Report

Origo roadmap

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

Publication Date:
2011

Permanent Link:

https://doi.org/10.3929/ethz-a-006784578 →

Origo Roadmap

Till Bay, Manuel Oriol and Bertrand Meyer

Chair of Software Engineering, ETH Zurich, 8092 Zurich, Switzerland
till.bay@inf.ethz.ch, manuel.oriol@inf.ethz.ch, bertrand.meyer@inf.ethz.ch

Abstract. Developing software systems is a complex activity that involves numerous tasks that are critical for a successful release. Increasing size of software systems, software maintenance, versioning and distributed global development combined with ever growing expectations from users complicate development. Using a well-designed software development platform one can automate and simplify considerable parts of the process. In our research we build the Origo Software Development Platform. The platform integrates into the development process of any team with its API that can be accessed from any development tool and it reacts to the ever evolving toolset used in todays development by providing a framework for integrating new tools.

1 Introduction

In this document we describe the Origo platform. Origo is a middleware framework used to combine different web applications to become one integrated platform. We first talk about the architecture of such an integrated platform that uses Origo. To start we are not defining what kind of integrated platform this is, but talk about components that are common in any integrated platform built with Origo. We go into detail of the technologies that are used and show motivation for the choices taken. Having shown the architecture we take a look at two design principles that are constantly pursued in the development of Origo. After this we take a look at a concrete instance of an integrated platform built using Origo. The platform we are showing combines a set of applications used for distributed software development. We are also showing how to modify a platform that already integrates a number of applications and we show how an integrated platform built with Origo fits into an existing workflow. We conclude with a development plan.

2 Architecture

Using Origo one combines existing applications to become one integrated platform. In the following the fundamental architecture of Origo is de-

scribed. It is this architecture that enables the combination of existing applications.

2.1 Communication and life-cycle

Communication between the different applications forming a bigger system is the most important aspect of Origo. The applications interact using each others data and exchange status information that is then reflected in the state of the entire platform. The exchange of data is multilateral, as one data item possibly can be used for further processing by more than one other application. One pattern for this communication we considered is the Blackboard [1]. A Blackboard would provide a central store, where applications could deposit their data and others could go and read that data. This way of working with data is also found in peer-to-peer systems where one peer aims to advertise information of its state and data to (possibly) many other peers. Using Origo it should be possible to integrate applications that are running on different platforms just as it should be possible for an integrated platform to scale up when need for this arises. These two reasons (platform plurality and scalability) combined with the nature of data exchange made us look into existing peer-to-peer framework implementations. With the JXTA [12] P2P specification standard we found a framework that we esteem suitable for our needs. Not only exist impressive applications enabling collaborative work, such as Collanos [5] but the standard is also implemented in Java and C thus providing a way to interface to applications written in Java and C with ease. As Origo itself is implemented using Eiffel [28], we made an implementation [23] of the JXTA standard in Eiffel too. With JXTA, Origo is able to propagate data through the integrated platform to all applications that require a certain data item. The routing and all other protocols related to that data exchange are handled by the P2P infrastructure and take place in a platform and technology independent fashion, thus not excluding future extensions of the integrated platform with new applications or technologies. The applications that form the integrated platform and offer their services to it have a life cycle. Their starting and stopping is likewise handled by the P2P framework.

Evaluation of different P2P frameworks In this subsection we are presenting a survey of existing P2P protocols with reference implementations or entire P2P frameworks. This evaluation is the basis for our choice for JXTA. Among the protocols we looked at are Tapestry/Chimera ,

