A NEW FRAMEWORK FOR THE SUPPORT OF SOFTWARE DEVELOPMENT COOPERATIVE ACTIVITIES

Arnaud Lewandowski, Grégory Bourguin
Laboratoire d'Informatique du Littoral, 50 rue Ferdinand Buisson, 62228 Calais Cedex, France

Keywords: Software development support, CSCW, tailorability, inter-activities, Eclipse.

Abstract: Software development is a cooperative activity, since it implies many actors. We focus on CSCW integrated global environments. Many studies have already shown, for a long time, that a 'good' cooperative environment should be able to take into account the users' emergent needs, and should be adaptable. Of course, such properties should also be found in environments supporting software development. However, our study of some existing platforms points out their lacks in terms of tailorability and cooperative support. Eclipse is one of these broadly used platforms. But even if it presents some shortcomings, its underlying framework offers some features particularly interesting for our purpose. Upon results previously obtained in the CSCW field, we propose to extend the Eclipse platform, in order to offer a new support for software development by creating a cooperative context for the activities supported in Eclipse by each integrated plug-in.

1 INTRODUCTION

Following the continuous growth of information technologies, users are looking for systems able to support their intrinsically cooperative activities. And today, these activities tend towards being realized through complex systems supporting this cooperation of actors, distributed through space and time.

The software development (SD) domain, which provides such tools as well as uses them, does not derogate from this rule. Systems are more and more complex, their development requires the cooperation of many actors, with different roles and cultures. Many studies of common practices in SD show how this cooperative dimension holds a strong place in this field (Lethbridge & Singer, 2002)(Pavlicek, 2000). Actually, the necessity to take into account this dimension in SD environments has been underlined for a long time (Kraut, 1995). However, the current systems do not bring to the fore, or according to recent work on CSCW, poorly support the cooperative dimension of these activities.

We have been working for several years on the problems tied to the creation of global and integrated CSCW environments. Our work is inspired by results coming from the Social and Human Sciences (SHS), especially the Activity Theory (AT), and aims at proposing tailorable models and systems according to the expansive properties of every human activity. These thoughts led us to define the Coevolution principle (Bourguin *et al*, 2001).

This paper presents a proposition to better take into account the cooperative dimension in SD tools. Our approach proposes an extension of the Eclipse platform to integrate a cooperative dimension in accordance with the Coevolution principle. The first part of this paper presents the implications tied to the support of SD cooperative activities, by integrating the results of our work in the CSCW domain. Then we present the solution we propose, through a cooperative extension to the Eclipse platform.

2 COOPERATIVE SOFTWARE DEVELOPMENT

Software development environments generally provide a poor support to the cooperative dimension of this activity. From our point of view, adding a cooperative dimension does not simply consist in adding specific communication tools that will bring a new coloration in the environment; rather, it seems more valuable to integrate in a more fine-grained way such tools in the environment, but it raises also many questions. We have been working since years in the CSCW field, and we have to integrate as many results we obtained in this field as possible, if we want our cooperative SD environment to be a 'good' CSCW environment too. We present now the underlying elements of our work in this domain.

DOI: 10.5220/0002493100360043

2.1 CSCW, Tailorability, Coevolution

For years, CSCW research has identified the need for tailorability in the systems. This necessity has been brought to the fore by many empirical and theoretical results, based on theories coming from the Social and Human Sciences, like Situated Action (Suchman, 1987), Ethnomethodology (Dourish & Button, 1998), or Activity Theory (AT) (Bedny & Meister, 1997). Besides, our research is founded on AT, which has been broadly used in the domain over the last ten years (Kuutti, 1993)(Nardi, 1996).

The AT gives information that can help systems designers to better understand the human activities they try to support. We cannot explain here all the results we obtained by founding our research on this theory, neither detail the reasons of our choices. More information can be found in (Bourguin, 2003). However, in order to facilitate the understanding of the paper, we briefly remind some basic elements of the AT on which we based our reasoning.

We use the basic structure of an activity proposed by (Engeström, 1987). This structure presents the human activity as an interdependent system involving a subject that realizes the object of the activity, and the community of subjects who are concerned with this realization. Relations between the subject, the object and the community are mediated. In particular, the subject uses tools to realize the object of his activity. Rules determine what means belonging to the community, and a division of labor describes how the work is shared up by the members of the community. Furthermore, this structure as a whole is dynamic and continually evolves during the realization of the activity. For example, subjects may transform the mediating elements, such as tools, as new needs emerge. Subjects themselves evolve during the activity, acquiring skills and developing some experience of its realization. Thus, when subjects transform the elements participating in their activity, the experience they acquired crystallizes in these elements. This experience, 'written' in the transformed artifacts, becomes available for others, which can benefit from it in other activities.

