources managed by the platform, role management information, and session management.

Custom Nodes

An important part of Origo are the services used for software development. The back-end contains three custom nodes that wrap these services: One node for configuration management (at the moment this is done with Subversion), one node for file upload when creating software releases (implemented with an FTP server) and finally a node that can send mail. Figure 5 shows the three nodes together with the other node of the back-end.

These custom nodes wrap an external server application as a node for the back-end. These server applications all have their own user management mechanisms and access protocols. The custom nodes of the back-end configure these server applications to integrate them fully into Origo. The login credentials for these services are created by the custom nodes that can write configuration files and execute pro-

[4]http://origo.ethz.ch/wiki/origo_api

cesses. This allows to integrate any application into Origo. The way developers are working evolves and the configuration management servers that are used today might be replaced by new servers that have different functionality in the future; using custom nodes allows such evolution and adaption of the back-end.

3.2 Scalability

Nodes of the same type perform the same service within Origo. Nodes receive messages and then return the results to the node that sent the message. They are not aware of the other nodes of the same type within their group of nodes. Nodes have unique identifiers inside the peer-to-peer back-end and can be addressed using this identifier. How to balance load among nodes of a same type can vary and it is left to the implementer of the node type. As a simple solution, round robin load balancing is used for the core and storage nodes.

3.3 Extendibility

Two attributes of Origo allow its adaption. The first capability lies in the nature of the nodes. Nodes do not need to be on one single computer; they can be distributed across a number of machines. Nodes of the same type can be on different machines. By measuring load and performance on the running platform, a favorable distribution across machines can be found.

The second attribute making it possible for Origo to react to change is that nodes can join the back-end at any time. If at one point in time the number of nodes of a certain type is not sufficient anymore (this can be caused by increased load for example) - more nodes of that type can be started. These nodes will then register themselves within the back-end and start processing messages. The inverse case of nodes leaving a system can also happen and can be used to react to changing needs. This mechanism is also used to update the running platform dynamically whenever new or updated nodes are becoming available.

3.4 Language Independence

The Origo back-end uses the JXTA [18] peer-to-peer protocol and benefits from the multiple implementations of JXTA that make it language independent. Both the communication inside the back-end as well as the interface to the outside are language independent. The messages exchanged are simple key value pairs of strings that are sent from one node to another. The calls Origo can receive from the outside world are similar and reach the system using the XML-RPC [25] transport protocol.

Chioce of P2P Framework

The available peer-to-peer frameworks today include Chimera (was Tapestry),[5] Pastry [20], Chord[6] (distributed hash functions and the Self-certifying File System [7]), GNUNet[8], XNap[9], and the Peer-to-Peer Trusted Library[10]. Some of these systems address the necessary P2P networking requirements sufficiently, others provide routing algorithms that are adapted to a specific peer-to-peer application (like for example file sharing). None of these frameworks, however provide an application construction framework. A peer-to-peer application construction framework is general and abstract enough to support building P2P applications that go beyond file sharing and instant messaging. The existing application construction frameworks are Juxtapose (JXTA) [18], Jini [24], and OogP2P[11]. Jini is implemented only for the Java language and OogP2P does not provide sufficient functionality to be used in Origo. This justifies the use of JXTA for implementing the back-end.

JXTA has extensive functionality. The JXTA specification contains protocols for routing, message passing, discovery of peers and support for secure communication using HTTPS. In addition, multiple language bindings (for Eiffel, Java and C) exist. JXTA is open-source, widely used, well supported, easy to extend and is general and abstract enough to support the functionality required by Origo. With JXTA, Origo propagates the messages. JXTA implements all routing and other communication protocols for such exchanges in Origo in a technology-independent fashion.

4. RELATED WORK

This section presents other existing software development platforms as well as other service integrating middleware architectures.

4.1 Development Platforms

Software development platforms are not new; the most popular platform known today is Sourceforge [22] with 155'000 projects and 1.5 million registered users. Other platforms are Google Code [13], BerliOS [16], GForge [6], Savannah [12], Trac [11] and Collabnet [9]. A state of the art study [3] summarizes and compares the functionality and services offered by these systems. Many more platforms and systems are examined in the study, but the present selection includes the most significant ones.

Considering the history of the platforms, most of them originate from the initial open-source version of the Sourceforge platform. Both GForge and BerliOS as well as Savannah are branches of its code base. The architecture of these platforms did not allow future extension and adaption - for example when a configuration management server system should be complemented by a new one. This happened when Subversion became more popular with development teams than the former CVS based solutions. Sourceforge today offers Subversion repositories for the code, but the quality of service is not constant and it took the platform more than a year to integrate. With the custom node such a change can be realized much faster for our platform.

Neither GForge, Sourceforge, nor Savannah offer and API.

BerliOS is running a GForge instance that is maintained by the Fraunhofer Institute for Communication. GForge, Sourceforge and Savannah do not host closed-source projects.

Trac is an open-source software development platform that integrates a Subversion repository with an extendible project tracking wiki. Trac does not have an API and Trac instances are for one development project only.

[5] http://current.cs.ucsb.edu/projects/chimera
[6] http://pdos.csail.mit.edu/chord
[7] http://www.fs.net/sfswww
[8] http://www.ovmj.org/GNUnet
[9] http://xnap.sourceforge.net
[10] http://sourceforge.net/projects/ptptl
[11] http://www.duke.edu/ cmz/p2p

The platforms by Google and Collabnet are both closed-source, have no API and do not offer hosting closed-source projects.

None of the platforms offer or enable any integration into development environments. For the platforms that are closed-source it is difficult to know if their architectures scale and to support extension or adaption. Besides Trac and GForge none of the open-source platforms are built with extension possibilities.