Pastry, Chord/DHash, GNUNet, XNap and The Peer-to-Peer Trusted Library. The Tapestry implementation Chimera [3] provides a structured peer-to-peer overlay. Pastry [19] is a protocol proposed by microsoft research that addresses deficiencies of other protocols. Another protocol that provides the functionality of distributed hash tables is Chord [4] that is based on Self-certifying File System [21]. GNUNet [9] is a framework for secure peer-to-peer computing. XNap [25] is a peer-to-peer program that is extendible to various networks. Finally, The Peer-to-Peer Trusted Library [20] is a C++ based P2P security toolkit using OpenSSL, but it is out of date out-of-date. Where some of these protocols are addressing issues of P2P networking better than others, implement routing algorithms better than others, they share the deficiency, that they are not embedded in an application construction framework. The P2P frameworks that we evaluated are JXTA (Juxtapose), Jini and OogP2P. They are all three general enough that they are not only meant for efficient file sharing or instant messaging, but provide abstract P2P based message passing and routing and the discovery of peers. Among these three frameworks we decided to use JXTA [12] because it is not only implemented for two technologies (C and Java) but is also widely used and provides a platform independent communication layer. Jini [11] is limited to Java and OogP2P [17] is a study project that provides to little functionality for Origo.

2.2 Origo Core

Having presented the underlying framework that manages the communication in Origo, we can now look at the higher level design. Origo is built according to the Model View Controller pattern [14]. This allows separating the presentation of the platform state from the logic and the data. The Origo Core is the platform manager and constitutes the Controller within Origo. The Origo Core is aware of all applications that are running to form an integrated platform. It is the Origo Core that directs the way the running applications are interacting. Just like them, the Origo Core is a JXTA service and exchanges data with the other applications by sending JXTA messages into the P2P network they build together. The Origo Core orchestrates all interactions of the set of applications that are running in an integrated platform. The sequence of API calls on the different applications are stored as use case scripts in the core. Using these use case scripts the Origo Core decides what applications are involved in handling a use case and determines what data is exchanged among them. A use case script also contains information about the API that a specific use

case should export to clients from outside the integrated platform, more about this in section 5. The use case scripts are editable and can contain calls on the entire API that the applications running in the integrated platform provide.

2.3 Security Manager

When building an integrated platform with Origo, a number of external applications are combined. All of them provide one or another form of user authentication, user management and role distribution. Users of such an integrated platform should not need to remember all usernames and passwords to be able to interact with the platform. To address this, Origo has a Security Manager that is implemented as a JXTA service just like Orgo Core mentioned in the previous section. The Security Manager stores all usernames and passwords required by the external applications. Using the web interface (see in section 4.6) passwords and usernames can be registered with the Security Manager. This allows to provide single sign on functionality and thus interacting with the integrated platform is easier. The Security Manager is aware of the external applications and their authentication mechanisms. If a user interacting with Origo has not provided a username and password for one of the applications that exist in the platform, the Security Manager can automatically create a user and a login. This generated username and password is then accessible in the same web interface of the Security Manager. To have finer control on allowed actions of Origo users the Security Manager provides role management facilities. Origo Users can have different sets of rights according to their role - it is recommended to use few different roles as we will see in the example in section 5. What different ways of authenticated and non-authenticated interactions are possible with Origo will also be detailed in that section. Communication within Origo is encrypted and observing the messages that are exchanged in the P2P network is impossible.

2.4 Information Manager

Within Origo we have to be able to store specific information about the state of the system and also about the configuration of the involved components. This happens in the Information Manager that provides an abstraction to a database back-end. The Information Manager constitutes the Model of the MVC pattern used. It can be accessed from the entire P2P network within Origo and is also implemented as a JXTA service.

Like all other services the interaction among the services happens with API calls as will be explained in more detail in section 3.2.

2.5 External Applications

The External Applications are the parts of the integrated platform that are built and maintained by someone else than the maintainers of and Origo platform. They can be any application, web-service or even other integrated platforms. The choice for a set of External Applications is determined by the purpose of the Origo platform that one aims to build. We will see such a choice for a set of application in the example in section 5. To make an External Application available within a integrated Origo platform it has to be a JXTA service offering an XML-RPC API [24] - we will elaborate the choice for the API protocol later in section 5. There are different ways for an External Application to offer such an API.

