SHAREXP: AN ECLIPSE PLUG-IN FOR EXPERTISE SHARING AMONG DEVELOPERS

Grégory Bourguin, Arnaud Lewandowski, Myriam Lewkowicz

SHAREXP: AN ECLIPSE PLUG-IN FOR EXPERTISE SHARING AMONG DEVELOPERS

Grégory Bourguin
Univ Lille Nord de France, F-59000 Lille, France, ULCO, LISIC
Maison de la Recherche Blaise Pascal - BP 719, F-62228 Calais, France, bourguin@lisic.univ-littoral.fr

Arnaud Lewandowski
Univ Lille Nord de France, F-59000 Lille, France, ULCO, LISIC
Maison de la Recherche Blaise Pascal - BP 719, F-62228 Calais, France, lewandowski@lisic.univ-littoral.fr

Myriam Lewkowicz
Troyes University of Technology, ICD/Tech-CICO
12 rue Marie Curie, BP2060, 10010 Troyes Cedex, France, myriam.lewkowicz@utt.fr

ABSTRACT

We nowadays live in a world of tailorable systems in which end-users are able to transform their working environment while achieving their tasks, day to day and over the time. Tailorability is most of the time achieved through dynamic component integration thanks to a huge number of components available over the Internet. In the context of software development, one of the main environments that developers are using worldwide for component integration is Eclipse. In this context, the main issue for developers is not anymore the integration of new components, but how to find the most interesting set of components that will fulfil their needs. Solving this issue could definitely increase their productivity and efficiency. We offer ShareXP, an Eclipse plug-in that allows members of a group to share their expertise, this expertise being embodied in the "assemblages" each of them has built.

KEYWORDS

Tailorability, Components, Software Ecosystems, End User Development, Expertise Sharing, Eclipse.

1. INTRODUCTION

Tailorability is today much more than a research concept and many of the actual widely used software are in fact tailorable. This statement can be verified in many different application domains: for instance image editing with Photoshop and its plug-ins, Business Process Management (BPM) with solutions offering diverse connectors for service integration, computer games with World of Warcraft and the success of it's WoW add-ons which enable users to enhance their playing environment, and, of course, software development with the Eclipse ecosystem and its "everything is a plug-in" approach (Gamma and Beck, 2004). Even if solutions exist and are used in everyday life, there are remaining issues, and a large part of the research work in end-user development tries to propose better tailorable systems. Not surprisingly, most of this research is directed towards new means for integration, since this way of tailoring has long been identified as offering a good equilibrium between the level of tailorability that can be achieved, and the level of expertise or effort that is required to achieve it (Mørch, 1997). A lot of work still remains for understanding how to facilitate this component integration.

However, many systems like Eclipse already propose well-defined integration means, and there is a real huge collection of components which can be easily downloaded and integrated. In this context, the matter for Eclipse users is not anymore how to integrate new components, but how to find the most interesting sets of components that will fulfil their needs. Facing this issue of finding and selecting good components for performing particular tasks, we suggest that it would be helpful for Eclipse users to take benefit of the experience of other users.

In fact, individual skills of the developers and their ability to share and generate knowledge within their communities and social networks play a crucial role for organizations which want to continuously learn. Networks of personal relationships which are created and reinforced through interpersonal conversation are critical in supporting knowledge sharing (Erickson and Kellogg, 2002). It has also been showed in a field study on the use of Eclipse done by Draxler et al. (2011a) where they found that integration work in the Eclipse ecosystem is a social activity where actors rely on their local social network. The diffusion or sharing is often rooted in personal contacts, project teams, work groups or even a whole organization. But what is the knowledge that users of the Eclipse ecosystem could share for their organization to learn? Finding a component does not give any knowledge by itself. In fact, the context of its use makes the difference. And context arises from embodied knowledge, which is information that is uniquely and integrally embodied in the person's personality, creativity, intelligence, perceptions, experiences and relationships (Fitzpatrick, 2002). Embodied knowledge is the essence of expertise. It is difficult to identify a priori what expertise could be shared, but it comes naturally in the course of a conversation, in response to hearing or seeing a connected theme and choosing to contribute (Fitzpatrick, 2002). In Eclipse, an element where knowledge is embodied is the perspective (Springgay, 2001). A perspective represents a particular point of view on the working environment that best suits a specific task. We will show in part 2 how, when faced with new needs, or as their expertise grows, Eclipse users can modify their perspectives: they add new components, modify their graphical arrangement, create new shortcuts in the toolbars, etc. Perspectives thus reflect some part of the expertise of their users, as they are the result of users' embodied knowledge while realizing their tasks. So sharing particular components arrangements with their perspectives is our proposition for expertise sharing in the Eclipse environment.