4.2 Middleware Architectures

An important part of Origo is its architecture that separates the external services needed in Origo from the back-end of the platform. This has two effects: (1) it allows to scale up the platform if needed, (2) it allows the integration of different implementations of a given service.

Middleware for dynamic adaptation flourished in the past years. As an example, Linda-like [7] coordination media such as AOS [4] use a shared tuple space to decouple applications. Origo is by nature more flexible as it takes into account replicated nodes.

Contrary to systems like HydroJ [17], LuckyJ [19], Fractal [5] or Service Groups [21] where the final recipients of messages are chosen by the infrastructure, Origo defines the entry points but lets the node types define their own policies to treat messages. Origo is similar to Matrix [1] in that it specializes in providing an infrastructure to build distributed applications of a particular type. In the case of Matrix however, the applications that are built have precise requirements in terms of distribution (load balancing of the servers during peaks) while in the case of Origo the main requirement is extendibility. In the end, Origo is much more expressive than Matrix (that could be coded using Origo nodes).

Origo nodes integrate already existing applications and make them communicate independent of the language in which they were programmed; this is a very common characteristics for a middleware infrastructure like CORBA [15] or web services [10, 8]. . .). Comparing to other infrastructures, Origo nodes include the necessary tools to integrate business applications such as identity management and information controller components. They also rely on a language independent communication layer (XML-RPC provides the external API, JXTA enables internal communication). This enables us to focus only on the parts of the system that really matter: the application logic and applications to compose using Origo nodes.

5. CONCLUSIONS AND FUTURE WORK

Origo bridges the gap between coding and publication in software development projects. It brings together development teams and offers them an information management platform that is easy to use and integrates - thanks to the API and the IDE plugins - directly into the development process. Accelerating the publication of software releases improves software quality. The overview of work items of all projects facilitates working on multiple projects and with different development teams simultaneously. The open-source platform Origo allows hosting closed-source projects as well. This makes the platform not only attractive and useful for the classic distributed software development projects known - the open-source projects - but also for all other groups of developers working collaboratively without wanting to disclose their sources.

With Origo currently managing 1000 users and hosting 300 projects from all over the world, ranging from commercial products to university projects the platform stands the test of reality.

Future work on Origo focuses on three main goals: Implementing explicit and implicit creation and detection of development communities and to provide specific communication platforms for them. The second goal is to improve the display of projects; the current listing of projects can take into account activity measurement connected to work items, number of developers on a project and web statistics. For users looking for a certain project on the platform a ranking of the projects can improve the usability. Detecting similarities among projects and suggesting them to users for consideration can be a valuable information source within communities and we plan to offer that. The third goal is to maintain and improve the running platform driven by the reported issues - one popular demand we are considering is the inclusion of a distributed configuration management service.

6. REFERENCES

[1] R. K. Balan, M. Ebling, P. Castro, and A. Misra. Matrix: Adaptive middleware for distributed multiplayer games. In *Middleware*, pages 390–400. Springer, 2005.

[2] T. Bay, P. Eugster, and M. Oriol. Generic component lookup. *Lecture Notes in Computer Science*, 4063:182–197, June 2006.

[3] T. G. Bay. Software development platforms state of art analysis. 2005.

[4] L. Bradford, S. Milliner, and M. Dumas. Experience using a coordination-based architecture for adaptive web content provision. In *COORDINATION*, pages 140–156. Springer, 2005.

[5] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The FRACTAL component model and its support in java. *Softw, Pract. Exper*, 36:1257–1284, 2006.

[6] G. L. L. C. Gforge. *http://gforge.org*.

[7] N. Carriero and D. Gelernter. Applications experience with Linda. *ACM Sympos. on Parallel Programming*, July 1985.

[8] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1. *http://www.w3.org/TR/wsdl*.

[9] Collabnet. Collabnet. *http://www.collab.net*.

[10] W. W. W. Consortium. Simple object access protocol. *http://www.w3.org/TR/SOAP*.

[11] Edgewall. Trac. *http://trac.edgewall.org*.

[12] GNU. Savannah. *savannah.gnu.org*.

[13] Google. Google code. *http://code.google.com*.

[14] Google. Google custom search engine. *http://google.com/coop/cse*.

[15] O. M. Group. *The Common Object Request Broker Architecture: Core Specification, Version 3.0.3*. OMG, 2004.

[16] F. Institute. Berlios. *http://www.berlios.de*.

[17] K. Lee, A. LaMarca, and C. Chambers. Hydroj: object-oriented pattern matching for evolvable

distributed systems. In *OOPSLA '03: Proceedings of the 18th annual ACM Conference on Object-Oriented Programing, Systems, Languages, and Applications*, pages 205–223, 2003.

[18] S. Oaks and L. Gong. *Jxta in a Nutshell*. O'Reilly & Associates, Inc., Reading, Massachusetts, 2002.

[19] M. Oriol and G. Di Marzo Serugendo. A disconnected service architecture for unanticipated run-time evolution of code. *IEE Proceedings-Software, Special Issue on Unanticipated Software Evolution*, 151:95–107, April 2004.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–??, 2001.

[21] S. Sadou, G. Koscielny, and H. Mili. Abstracting services in a heterogeneous environment. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, pages 141–159. Springer, 2001.

[22] I. SourceForge. Sourceforge. *http://sourceforge.net*.

[23] B. Strasser. Vampeer. *http://vampeer.origo.ethz.ch*, 2007.

[24] Sun Microsystems. JINI Connection Technology, 1999.

[25] XML-RPC. Internet remote procedure call. *http://www.xmlrpc.com/spec*.