From our point of view, a system that supports a specific activity is a mediator of it. The system does not contain the activity, but rather takes part in it. A particular system supports a particular activity and this is why we call it an Activity Support (AS). Inspired by the AT, we have defined a set of concepts representing the elements participating in an AS. These concepts are represented in Figure 1. In an AS, a set of subjects are part of a community and realize a task. As defined by Leont'ev and taken up by (Bedny & Meister, 1997), a task is "a situation requiring the realization of a goal in specific conditions". The realization of the task corresponds to the activity supported by the system. Then, in our approach, the task specifies the object and a set of tools and roles involved in the activity. The role represents a part of the division of labor and some rules existing in the activity supported by the system. It defines how a subject may use the tools allowing it to act in the system. As it has been underlined by (Christiansen 1996), the tool enables and limits the actions that may be performed by a subject. This fact is even more true with computer tools that may implement a part of the rules guiding the activity. A task may specify another task. For example, a task may need the realization of another task by other subjects to be completed. This link is useful to represent networks of activities as it has been proposed by (Kuutti, 1993). Finally, when a task specifies another one, a role in the task may imply another role for a subject in the other task.

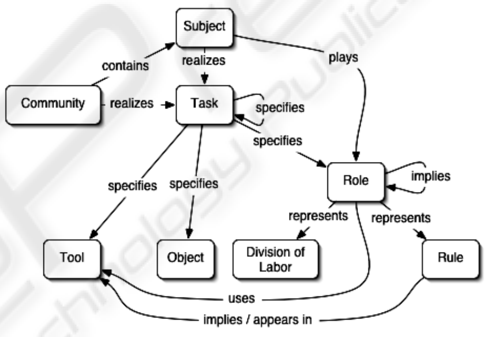


Figure 1: The elements participating in an Activity support.

These elements help us to define an AS. Inspired by the AT, we know (as we underlined before) that all these elements should evolve during the realization of the activity. Therefore, the AS has to be tailorable: it must provide the means to adapt it during the activity it is involved in. This is why we define the AS as a reflective system. In our approach, the task that is supported by the system is made available from the AS. This is realized by introducing particular tools we call meta-tools. The meta-tools allow the subjects to access to the activity definition, i.e. the task specification. In our approach, an AS is considered as the instance of a task. The task can then be inspected and/or transformed from the AS. We define a causal relationship between the task and the AS: transformations of the task have direct repercussions on the system. For example, modifying a role defined in the task directly affects the way the subjects playing this role will perform their activity in the corresponding AS.

This model underlines that, according to the mechanisms described in the AT, a part of the human activity is a meta-activity whose object is to reflect on the activity itself for solving contradictions that may appear in its constituting elements. In our AS model, using the meta-tools correspond to a meta-activity. As the meta-tools are managed in the AS as any other tool, the role of the subjects will affect the way they participate in the meta-activity too and then, the meta-activity is a real cooperative activity that is also supported by the system. This is a simple definition of our approach of tailorability that we have called the Co-evolution (Bourguin *et al*, 2001): the system supports a particular cooperative activity like the development of a particular software, but it also supports its own cooperative (meta-)activity of (re)design.

Finally, this tailorability can be associated with some mechanisms of experience crystallization and reuse. These mechanisms are synthesized in figure 2.

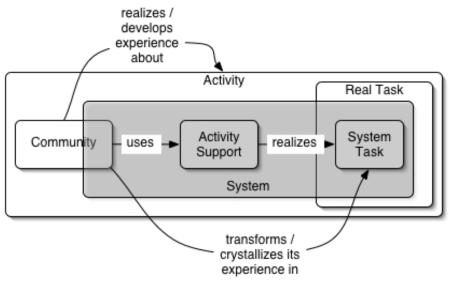

realizes / develops experience about
Activity
Real Task
Community
uses
Activity Support
realizes
System Task
System
transforms / crystallizes its experience in

Figure 2: Experience crystallization through the system and in the system task.