We offer ShareXP, a sociotechnical system in the sense that we aim at designing usability while supporting sociability (Preece, 2000). By using ShareXP, users are able to browse particular assemblage(s) created by others, to preview an assemblage according to a chosen perspective, to chat about them, and eventually to integrate a full assemblage or some of its (sub-)component(s) in their own environment. By offering these functionalities, we follow three of the guidelines defined by Wulf et al. (2008) to improve component-based tailorability: (1) We offer an "exploration environment" allowing the simulation of the interface, (2) the shared repository (containing the components) is integrated into the tailoring environment, (3) the shared repository is activated directly with only those components that are relevant for a certain tailoring context being displayed (the perspectives).

This paper is organized as follow: after a presentation of the Eclipse ecosystem and the need to support the sharing of components arrangements, we will list related work in this domain of sharing components arrangements. We will then present the specificity or our approach and the plug-in that we offer: ShareXP. An example will illustrate the potential benefits of its use and we will finally discuss the limits and future work.

2. THE ECLIPSE ECOSYSTEM

Eclipse is one of the most used Integrated Development Environment (IDE). Its end-users are mainly software developers organized in teams involved in the realization of complex and multi-facets projects. Eclipse is a platform that offers radical component based tailorability by following the approach named "everything is a plug-in" (Gamma and Beck, 2004). According to this principle, the platform itself consists in a set of plug-ins that take in charge the basic mechanisms for loading plug-ins, starting them, etc. On this core platform, all the functionalities that may be proposed by Eclipse stand as plug-ins – or sets of plug-ins, also called features[1]. Eclipse thus proposes different packages that are dedicated to different purposes; for example, we can find Eclipse IDE for Java Developers, Eclipse Modeling Tools, Eclipse for Testers, etc.

Besides this rich diversity, the openness of the platform has led to the emergence of a huge amount of third-party solutions (i.e. features not proposed nor developed by Eclipse, but by other users/organizations) that can be downloaded by Eclipse end-users in order to customize their working environment and dynamically adapt it to their needs. As a result, Eclipse users usually install many different features dedicated

[1] A feature is a logical unit defining a set of plug-ins (and eventually other features) working together, and on which it depends.

to their different tasks. However, if many features are then available in their Eclipse configuration, users do not use all of them at all time. Only a subset of features is involved in a particular task. In Eclipse, such a subset, a particular assemblage of features, corresponds to the perspective core concept. A perspective controls the presence of elements provided by features and that contribute to the user's interface (windows, action buttons, etc.) (Springgay, 2001). The purpose of a perspective is to present a selection of the functionalities that may be useful in a specific activity. Users can have several perspectives, one for each task they have to perform. Some perspectives are included with Eclipse packages, others are provided with specific complex features. For example, the Java perspective defines a set of tools (package explorer, code editor, etc) that are useful for Java development. One perspective is active at a time, and users can easily switch from one perspective to another as the focus of their work changes. But perspectives are not frozen. A user can adapt and/or create perspectives by modifying their elements (plug-ins to be shown, placements in the interface, available actions in the toolbars, etc.).

All these elements constitute a software ecosystems (Messerschmitt and Szyperski, 2003). As recalled by Draxler et al. (2011a), software ecosystems "consist of an open, extensible software platform that attracts different manufacturers and hobbyists, creating small-scale components, which can be individually assembled by end-users". The Eclipse environment can be considered as one of the largest existing and living software ecosystem: the Eclipse Marketplace centralizes about 1300 different solutions that can directly be downloaded and installed. In this context, the matter for end-users is not how to integrate new plug-ins, but how to find the most interesting set of components that will fulfil their needs.