- Offer an XML-RPC API natively (like Flickr [8], various Google API's, Drupal [6])
- Offer another protocol for API calls (like Java RMI) - the calls will be translated by the JXTA service that has to be implemented to make the External Application available in the P2P network.
- Offer no API (like MediaWiki) - the interaction with the application has to be made in the appropriate way (e.g. by sending correctly formatted http requests - more on this later) and the JXTA service that has to be implemented to make the External Application available in the P2P network has to offer the XML-RPC API to Origo and has to implement the appropriate communication to the Application.

This way it is possible to promote any existing application or web-service to become a part of an integrated Origo platform. We will show for which applications we have already implemented JXTA services and how that works below.

3 Design Principles

Origo and the sample integrated development platform using Origo follow a few design principles. The two most important ones are explained in detail below: Extensibility and integration. Besides these two a third principle can be mentioned - Less is more. When designing any part of the Origo framework or the integrated platform, we always tried to reduce complexity. Less complexity in the web interface, less complexity in

the exposed API and less complexity in all other aspects. If a part of a platform built using Origo should expose complex functionality, it should be directly the external application and not any part of what we develop for Origo.

3.1 Extensibility

Extension of the platform means that a new External Application can be added to an Origo platform, without a complete rewrite of the integrated platform and a complete redefinition of the involved databases and associated queries and API's.

3.2 Integration

Origo does not interfere with the way you develop software. But once the system under development is ready to be released this can be done by using the API exposed by the integrated platform directly from within the tools you are using to develop your software system. Integration means that by providing and API that is general enough, interactions coming from outside the platform are not only possible but encouraged. We are providing and XML-RPC [24] API layer. This makes it possible to quickly integrate Origo API calls into any scripting environment and development tool that you are using. Our efforts go in the direction of integrating the entire API exposed by our Origo instance into EiffelStudio - the development environment for Eiffel [28] technology. But it is more than likely that people coming from other development tools and communities will integrate Origo access into their applications, once they start using the platform to host their projects.

4 Origo Instances

An Origo Instance is an integrated platform that is built using Origo. Our main goal was to build a framework, that allows combining external applications to become an integrated platform for developers. In this section we show how such an integrated platform can be built. We discuss the external applications that are selected, show how to integrate them into an Origo Instance. We then also show how an integrated platform can evolve over time and how a new external application can become a part of the platform as well. The basic parts of the development platform we are building with Origo are shown in figure 1. It is an online platform

that developer teams as well as software users can interact with through various ways (explained in more detail in section 4.2). The development platform is used to host the generated binary releases of the software projects, it hosts the website and the documentation of the project and it redirects to external sources of information. The platform integrates a bug tracking system, a configuration management repository, mailing lists and newsgroups. Origo Core (see Section 2.2) orchestrates the set of these external applications. Figure 1 shows the entire integrated development platform. The grey parts come from the Origo framework and the colored parts show the external applications that are integrated into the platform. The surrounding API layer is one of the ways to interact with the platform and will be mentioned in section 4.2.

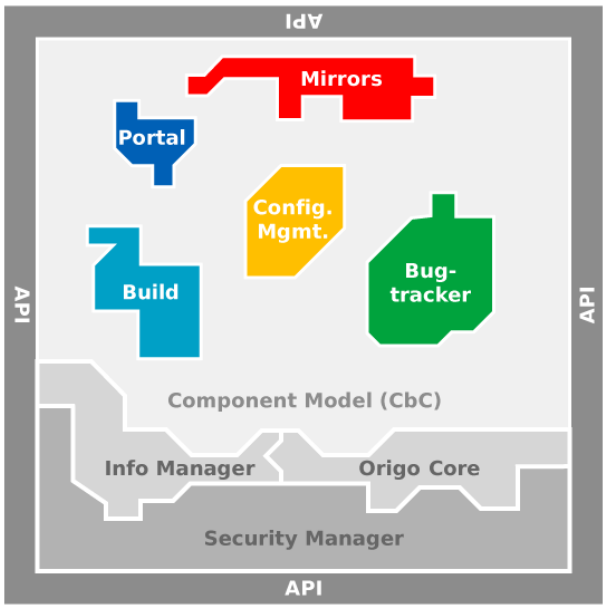

Fig. 1. Development Platform

The external applications used in todays software development process are evolving. A platform that wants to remain competitive and up to date has to provide means for integrating emerging applications. In figure 1 the colored parts are the evolving external applications and the grey parts show what is coming from the Origo framework itself. Integration of a new application will be discussed in section 4.3

4.1 Which External Applications

The Origo instance described here consists of the three parts of all Origo platforms: The Core, the Security Manager and the Information Manager. The External Applications that are used to realize the distributed development platform are detailed below.