The community realizes an activity in the real world. This activity is supported by the system. The system task is the part of the real task that has been specified inside the system to create an AS. The community acquires some experience while performing its activity. This experience can be made explicit through the system by making its task evolve. For example, an evolution in the division of labor in the real activity may result in a new set of roles specified in the system task and that will affect the corresponding AS. This new evolved task corresponds to a new AS model that can also be instantiated for another community that needs a computer tool support for realizing a similar task. The crystallized experience developed while the realization of a task can then be transmitted through the tailorable system thus supporting an important mechanisms underlined in AT.

2.2 Shortcomings of Existing SDEs

Today, many platforms support SD cooperative activities. Actually, many studies have already been conducted on the evaluation of SD tools and environments. For example, (Barthelmess & Anderson, 2002) focus on process-centered SD environments (PCSDE) found in the literature, analyzing and evaluating them from an AT viewpoint. What emerges from that study is that even if presenting positive aspects, “PCSDEs suffer from the production-oriented philosophy of software engineering”, assuming that the routine steps in a process model are sufficient for the execution of an activity. Consequently, they observe a lack of tailorability in such systems. This observation is relevant not only for PCSDEs, as we will see by examining other environments. In this part, we have chosen to particularly focus on web portals and Integrated Development Environments (IDE) that are widely used by communities of developers during the SD process, in order to point out some of their general drawbacks.

Web portals, like SourceForge (http://sourceforge.net) and Freshmeat (http://freshmeat.net), provide a global environment that integrates many tools, such as planning tools, concurrent versions systems, forums, bug reporting tools, etc. These integrated tools aim at supporting some cooperative activities tied to the SD process. A positive aspect is that these portals mostly take into account the major elements constituting an AS, as we defined them before. Especially, we find in such solutions some mechanisms that define, for example, the role of each community member (by granting them rights on the integrated tools). However, those web environments present some drawbacks, especially with regards to tailorability. Indeed, the latter is in most cases greatly reduced, since the available tools are defined *a priori* in the system. The dynamic integration of new tools is generally not possible. And when it is, this integration remains at a graphical interface level, which actually does not differ very much from using such tools outside the environment. In their study of such Collaborative Development Environments (CDE), or virtual spaces on the web, (Booch and Brown, 2003) recognize that “there are a number of substantial barriers to successful adoption of a CDE”, especially because “no CDE supports all the features” that should be found in an ideal environment. As we underlined before, we think that it is not possible to conceive such an ‘ideal’ AS *a priori*. Instead, a better solution – even if not easy – is to provide a tailorable environment able to be adapted to the needs emerging during the activity. The last important point we emphasized is that the artifacts constituting the AS should crystallize the experience of the subjects. It is especially true for the tools used in SD activities. For instance, developing a web site in php and developing a j2ee application – even if both are SD activities – may imply different development methods, tools, and even roles. The experience acquired during each activity may differ from one to another (methods, tools, etc.). An environment supporting the SD global activities should be able to crystallize these kinds of experience that could be useful in other similar projects. However, this feature seems to be missing in the web-based solutions examined, where it is not possible to reuse the experience developed during a project in another one.

Integrated Development Environments (IDEs), such as NetBeans or Eclipse, also integrate sets of tools dedicated to support producing code activities. Unfortunately, most IDEs only focus on these producing code activities, and avoid or forsake their cooperative dimension. As underlined by (Sarma, 2005), "coding has traditionally been considered the most important activity of a developer in software engineering. As a result, tool builders have focus on creating better programming languages and environments that facilitate coding, while ignoring other activities". Therefore, the many elements identified that constitute an AS are mostly missing from such environments. Actually, IDEs provide gates towards a common repository – such as CVS (Concurrent Versions System) – that supports and manages documents sharing, but not the communication between developers. Some collaborative extensions to Eclipse try then to palliate this lack (Chen *et al*, 2003). But from our point of view, even if needed, this kind of extensions – that provides some collaborative functionalities – still remains superficial, and does not tend to take into account the cooperation at a global level. Eclipse has not been designed in that orientation, and it does not manage any notion of role, or something like this that takes into account the status of a user in the global cooperative activity he participates in. As a result, the user has to integrate the tools (plug-ins) he needs himself, and to configure them according to his role in the real supported activity. Despite this drawback, some of these environments provide functionalities that foster tailorability: for example, Eclipse provides a powerful extension mechanism that allows the platform to be adapted (by integrating new plug-ins) to support new needs. In most environments, like in NetBeans and Eclipse, experience crystallization is reduced to patterns (supporting the creation of different projects) that will configure the environment in a way that seems suitable for such a project. Eclipse gives to the user other means to customize their environments: it manages 'perspectives' that are specific views, or visual arrangements of the tools in the environment. People can create their own perspectives that can be reused later, in similar projects. This feature could be improved by taking into account the cooperative dimension that constitutes an AS.