3. THE NEED FOR SHARING CONTEXTUALIZED EXPERTISE

In order to help users finding their desired features through 1300 available solutions, the Eclipse Marketplace is structured around 47 categories. Each plug-in is presented with a small description, screenshot(s), eventual users' reviews, etc. Another interesting section is the "Favorited by" part that shows a list of names of other users that have marked a solution as one of their preferred one.

One can note that Eclipse Marketplace was originally developed by some users who felt the need to organize the access and the promotion of third-party features. This user-motivated initiative illustrates the important need to reify the notion of component discovery, and to set up adequate sharing mechanisms. This need is now clearly in the scope of Eclipse objectives since the Marketplace has now been adopted by the Eclipse foundation, and is fully integrated in the Eclipse website and platform. However, if this solution is indisputably very helpful for Eclipse users, it has revealed to be not sufficient.

As an example, let's consider what happened a few time ago among a small team of researchers who were developing a prototype in our own laboratory. The prototype was developed using Eclipse and no collaborative tool was used apart the email and Subversion[2] for sharing its source code.

One of the developers (Peter) needed to build some user interfaces in Java using the SWT toolkit. Many Eclipse plug-ins are available on the Internet for this purpose but he did not know which plug-in/feature to choose for performing his task. Peter knew that a colleague (John) involved in the same project had already performed this kind of task (GUI design). Peter went to John's desk and asked him which tool(s) he should use. John opened the adequate perspective in Eclipse and briefly showed the development environment he is using for GUI design. Peter saw a tool that could be very useful for his task and which could correspond to his style of working. He pointed the desired feature to John's screen and asked him for its name. John searched the plug-in name through the list of his installed features, and sent a small mail to Peter. Back to his desk, Peter installed the plug-in, and faced many difficulties to use it properly. It even did not look like in John's perspective. Peter went then back to John's desk to discuss the problem, and they finally discovered that John made a mistake: he transmitted the name of another GUI design plug-in. The problem came from the fact that John had previously tested several features/plug-ins dedicated to GUI design before finding his preferred one, and even if they were not currently used, they were still present in its (technical) working environment (they did not the trick in this project, but may reveal helpful for another one...).

This example illustrates several needs. Firstly, Peter preferred asking John for advices about features to use rather than searching for the good one on the marketplace. In fact, finding the good feature on the

[2] Apache Subversion, http://subversion.apache.org/

Internet is not an easy task: before John had made his choice, he had to browse through several features and preferred testing them rather than simply trust reviews and notations displayed on the marketplace, even if these latter gave him interesting (first) indications about their quality. Secondly, the visual aspect of the feature, and its integration in someone else's environment played an important role in helping Peter to imagine how this feature could fit his own working environment.

These needs are aligned with those presented in (Draxler et al., 2011a) and (Stevens and Draxler, 2010), based on their field study of teams of software developers using the Eclipse ecosystem. One of their issue was "How and based on what information do people modify their personal software installation?". Strongly inspired by the work of Mackay (1990), they underline that this evolution mainly results from end-users "collegial" collaboration. For example, the authors report a usual situation they call a kind of "initiation rite" in which experienced senior Eclipse developers would tell a new team member what plug-ins to install by passing the plug-ins names or even the whole set of artefacts to the newcomer. As a result, these developers worked for a certain time with the same configuration, and their environment often later drifted apart as each one modified its own assembling according to its own experience while realizing its tasks. The authors report another example where Eclipse users sat together at one machine to discuss a problem. They show that the discovery of the colleague's environment also often "accidentally" leads to an exchange of potential interesting plug-ins for performing their task.

More generally, the study presented in (Draxler et al., 2011b) shows mechanisms about how end-users manage what they call "software portfolios". The authors compare practices of developers managing their set of Eclipse plug-ins, with practices of players managing their set of World of Warcraft add-ons. They demonstrate that in both worlds, users rely on the recommendations of other people to compose their environment, and they trust advices coming from their well-known colleagues or friends much more than "anonymous" recommendations found on the Internet or magazines. Finally, a particular composition is often shared among users who are involved in the same kind of task. These studies conclude by pointing "a clear lack of tools for supporting the various practices of collaboration with regard to appropriating/tailoring".