Portal The software used to maintain the project pages is MediaWiki [13]. The searching functionality of the MediaWiki is not optimal, but the possibility to maintain a project web page in a collaborative manner combined with the searching capabilities of Origo outweigh this downside. MediaWiki is not only used by Wikipedia, but also used by Eclipse [7] foundation to mention a software development related usage.

Mirrors The current platform is not integrating any specialized mirroring software, but uses a redundant file-server that offers Samba shares to the system. The releases of the projects are stored there and can be downloaded from that file server.

Build The build-server used is the NMI Build and Test Lab [16].

Configuration Management Subvsersion [22] repositories are used.

Bug Tracking As a bug-tracker, Bugzilla [2] is used, but we are thinking about providing a more simple solution alike the Google Code Project [10] as Bugzilla is in many ways too sophisticated for projects of medium size. The possibility to integrate other External Applications with ease becomes more important here, because we are not finally decided on a solution to use.

4.2 Ways to interact with an Origo Instance

Software development involves many different tools and applications. What remains common to the development process is the path from the source code that is being written to the publication of a release. Along this path all the different applications that are used for development come into action. The integrated development platform that we are building with Origo takes a precise look at the different actions along the way and proposes automation and integration where possible. The platform provides a number of interfaces for interaction. Origo Core itself has a web interface that can be accessed with a browser. This web interface provides a user interface for all parts of Origo and for some of the functionality of the external applications - more on this in section 4.6. The second possibility to interact with Origo is the API that the integrated platform exposes. As discussed before, many actions along the path from the code to the publishing of a projects' release can be automated. Automation is

only possible if the integrated platform that is hosting the project provides an API that allows other applications from outside the platform to use it. Defining what API calls are exposed to the outside world is part of the task of the designers of an integrated platform solution that uses Origo. We will show our selection of API calls in section 4.5. The third way of interacting with the integrated platform is to use only the authentication services offered by the Security Manager to gain access to the external applications. After being authenticated, a user can interact with the external applications just the way he would if they were running as standalone applications. A fourth way of using the platform is without authentication and without going over an API invocation. The platform has a web interface that is also available to users that are not logged-in. Actions like searching for a project that is hosted on the platform and downloading a release or submitting a bug report are possible.

4.3 Integrating a new component into an Origo instance

Integration of an External Application into an existing platform is a challenge. Origo provides with the underlying JXTA based network a platform and technology independent communication layer for integrated applications. Applications that should be integrated have to be wrapped as to become a JXTA service, so they can be used in an Origo platform. The wrapping of an External Application is done in the following steps:

- Define and expose the starting and stopping mechanism of External Application for start/stop messages of JXTA service as well as the hooks for authentication.
- Define remaining desired interface that should be exposed by JXTA service to the Origo platform. This is hereafter referred to as the Internal API.
- Connect the External Application to the Internal API exposed by the JXTA service. This connection can happen in various ways. If the External Application exposes and API itself, the connection can be a simple API call translation. If the External Application is not exposing an API, another way of interaction by passing through a proxy that translates Internal API calls to the corresponding actions (e.g. Http requests of the correct format) has to be taken. This solution using a proxy is tedious, but it is to date the only way to provide a way of interfacing to an application that was not forseen all while not patching the application itself.

Once an External Application as a JXTA service, it has to be integrated into the integrated platform. We take the integrated development platform that we are building as an example and illustrate how we can integrate a new External Application called Hype Tool. As Figure 2 shows, the Hype Tool exists inside the integrated Platform as a running JXTA service. Since the platform may wants to use the new application together with the existing ones, the Use Case Scripts that Origo Core is using have to be update and like wise the data that is stored in the Information manager. The Security Manager does not need updating, as it reuses the same JXTA messages to authenticate to all services running within the platform the the providing of the hooks for authentication had to be made when wrapping the Hype Tool to become a JXTA service. These modifications are illustrated with the orange shaded parts of Origo showing in the figure.