As we see, lacks remain in the existing global environments supporting some SD practices. Even if the many kinds of environment present interesting features and mechanisms, we note that none of them seems to meet the main 'requirements' of an AS we have identified. In practice (Webster, 2003), due to these lacks in commonly used platforms, the actors of SD use in a complementary way many tools or environments (IDE, PCSDE, web portals, synchronous discussion tool, etc.), each one supporting one or more (sub-) activities. Faced with such statements, we aim at proposing an AS that palliates these needs: a tailorable platform supporting SD cooperative activities, inspired by our previous work on the Coevolution principle.

3 COOLDEV: COOPERATION UNDER ECLIPSE

The CooLDev project (Cooperative Layer for software Development) is directly inspired by results obtained during the DARE project that evolved until becoming CooLDA (Cooperative Layer supporting Distributed Activities) (Bourguin, 2003), the generic underlying platform on which CooLDev lays.

3.1 The Inter-activities Approach

A major choice in our reasoning to design a CSCW environment is to consider that many tools already exist, which are useful in supporting some activities we are interested in. Thus, our main goal is not to create such tools, like a new code editor. Rather, we want to create an environment that integrates these many tools. From our point of view, detailed in (Bourguin & Lewandowski, 2005), each tool supports one kind of activity. When several tools are used in parallel by a group of actors, they generally serve a more global activity than the original activity they were designed for. For example, a group may use in parallel an IRC, a CVS, and a code editor. Each of these tools supports a particular activity (discussion for the IRC, etc.) but they do not know each other. However, they are used in a complementary way by the group since they are used in the context of a global cooperative activity: software development. In such a case, the coherence of the environment is mainly mentally managed by the users. Then, our purpose is to provide an environment that can create a context for the use of the different tools involved in a global cooperative activity (e.g. a SD activity), managing the links between its different (sub-)activities. Assuming that each tool supports a specific activity, our environment is intended to manage what we call the inter-activities.

To achieve this, we have created an activity model (Figure 3)(Bourguin & Lewandowski, 2005) – inspired by the elements presented before – that conceptualizes the elements constituting an AS and that allows the specification of the links of the inter-activities. Each activity is linked to a resource that proposes operations. A resource corresponds to a software tool (an IRC client for example). A user is an actor in the activity, as he plays a role in it, role that allows him to do actions. An activity can be linked to other ones, when the role of one of its actors implies that this user plays another role in another activity. The fact that a user plays a particular role in an activity also has an impact on the configuration of his tools. Finally, an activity is an instance of a task, which constitutes an activity model, or pattern. As we said before – and as we will illustrate later – the task is intended to crystallize the experience developed by the actors.

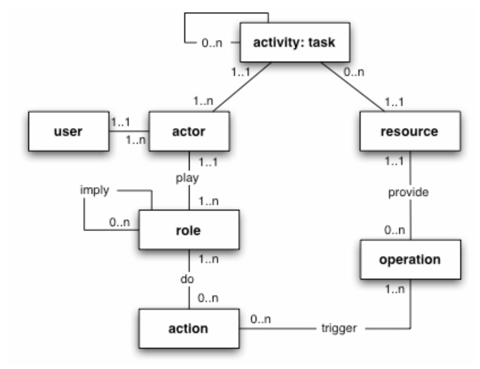


Figure 3: The activity model of CooLDA.

3.2 Choosing Eclipse

Our work in the SD field led us to look closer at the Eclipse platform, which has been adopted by many developers. The success of Eclipse has been a driving force for the development of many tools that can be integrated into the platform as plug-ins. As we have underlined before, we do not aim at developing new tools, but at giving the means to articulate them in a global cooperative activity. From this viewpoint, Eclipse is very interesting since many integrable plug-ins exist.