4. RELATED WORK

The Eclipse platform presents strong mechanisms that provide good basis for our purpose: plug-ins repositories can be browsed, plug-ins can be dynamically installed/updated from the platform, and APIs exist for plug-in management and collaboration. However, none of these means support expertise sharing. As a result, some external solutions have recently been built over Eclipse in order to tackle this important issue.

4.1 Sharing features

Peerclipse (Draxler et al., 2009) is a research prototype. It allows sharing and retrieving Eclipse configurations in a local community. The set of features available in the configuration of each member is shared over the community through JXTA, and other members are then able to browse and install them in their own environment.

Sharing configurations at the feature level is a good idea since empirical study (Stevens and Draxler, 2010) has shown that Eclipse users try to share sets of plug-ins at this abstraction level. Indeed, in usual Eclipse installation, the average number is more than 300 plug-ins per configuration. It is clear that it will be hard to find the adequate plug-in(s) inside a set of more than 300 possibilities. As features define dedicated subsets of plug-ins, it seems easier to browse configurations at this level.

However, Eclipse users usually manage around 40 features per configuration. Moreover, a usual scenario may imply a novice user who needs to browse an expert configuration for discovering helpful tools. As an expertized user is usually involved in different parts of a complex project, or in many different projects, there is a high probability that the Eclipse configuration of this expert contains much more than 40 features. Stevens and Draxler (2010) found a maximum of 196 features in a single Eclipse installation. Indeed, browsing such consequent set of features for finding the one(s) that could help in the realization of a particular task may also reveal a complex and painful activity.

A platform like Eclipse is usually used for several projects. Users develop their environment over a long period and the resulting configuration comes from their implication in diverse tasks. Thus, each installed

feature may participate to some of the user's activities, but certainly does not participate to all of them. In other words, each of the 40 features found in an Eclipse configuration does not participate in each task that is or has been realized by its user.

It may be instructive to discover the whole environment of a user who has developed a certain expertise in a particular task. However, the main goal of a novice user is usually to perform his/her specific task, and it is in this context that s/he would like to benefit from an expertise. The whole listing of tools/features installed by the expert while realizing tasks with no direct link with the novice's one is not interesting at that time. It can even submerge the novice user under a huge quantity of information without a direct link with the sought one. To sum up, "simply" browsing the set of features installed in an Eclipse configuration does not permit to provide the useful information which is the context of use of each feature.

From our point of view, the remaining questions are: for which task(s) my colleague used this feature? Which other features did s/he use when solving this task? And eventually, how did s/he assemble these features together?

4.2 Contextualizing features

Yoxos[3] is a commercial product which aims at managing several Eclipse workspaces by introducing the Yoxos profile concept. A Yoxos profile is usually dedicated to a specific task and contains a specific list of features that should be used for executing a specific task. The Yoxos launcher lets users select the profile they want to start and the tool configures an Eclipse workspace according to it, with only the needed features. Users may have access to different profiles to be able to start working with different environment according to specific tasks. When a user starts a profile referencing a feature that is missing in his own environment, the feature is dynamically downloaded and installed through the Yoxos certified shared repository of components that provides a service similar to the Eclipse Marketplace.

Yoxos users have a direct access to the Yoxos repository where they can find components (like in the Marketplace) and can integrate them in their own profiles, thus making them evolve. Yoxos profiles can also be shared. As a result, Yoxos users can browse the profiles of other users, discover the list of features involved in each one, decide to install specific components, or decide to install the whole profile.

As profiles are usually created by expert users and dedicated to specific tasks, this approach helps to discover a set of features from other users in their specific context of use. However, a set of features is mainly described by text; there is no exploration environment that presents the integration of a set of components at the user interface level. It is then less obvious for a user to really understand the context of use of this set of components. As we pointed out in our example scenario and as it was already shown by Draxler, a user usually asks for expertise sharing when looking at the working environment of the expert.