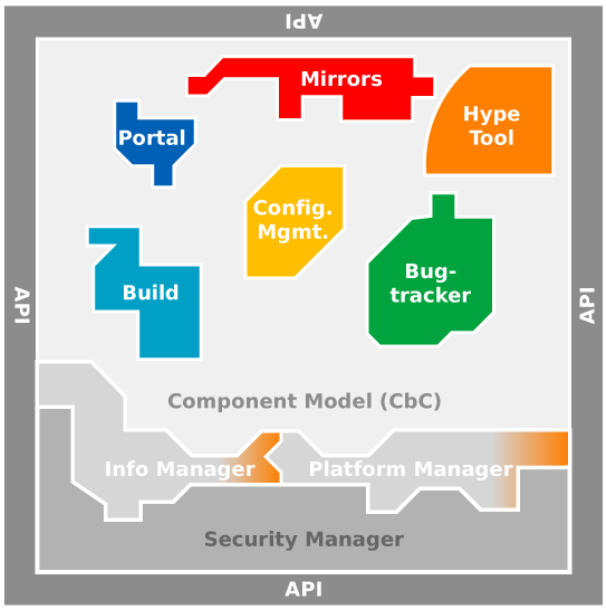

Fig. 2. Extending the Development Platform

4.4 Possible Applications to integrate

In this section we discuss a number of applications that we plan on integrating into Origo. For each of the applications we describe what they do and how we see a possible integration into the platform.

Pacman [18] is a universal Redhat/Debian packaging mechanism for Linux packages - both source and binary packages are generated. Pacman does not offer an API other than its command-line interface. Turning it into a JXTA Origo service would have to be done the way described in section 4.5 about exposing the API of a command-line application.

4.5 API of an Origo Instance

The API that the integrated platform offers is the part of the system that allows integration of the platform into a development teams' process. By providing an API layer that is independent of the tools developers are using to program debug and compile their projects, the platform can be used by everybody. With the choice for XML-RPC [24] for the API a wide range of programming languages already offer frameworks to communicate with the platform. For those that do not provide ready made XML-RPC frameworks, the protocol is using http as transport layer and support can thus be implemented quickly. Origo Core is a daemon running on a port. Accessing the XML-RPC API can be achieved by sending requests to that daemon.

Extending the API: As shown figure 2, the Core has to be updated. By adding a new Use Case Scripts describing not only how the External Applications are interacting with the platform, but also specifying what API the Use Case Script promotes to the outside of the platform, the API is extended. Such and extension is then visible in the web interface of the platform that lists all possible API calls and provides help and documentation for them. The entire API list can also be queried by API.

Bypassing the API for more complex interactions: When wrapping an external app that has an interface to the outside world that a user may want to use directly, there should always be a way to expose that interface to the user. in the case of web applications this is shown by the links to the MediaWiki or Bugzilla instances in the web interface of origo, a click on them brings you directly to their respective web interfaces. In that case the platform served only to authenticate the call. In the case of a command-line application exposing a direct interaction is also possible. The wrapping of the command-line app to a JXTA service includes an API call that allows sending shell commands directly.

4.6 User Interface

The integrated platform that is built using Origo unites many external applications. Each of these applications can possibly have it's own web interface, like for example an installation of MediaWiki that developers are using as project page for their project documentation. But also external applications that have no web interface themselves offer their functionality to Origo and the platform may want to provide its own user interface for managing these applications. When combining a number of applications both with and without user interface to become one integrated platform it is neither possible nor desirable to leverage the entire functionality in the web interface. Successful web applications today present slim, but highly functional UI's. Origo aims in the same direction. The web interface that is presented to the user only allows a subset of actions to be carried out. To be able to use all functionality one has to use the platform's API. The subset of actions that can be taken in the web interface is not fixed. If use of the platform shows that a certain desired action is missing, it is possible to integrate it into the web interface as well. The design restriction that the web interface should remain slim and fast remains however. The applications that provide a web interface themselves continue in Origo to do so. The single sign on functionality of Origo serves as gateway to them. In this case Origo only forwards the interactions taken and the added value is that the authentication is taken care of.