Basically, the platform is constructed around a core of services called *run-time* that supports the dynamic discovery, installation and activation of plug-ins. A *plug-in* is a component that provides a certain kind of service and respects Eclipse's plug-in specification. It may extend existing features (provided by other plug-ins), implements its own features, and provide extension points (in order to be eventually extended by other plug-ins). This framework allows plug-ins to integrate finely with the environment and other plug-ins. Thus, from our viewpoint, what makes the success of Eclipse is that it has been principally conceived in terms of tailorability. The end-user can adapt the environment according to his emergent needs. He can discover and dynamically integrate tools that can help him to realize his activity.

Another element in Eclipse, which is in tune with our work inspired by the AT, is the perspectives mechanism. A perspective corresponds to a particular visual point of view on the working environment (and the activated plug-ins) during the realization of a kind of activity. This perspective manages the plug-ins activation and arrangement at the user interface level. Eclipse lets the user create and modify his own perspectives, thus saving his preferences for a kind of activity. From our viewpoint, the perspectives mechanisms provide a powerful mean to crystallize the user's experience. However, one can notice that Eclipse's perspectives can only activate plug-ins that are available on the user's station. In other words, if a plug-in is referenced by a perspective but is not installed, it will be skipped. Another point to notice is that these perspectives are not intended to be shared by users. Even if some people may work with the same perspective because it has been packaged within a specific plug-in, no mechanism has been set up in the environment to give these users the means to share their perspective.

Finally, thanks to its introspection mechanisms, and as we will present it thereafter, Eclipse framework provides very useful means to specify and to support the inter-activities. These mechanisms let us dynamically create the links, until now not supported, that exist between the (sub-)activities supported by the plug-ins in a global cooperative activity. Finally, one must keep in mind that at the time we are writing, as mentioned before, Eclipse does not provide the cooperative dimension we need.

3.3 Managing Inter-activities

Our contribution lies within several levels: first, it consists in extending the Eclipse framework by integrating the elements of our model of activity. According to this model, each plug-in supports one activity. The architecture of the solution we propose is presented in Figure 4. We propose to manage the inter-activities thanks to a meta plug-in named CooLDev, whose role is to articulate the other plug-ins in the context of global cooperative activities. This meta plug-in is connected to a CooLDA server, that manages the persistence of the instances of our model. However, each other plug-in – if distributed – is free to use its own communication protocol, server(s), etc. independently.

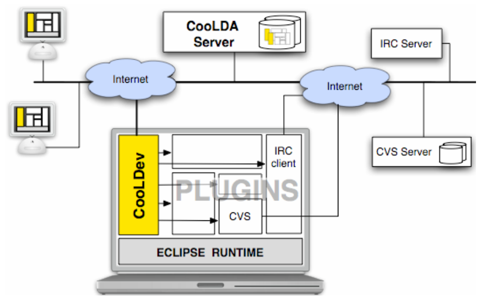
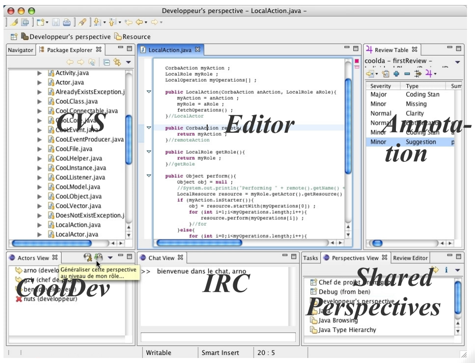


Figure 4: CooLDev's architecture.

The Figure 5 presents an example of such a cooperative global environment for SD that integrates the mechanisms we will describe now.



```
CorbaAction myAction ;
LocalRole myRole ;
LocalOperation myOperations[] ;

public LocalAction(CorbaAction anAction, LocalRole aRole){
    myAction = anAction ;
    myRole = aRole ;
    fetchOperations() ;
}//LocalActor

public CorbaAction remote(){
    return myAction ;
}//remoteAction

public LocalRole getRole(){
    return myRole ;
}//getRole

public Object perform(){
    Object obj = null ;
    //System.out.println("Performing " + remote().getName() +
    LocalResource resource = myRole.getActor().getResource()
    if (myAction.isStarter()){
        obj = resource.startWith(myOperations[0]) ;
        for (int i=1;i<myOperations.length;i++){
            resource.perform(myOperations[i]) ;
        }//for
    }else{
        for (int i=0;i<myOperations.length;i++){
```