5. SHAREXP

As we described in section 2, a perspective represents the setup of a user and provides links to the list of features involved in a particular task, links between these features for realising this task, and even the placement of the tools at the Graphical User Interface (GUI) level. As Eclipse perspectives can be created and adapted by end-users while performing their task, they are key elements which embody their expertise developed when realizing a specific activity. Drawing from this assumption, we offer ShareXP, a plug-in prototype that supports expertise sharing through perspective sharing.

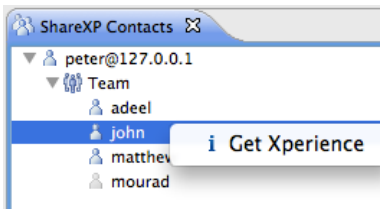
ShareXP has been realized as a standard Eclipse plug-in that can be installed/started by users to enhance their development environment. Following the main trend in Eclipse collaborative development tools like Jazz (Hupfer et al., 2004), ShareXP has been built over the Eclipse Communication Framework (ECF). As a result, ShareXP may well integrate or complement Eclipse workspaces that cover other collaborative dimensions thanks to tools like those offered by Jazz.

ShareXP relies on the eXtensible Messaging and Presence Protocol (XMPP) so that XMPP users do not have to install additional server, and can thus for example use their existing gmail account for sharing with their existing contacts (including their team fellows). The ShareXP contacts view (see figure 1) shows

[3] Yoxos, http://eclipsesource.com/en/yoxos/

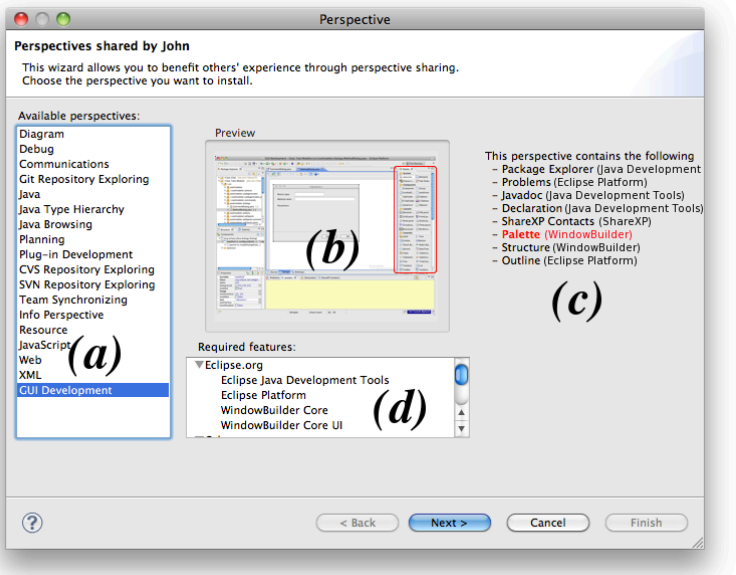
whether other users are online or not. Users can discuss together using their usual chat tool/plug-in, or decide to use ShareXP's included one. As ShareXP uses the existing user's XMPP account, using the included chat plug-in does not involve the creation of a new discussion channel. This approach prevents from introducing a split of the conversation thread between different tools.

Figure 1. View showing user's contacts and giving access to their expertise



Back to our example scenario presented in section 3, Peter and John have now integrated ShareXP in their working environment. Peter knows that John has already tested some features for GUI development. Thanks to ShareXP, he can select his colleague in the contact list to discover his shared perspectives. To share the perspectives of a user, ShareXP serializes their internal description, along with a screen capture of each one. An object representing these perspectives is then sent to the network so that every contact asking for the perspectives of this fellow can get the object describing his perspectives. Figure 2 shows how Peter is able to browse the list of John's perspectives (part (a)). When Peter selects GUI Development, he can see a preview of this perspective (part (b)). As the mouse flies over the different parts of this preview, the wizard highlights the pointed components, their corresponding names and the feature they are packed with(part (c)). Some of the features used by John in this particular task may not provide a specific GUI. Peter can also discover them in part (d) that presents the whole set of features involved in the selected perspective.