User Interface Example We believe that developing applications not only involves the team of engineers, but should also involve the users of an application. Users are the main source of feedback for the development team. The closer the user community and the developer community can be, the more feedback the developers get. This results in better software that is maintained better and will again attract more users. In the following we are presenting a first version of the web interface of the development platform we are building with Origo. The web interface is not in the state we want it to be, but the layout of the components show the direction clearly and we are able to explain the concepts that guided us. In figure 3 we see the page where the contacts of a platform user are listed. These contacts are developers and users of software alike. Under my software the list of applications that platform user is interested in. This list contains applications that are hosted on the platform and that were selected by that platform user. Whenever one of these projects building one of these applications releases a new version or a patch, the information

about this will be visible to all Origo users that have listed them as an application they are interested in. This way the platform addresses update notification - an issue that especially smaller open source projects still have difficulty with. In the third row Everyones Software is shown. This row displays releases of projects that have been esteemed noteworthy by the entire platform user community. A reason for this can for example be, that a project that has a large number of users has released a new version.

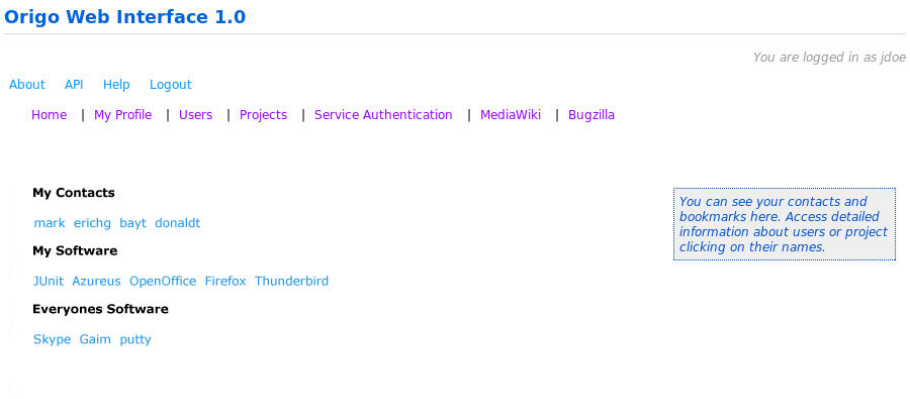

Fig. 3. Origo Webinterface for Contacts

The second aspect of the platforms web interface is the Service Authentication. On this page, a platform user like John Doe in our case can manage the authentication information for external applications that are part of the development platform. Here we see that John has provided login information for a MediaWiki and a Bugzilla that are running inside the platform. He then gets to interact with both of these applications through their own web interface.

The figure of the web interface does not show the Software Tagging functionality. Software Tagging allows both developers and users to annotate releases of a project with micro descriptions. A micro description is called a tag. By tagging a release users and developers can classify it. Tags allow an open classification scheme that is not restricted to the developers. Imagine the following scenario: You are looking for an instant messaging application that you would like to download and install. Both in closed and open source development there exist an big number of projects that release such applications. Obviously you have more requirements than

only that, but you are not sure which of the projects would be the best for you. You need an instant messenger that works on your mac, but also on windows and supports a number of different instant messaging protocols. With normal software categorization schemes you will not be able to find out fast which choice of applications you have. With the possibility for users of a software to contribute tags the developer can share the categorization effort and further more, they can learn about other ways of seeing their application.