Activity.java
Actor.java
AlreadyExistsException.j
CoolClass.java
CoolConnectable.java
CoolCo...java
CoolEvent.java
CoolEventProducer.java
CoolFile.java
CoolHelper.java
CoolInstance.java
CoolListener.java
CoolModel.java
CoolObject.java
CoolVector.java
DoesNotExistException.j
LocalAction.java
LocalActor.java

coolda – firstReview –

Severity	Type	Sum
Major	Coding Stan	
Minor	Missing	
Normal	Clarity	
Normal	[illegible]	
Minor	Coding Stan	
Minor	Suggestion	p

arno (develo...
gzh (chef de p...
ben (developpeur)
nuts (developpeur)

Généraliser cette perspective au niveau de mon rôle...

>> bienvenue dans le chat, arno

Chef de projet (from gzh)
Debug (from ben)
Developpeur's perspective
Java
Java Browsing
Java Type Hierarchy

Figure 5: CooLDev from a particular actor viewpoint.

Because of our meta plug-in, the user has first to identify himself to launch Eclipse, in the same way as classical CSCW tools. Thanks to this, we can retrieve information from our CooLDA server concerning the role of the user in the appropriate global activity. This role is an instance of a type of role that can be shared by several actors. It allows then the meta plug-in to configure the user's working environment. To do this, we have extended Eclipse perspectives, in the context of a cooperative activities management: users that play the same role in a particular activity retrieve an instance of the same CooLDev's perspectives. In other words, users playing the same role can get the same environment configuration thanks to CooLDev's perspectives. However, once they have retrieved their perspective, users can adapt it during the activity. As we will see later, these extended perspectives can also reflect the user's preferences in his role. Moreover, as we underlined in previous part, when an Eclipse perspective was trying to activate an unavailable plug-in, this latter was skipped. In case of a standard use of Eclipse, this is acceptable, since things go otherwise: installed plug-ins are packaged with some perspectives, suitable for their use. In our context, it is the role of the user, and consequently his perspective, which determines which plug-ins will be used in a particular activity. Thus, we have extended the basic perspective mechanism so that it can automatically download plug-ins that are specified in the perspective but not installed on the user's station.

Even if the tools that support the user's activities are instantiated by these extended perspectives, it is not enough to support the inter-activities as we define it. We use our model to specify the actions that have to be processed by a role when its user joins the activity. These actions configure the plug-ins for this user. For example, when a user joins a code reviewing activity, in a development project, CooLDev plug-in uses the user's role to instantiate a CVS plug-in, an IRC plug-in, a code editor and an annotation tool. For the user to avoid identifying another time in the IRC (he has already identified himself while connecting to our environment), we have to indicate to this plug-in what is its configuration (pseudo, server, etc.). From our model's point of view, the user's role realizes actions that trigger operations on the linked plug-in. Technically speaking, the operations are mapped on methods provided by the plug-in and discovered by CooLDA, using introspection mechanisms. We will move later in this paper on the way these links can be set up by users.

3.4 Managing Tailorability and Experience Crystallization

As CooLDev focuses on the inter-activities management, the tailorability it provides sits at this level. First, we take benefit from the tailorability in Eclipse, and we extend it in a cooperative context. Thanks to the plug-ins and perspectives mechanisms, each user can customize the AS by adding, removing and arranging tools that serve his activity. However, in the global cooperative activity, a particular perspective reflects not only the user's preferences, but also his role. Thus, it crystallizes the experience developed in his function. We have set up a mechanism that allows the user to generalize this perspective at the task level, i.e. in his role model. A particular view (Figure 6) we have developed shows the many actors implied in the activity, their role in it, and whether they are online or not. Behind that, this view proposes the mechanism we have described above: the crystallization of the current user's perspective in his role model. Technically speaking, the CooLDev perspective is sent to the CooLDA server that modifies the appropriate activity model also called task by associating this perspective to the user's role instance. Of course, in the framework of the Coevolution principle (that is a cooperative management of the system adaptation), this action can be proposed only to users with a specific role. One can also imagine that the decision to generalize a particular extended perspective in a role has to be negotiated between actors. Thus, when a new user joins an activity with a particular role, he retrieves the experience of users that have already played the same role, experience that has been gradually constructed and crystallized by these users. Finally, if the scenario of the activity allows it, the user can again modify his own perspective.