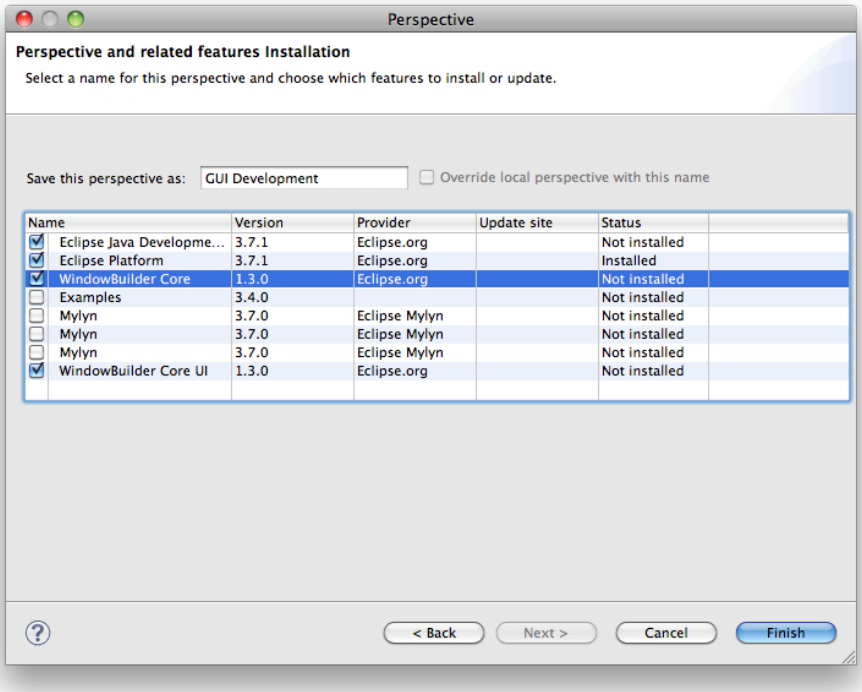
Figure 2. Browsing John's perspectives (a) – with preview (b,c) and related features (d) – helps Peter to find the appropriate one for his task



Once Peter has found the component(s) that best suits his needs, his next step (figure 3) is to select what to install in his own environment. He can select some specific features, or decide to install the whole perspective, including the graphical arrangement of each view, actions toolbars, etc. created by John. ShareXP checks if the features are already available in the local environment, along with their version, and everything is then automatically installed. This installation is performed through the public Eclipse API

dedicated to plug-in installation (Equinox/p2) for the features, and eventually by copying John's perspective in the local environment.

Figure 3. Select which features to install along with John's perspective



This approach presents three major benefits. First, ShareXP allows users to browse and discuss others (whole) configurations filtered by perspectives, so including the context of work. Only the features used in the selected perspective are shown: Peter can only see the features used by John in the specific context of the GUI Development task. This prevents him to have to find his way through a set of many components that are out of his scope. Secondly, the approach encourages the discovering of unexpected features which are directly related to the context of work; As Peter is browsing John's GUI Development perspective, he may discover other types of features (useful for this task) next to the ones he was looking for. Finally, Peter benefits from John's expertise in GUI Development even at the GUI level: the visual arrangement of the useful features is written in the shared perspective; windows are placed at the right places, and toolbars are configured ideally according to John's experience.

6. CONCLUSION

Realizing ShareXP was possible thanks to Eclipse good properties regarding tailoraibility. However, we have to underline a pitfall: Eclipse perspectives have not been designed for being programmatically manipulated or shared. Perspectives are internal objects and no public API is proposed to access their entire definition. We had then to go deep into the code of the platform in order to find the mechanisms needed for reifying perspectives and share them. Considering that perspectives are powerful means that embodies end-users expertise, it would be a great benefit if the openness of the platform was improved by considering them as first class objects which could be programmatically manipulated and shared like plug-ins and features are.

In this first version of ShareXP, we decided to rely on XMPP that proposes synchronous communication. However, this choice can present limitations since users can only benefit from the expertise shared by other users that are connected when the search is performed. It would of course be possible to create local copies of the perspectives of distant users as soon as a member connects to the ShareXP network, and thus support browsing others' configurations even after their disconnection. However, this kind of mechanism was not in

the direct scope of ShareXP since our approach was to complement other already existing synchronous collaborative tools like Jazz (Hupfer et al., 2004). We aim at fostering discussions around personal configurations, like those starting when someone is looking "over the shoulder" of his/her colleague.