5 Using Origo

In this section we are describing ways of using the integrated platform built with Origo. The example shows how an Administrator, a Developer, a User or a Visitor can interact with the platform. We are using a sample project called MyApp. MyApp is maintained by a team of globally distributed Developers and they are using the integrated platform for hosting and releasing MyApp as well as for maintaining the documentation and the Issue Tracking system.

fixmemake a statement on how to build other platforms with the framework

5.1 Use cases

Reporting an Issue

Authenticated				Not Authenticated
Administrator	Developer	User	API	Visitor
⋆	⋆	⋆	⋆	⋆

Reporting an Issue should always be possible, even if the reporter is not registered on the platform. To report an issue we are also providing an API. This is mainly for projects migrating to the platform, so they can enter their existing issues into our system. The web interface for reporting issues for Visitors contains a captcha in order to prevent misuse.

Developing software

Authenticated				Not Authenticated
Administrator	Developer	User	API	Visitor

Developing of the software happens outside the platform.

Reading documentation

Authenticated				Not Authenticated
Administrator	Developer	User	API	Visitor
⋆	⋆	⋆		⋆

Anyone can read documentation, but there is no API for it.

Editing documentation

Authenticated				Not Authenticated
Administrator	Developer	User	API	Visitor
⋆	⋆	⋆	⋆	

Editing Wiki pages is possible for authenticated users, there is API for it that allows automated edits on newly created Wiki pages.

Uploading a release or a fix-release

Authenticated				Not Authenticated
Administrator	Developer	User	API	Visitor
⋆	⋆		⋆	

Only Administrators and Developers can upload releases.

Granting API access

Authenticated				Not Authenticated
Administrator	Developer	User	API	Visitor
⋆			⋆	

Only Administrators can grant API access to other applications that want to interact with the platform. Every application has to request and API key in order to being able to issue API calls.

Adding a Developer to a Project

Authenticated				Not Authenticated
Administrator	Developer	User	API	Visitor
⋆			⋆	

Only Administrators can promote a User to become a Developer of a project.

Downloading and searching projects and releases

Authenticated				Not Authenticated
Administrator	Developer	User	API	Visitor
⋆	⋆	⋆	⋆	⋆

Everybody is allowed to search projects and download releases from the platform.

Creating a Project

Authenticated				Not Authenticated
Administrator	Developer	User	API	Visitor
⋆	⋆	⋆	⋆	

Every User can create a new project, he will then be promoted to be Administrator of the said project.

Modify project information or archive a project

Authenticated				Not Authenticated
Administrator	Developer	User	API	Visitor
⋆				

Only Administrators are allowed to modify the project information, currently there is no API planned for this.

Add a label to a release

Authenticated				Not Authenticated
Administrator	Developer	User	API	Visitor
⋆	⋆	⋆	⋆	⋆

Everybody can add a new label to a release. This improves searchability an makes classification of the software easier.

5.2 Origo Security Model and Sessions

Analog to the design of the entire platform the security model is kept simple. A project hosted on the platform has administrators, developers and users. All other visitors of the platform are not authenticated and can only execute actions like downloading a release, reading documentation or tag a project. Registered users can submit bug-reports and edit wiki pages. Developers can commit code and perform all other tasks that are

possible but they cannot archive a project, promote users to developers or block users. Administrators can perform all tasks for a given project. As mentioned in the introduction, the communication within the platform is encrypted. The JXTA messages as well as the XML-RPC calls are exchanged over Https. An authenticated interaction with Origo creates a session. The authentication and the creation of the session token are handled by MyProxy [15]. The Security Manager takes an authentication request, looks up what kind of session should be created (a session for a user, a developer or an administrator) and then forwards that request to an instance of MyProxy that is running as an External Application wrapped as a JXTA service inside the platform.

5.3 Comparing the development platform to the Competition

Platforms for distributed software development and for publication of the releases have existed for some time now. In this section we are discussing two of them and show how interaction with them takes place. For a comparison to other platforms we refer to the state of the art analysis [27].