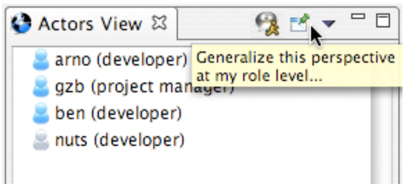


Figure 6: Zoom on the "Actors view" and on its mechanism that generalizes a perspective at the role level.

In order to complete the mechanism that generalizes a perspective in the role model, we have developed a tool (Figure 7) allowing actors to share their perspectives. This view shows the perspectives shared with others, and allows the users to 'try' the perspectives they receive and to send their perspectives to others. Thus, actors can share extended perspectives without having to crystallize them in their role model, which would be constraining for actors who just want to test perspectives, or to share a viewpoint before deciding to crystallize it. Indeed, the experience crystallization in the roles is an important mechanism, since it may have an impact on all the other actors playing roles that are based on the same model.

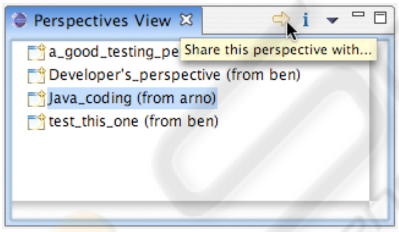


Figure 7: Zoom on the shared perspectives view.

We have also introduced tools that provide tailorability at finer levels of the inter-activities management and help creating links between the tools in our activity model. Thanks to the introspection mechanism, it is possible to (re)define the task during the activity, i.e. the elements that participate in it (for example the actions for each role). The lowest abstraction level is the one of operations that define the links between the actions (of a role) and the methods (provided by a plug-in). CooLDA is able to dynamically retrieve these methods, allowing to (re)define the operations used in the specification of the roles' actions. We are aware that the abstraction level of this kind of tailorability is still hardly within the reach of every end user. However, implementing these mechanisms helps us, at first, to verify the technical feasibility of such an approach. We are now working on user interfaces of higher abstraction level, to give end users the means to access this really fine-grained inter-activities management.

More generally speaking, the evolution of an activity – its tools (integrated plug-ins, operations) and roles (extended perspectives, actions) – is synthesized in the model of activity presented that constitutes the *task*. A task forms thus a model that has crystallized the experience developed cooperatively by users during their global activity, and that can be re-instantiated in order to support other similar activities. These latter, evolving in turn, will be able to modify their task, or to create new ones. This reflexive approach – even if promising – raises also many problems. We cannot develop them here, but one can refer to (Bourguin, 2003) to discover the many stakes in it that we have already identified.

The mechanisms described here present the tailorability and crystallization currently provided in our AS. These mechanisms are not at the same abstraction level. As underlined by (Morch, 1997), the tailorability level increases proportionately with the difficulty for end-users to access it. The basic integration of tools and the mechanisms tied to the extended perspectives are more directly aimed at end users. The evolution of roles and actions are aimed at CooLDev specialists (understanding the AS model), and the definition of operations by introspection on methods at developers (understanding object-oriented concept). At first, we will package CooLDev with a set of predefined tasks. Users will then be able to adapt these tasks according to their needs and to their abstraction level, providing thus new activity patterns. The many abstraction levels, and the cooperative dimension of the supported activities let us hope that the end users, interested in the tailorability levels they will be able to reach, will increase their experience towards the system, and will become more and more expert of it, being able to access more advanced levels of tailorability. They will be able then to share this experience through the system, meeting the Coevolution principle.

Of course, this work still needs improvements. Especially, we work on a proposition to extend the plug-ins model, in order to add a semantic level that should facilitate a finer integration by end users. And in order to validate our approach, we plan to experiment the platform in real projects that will help us to verify the stability of the system with regards to its reflexive properties, from a technical and especially a human viewpoint. Indeed, for the system to be 'correctly' adapted, the activity must pass by stable stages, allowing users to increase their experience. So we want to verify that the human dimension brought to the fore in CooLDev (the system is adaptable, but not self-adaptive) permits, not a continuous evolution of the environment, but rather the crystallization of a true experience for end users.