We however envision an evolution or merging of tools like the Eclipse Marketplace, Yoxos and ShareXP. As we said before, the Eclipse Marketplace centralizes about 1300 different solutions, organized in 47 categories. Finding an appropriate component is then quite difficult. The current version of the Marketplace or Yoxos repository offers the users the possibility to read/write reviews and to indicate their favourite solutions that can be found in this shared repository. This kind of information clearly tries to capture and share users' expertise. Even if features are categorized and commented by end-users, there is poor information about the specific user task(s) they were used for, either about which other components are involved in the same task(s). Yoxos goes a step further by allowing its users to share profiles. Like in the Marketplace, each feature is described, with notations, comments and a snapshot. However, as we said before, there should be more information concerning its context of use. ShareXP somehow complements this information by also providing a snapshot of contexts of use. A first step forward could for example be to merge Yoxos and ShareXP functionnalities.

More generally, and following the approach we developed in this paper, we believe that it would be possible to enhance actual repositories by capturing, sharing and allowing to browse more information that can be directly extracted from the end-users expertise embodied in their working environments.

It would be a step towards a better knowledge management among developers which could help software development companies to implement effective organizational learning, without adding new dedicated processes, techniques or tools, but "simply" by enriching the developers working environment.

REFERENCES

Draxler S., Jung A., Boden A., and Stevens G., 2011. Workplace warriors: identifying team practices of appropriation in software ecosystems. *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE '11, New York, NY, USA*. ACM, pp. 57-60.

Draxler S., Jung A., and Stevens G., 2011. Managing software portfolios: A comparative study. *End-User Development - Third International Symposium, IS-EUD 2011, Torre Canne (BR), Italy, June 7-10, 2011.*, volume 6654 of Lecture Notes in Computer Science, Springer, pp. 337–342.

Draxler S., Sander H., Jain P., Jung A., and Stevens G., 2009. Peerclipse: tool awareness in local communities. Demonstration at *ECSCW 2009, Vienna, Austria, September 7-11, 2009*.

Erickson T., Kellogg W. A., 2002. Knowledge Communities: Online Environments for Supporting Knowledge Management and Its Social Context. *M. Ackerman, V. P. and (eds), V. W., editors, Sharing Expertise: Beyond Knowledge Management*, MIT Press, pp. 299-325.

Fitzpatrick G., 2002. Emergent Expertise Sharing in a new Community. *M. Ackerman, V. P. and (eds), V. W., editors, Sharing Expertise: Beyond Knowledge Management*, MIT Press, pp. 81-110.

Gamma E., Beck K. 2004. Contributing to Eclipse: principles, patterns, and plug-ins. *Eclipse series*. Addison-Wesley.

Hupfer S., Cheng Li-Te, Ross S., Patterson J., 2004. Introducing collaboration into an applicationdevelopment environment. In *Proceedings of ACM CSCW'04 Conference on Computer-Supported Cooperative Work, Dynamic architectures*, pp. 21-24.

Mackay W. E., 1990. Patterns of sharing customizable software. *Proceedings of the 1990 ACM conference on Computer-supported cooperative work, CSCW '90, New York, NY, USA*, ACM, pp. 209-221.

Messerschmitt D. G., Szyperski C., 2003. *Software Ecosystem: Understanding an Indispensable Technology and Industry*, MIT Press, Cambridge, MA, USA.

Mørch A., 1997. *Three levels of end-user tailoring: customization, integration, and extension*, MIT Press, Cambridge, MA, USA, pp 51-76.

Preece J., 2000. Online Communities: Designing Usability and Supporting Sociability. John Wiley & Sons.

Springgay D., 2001. Using Perspectives in the Eclipse UI. http://www.eclipse.org/articles/.

Stevens G., Draxler S., 2010. Appropriation of the Eclipse Ecosystem: Local Integration of Global Network Production. *COOP'2010, Aixen Provence*, Springer.

Wulf V., Pipek V., Won M., 2008. Component-based tailorability: Enabling highly flexible software applications. *International Journal of Human-Computer Studies, 66(1)*, pp. 1-22.