SourceForge.net

The most popular one among those platforms is SourceForge. The number of hosted projects and the selection of service components it integrates have contributed to its success. The platform has the biggest number of users and projects hosted on a single platform. The high availability of the platform as well as its design make it difficult for SourceForge to integrate emerging technologies quickly. An example for this is the integration of Subversion repositories into the platform. SourceForge has suffered for a long time of performance bottlenecks on their CVS repositories. This issue combined with the availability of a new, improved source configuration management system - Subversion - made the need to move to a new technology urgent. But despite this need it was really difficult for the platform to integrate Subversion and it took considerable time. The other issue with SourceForge is that it is very hard to integrate it into a development process. The platform offers no API and it is not really offering a very convenient web interface either. When releasing new software a developer has to upload the files belonging to a release to an anonymous ftp server account. Then he has to log on to SourceForge using the web interface, go to a designated page that shows the contents of the anonymous ftp repository and allows selecting files for a release.

6 Development Plan

6.1 Alpha

We believe that a platform can only become better if it is used by its developers. After all developing our own open source libraries (e.g. EiffelMedia [26]) was the impulse for Origo. Today we are using the External Applications mentioned in this document in a loosely coupled fashion. As described in the previous section an integrated platform has many advantages over not only this way of using applications for development, but also compared to other existing development platforms. Nevertheless it is correct to say, that the integrated platform we are building with Origo is in Alpha state.

The parts that are available are:

- Single sign-on for any web application
- First version of a web interface (Web 1.0)
- An XML-RPC API for all implemented functionality of the Single sign-on service
- Framework for generic search including real life examples
- P2P framework

6.2 Beta - Spring 2007

- Origo Core capable of running the Use Case Engine
- Second version of web interface (Web 2.0)
- More XML-RPC API
- Framework for generic search uses search of all External Applications
- Integration into development environment EiffelStudio

6.3 Future work

Future work includes integration of more External Applications like a complete file mirroring software for example. Interfacing with existing development platforms such as Sourceforge and Freshmeat would enable unifying the software directories that exist today.

References

1. Blackboard pattern. Available online under: http://www.vico.org/pages/PatronsDisseny/Pattern Blackboard.
2. Bugzilla. Available online under: http://www.bugzilla.org.

3. Chimera. Available online under: http://current.cs.ucsb.edu/projects/chimera.
4. Chord. Available online under: http://pdos.csail.mit.edu/chord.
5. Collanos workplace. Available online under: httphttp://www.collanos.com.
6. Drupal. Available online under: http://drupal.org.
7. Eclipse foundation. Available online under: http://www.eclipse.org.
8. Flickr api. Available online under: http://www.flickr.com/services/api/.
9. Gnunet. Available online under: http://www.ovmj.org/GNUnet.
10. Google code. Available online under: http://code.google.com.
11. Jini. Available online under: http://www.jini.org.
12. Jxta (juxtapose). Available online under: http://www.jxta.org.
13. Mediawiki. Available online under: http://www.mediawiki.org.
14. Model view controller pattern. Available online under: http://en.wikipedia.org/wiki/Model-view-controller.
15. Myproxy. Available online under: http://grid.ncsa.uiuc.edu/myproxy.
16. Nmi build and test lab. Available online under: http://nmi.cs.wisc.edu.
17. Oogp2p. Available online under: http://www.duke.edu/ cmz/p2p.
18. Pacman. Available online under: http://www.archlinux.org/pacman.
19. Pastry. Available online under: http://research.microsoft.com/ antr/Pastry.
20. The peer-to-peer trusted library. Available online under: http://sourceforge.net/projects/ptptl.
21. Self-certifying file system. Available online under: http://www.fs.net/sfswww.
22. Subversion. Available online under: http://subversion.tigris.org.
23. Vampeer - jxta p2p framework for eiffel. Available online under: http://origo.ethz.ch/index.php/VamPeer.
24. Xml remote procedure call. Available online under: http://en.wikipedia.org/wiki/XML-RPC.
25. Xnap. Available online under: http://xnap.sourceforge.net.
26. T. G. Bay. Eiffelmedia. Available online under: http://eiffelmedia.origo.ethz.ch.
27. T. G. Bay. Software development platforms state of art analysis. 2005.
28. E. Software. Eiffel. Available online under: http://www.eiffel.com.