4 CONCLUSION

As software development (SD) is today strongly cooperative, we focused in this paper on the means that can support it. We have been working for years in the CSCW research domain, trying to take benefit from SHS theories, especially the Activity Theory (AT). This work has led us to identify the crucial need for tailorability in cooperative environments, and to define the Coevolution principle. By studying several platforms broadly used by developers, we have identified their shortcomings, in line with the stakes defined in the CSCW field. Therefore, we have proposed a solution that consists in an extension of the Eclipse platform which is already broadly used for SD, but which does not integrate the cooperative dimension of such activities at a global level.

Basing on results coming from the AT and on Eclipse properties, we propose a model of activity and a meta plug-in that contextualizes the activities supported by plug-ins. We aim at creating a tailorable support for managing the inter-activities and setting up the Coevolution: the system must support SD cooperative activities and its own cooperative (re)design (meta)activity, while fostering crystallization and sharing of the experience developed by its users. Our proposition brings several levels of tailorability intended both to end-users and to users with more advanced skills concerning our platform.

Although it already provides a tailorable support for the inter-activities management, our proposition needs to be developed further and to be tested and validated by experiments in real situations. We have to work on raising its abstraction level. In order to achieve this, we plan to pursue our efforts and to look closer at the problem of the semantic associated to components available on the Internet. Indeed, even if solutions trying to palliate this problem exist, one must agree that most of the existing component models are intended to software developers, whereas the results of studies in many fields show that the means for discovering, and dynamically and finely integrating tools would be useful for end users, as it would take into account *in situ* their emergent needs.

REFERENCES

Barthelmess P, Anderson KM, 2002. A view of software development environments based on activity theory. In *Journal of CSCW*, 11(1-2), pp. 13–37.

Bedny G, Meister D, 1997. *The Russian theory of activity, Current Applications to Design and Learning*. Lawrence Erlbaum Associates, Publishers.

Booch G, Brown A, 2003. Collaborative development environments. In *Advances in Computers*, 59.

Bourguin G, 2001. Les leçons d'une expérience dans la réalisation d'un collecticiel réflexif. In *Actes de la 15ème conférence francophone IHM 2003*, pp. 40-47.

Bourguin G, Derycke A, Tarby JC, 2001. Beyond the Interface: Co-evolution Inside Interactive Systems – A proposal Founded on Activity Theory, *People and Computer vol. 15 – Interaction without Frontiers*, Springer Verlag, Proc. of HCI 2001, pp. 297-310.

Bourguin G, Lewandowski A, 2005. Inter-activities management for supporting cooperative software development, *Proc. of the 14th Int. Conf. on Information Systems Development (ISD'2005)*, Karlstad, Sweden.

Cheng L, Hupfer S, Ross S, Patterson J, 2003. Jazzing up Eclipse with collaborative tools. In *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, Anaheim, California, pp. 45-49.

Christiansen E., 1996. Tamed by a Rose: Computers as tools in human activity, in (Nardi, 1996), pp. 174-198.

Dourish P, Button G, 1998. On “Technomethodology”: foundational relationships between ethnomethodology and system design. In *Human-Computer Interaction*, vol. 13, Lawrence Erlbaum Associates, pp. 395- 432.

Engeström Y, 1987. *Learning by expanding*. Orienta-konsultit, Helsinki.

Kraut RE, Streeter LA, 1995. Coordination in software development. In *Communications of the ACM, 1995, 38*(3), pp. 69-81.

Kuutti K, 1993. Notes on systems supporting “Organisational context” – An activity theory viewpoint, COMIC European project, D1.1, pp 101- 117.

Lethbridge T, Singer J, 2002. Studies of the Work Practices of Software Engineers. In *Advances in Software Engineering: Comprehension, Evaluation, and Evolution*, Springer-Verlag, pp. 53-76.

Morch A, 1997. Method and Tools for Tailoring of Object-oriented Applications: An Evolving Artifacts Approach, part 1, Dr. Scient. Thesis Research Report 241, University of OSLO, Department of Informatics.

Nardi B, 1996. *Context and consciousness: activity theory and human-computer interaction*. Cambridge: MIT Press.

Pavlicek RG, 2000. *Embracing insanity: open source software development*. Indianapolis, Sams Publishing.

Sarma A, 2005. A survey of collaborative tools in software development, Institute for Software Research Technical Report, #UCI-ISR-05-3.

Suchman L, 1987. *Plans and Situated Actions*. Cambridge University Press, Cambridge, UK.

Webster M, 2003. An end-user view of the collaborative software development market. Market Research Report, IDC #30608, Vol. 1, http://www.collab.net