Towards New Links between HSS and Computer Science: The CoolDev Project

Grégory Bourguin, Arnaud Lewandowski

Grégory Bourguin and Arnaud Lewandowski
Université du Littoral Côte d'Opale, LIL
50, rue Ferdinand Buisson BP 719 62228 Calais Cedex
France

Towards new links between HSS and Computer Science: the *CoolDev* project

18.1. Introduction

In 1991, K. Kuutti [KUT 91] started a mini revolution in the field of research related to computer-supported cooperative work (CSCW) by publishing *The concept of activity as a basic unit of analysis for CSCW research*. Wanting to address the problem of the definition itself of CSCW, Kuutti proposed to use a theory from the works of human and social sciences (HSS) and, more specifically, the concept of activity from Activity Theory (AT). However, although today there are more than fifteen years that this theory is involved in CSCW, the question remains as how to best use it to develop groupware [HAL 02]. Many researchers have shown how AT provides a framework to analyse and improve CSCW situations. Although these works are very interesting, the results obtained are strongly related to the case studies and, from the point of view of the computer scientist, hardly reusable in another context. Starting from this observation, we have chosen to tackle the problem in reverse. The conceptual framework of AT describes generic mechanisms, that is elements common to all human activities. Our idea is to use this genericity to identify computer techniques that *a priori* support them. This article will be based on our own experiment in the context of the creation of the *CoolDev* environment, a tool to support the cooperative development of software.

.

18.1.1.*A new support to cooperative activities of software development*

Software development is an inherently cooperative activity. However, many studies have shown that this dimension is always poorly supported in IDEs (integrated development environments) such as *Visual Studio Team System* [SRI 04] or *Eclipse* [IBM 06], which aim to integrate various tools (editor, debugger, code sharing and other) involved in development activities. Trying to fill this gap, we initiated the *CoolDev* project, a new IDE built as an extension of the *Eclipse* platform.

18.1.1.1. *The* CoolDev *project*

CoolDev (Cooperative Layer for software Development) (Figure 18.1), proposes to orchestrate a set of tools involved in development activities. The basic function of *CoolDev* is, for each player of the process, to recreate his work environment based on his role. In a very simplified way, when he starts *CoolDev*, a user is prompted to identify himself, allowing the distributed environment1 to find what his role is (*developer*, *project manager,* or any other role having been defined) in the activity in progress. This has the effect of shaping his client workstation by loading (starting-up, positioning in the interface) and configuring (connections to dedicated servers, rights management, communication channels with the other players, etc.) the tools (editors, sharing documents, instant messaging) that he needs to fulfill his role.

In our example, the user *arno* was identified as *developer*, which had the effect of loading the tool (*CooLDAView*) that allows him for example to see who are the other actors involved, to verify whether they are connected or not and in what role, to start code editor connected to the CVS (Concurrent Version System) server of this activity, in the right project with the proper rights, to start the Chat tool that allows him to discuss with the other actors, etc.

However, *CoolDev* does not simply configure the environment at start-up. Indeed, the actions of each actor can have repercussions in the global environment. For example, updating a portion of code in the CVS server by a developer can lead to the display of a message in the Chat of all the actors and modify the status of the role of *testers* in order to enable them to comment on the concerned portion of the software.

---

1 The environment is both distributed in space (the actors are connected on remote machines) and in time (the actors are not necessarily all connected at the same time).

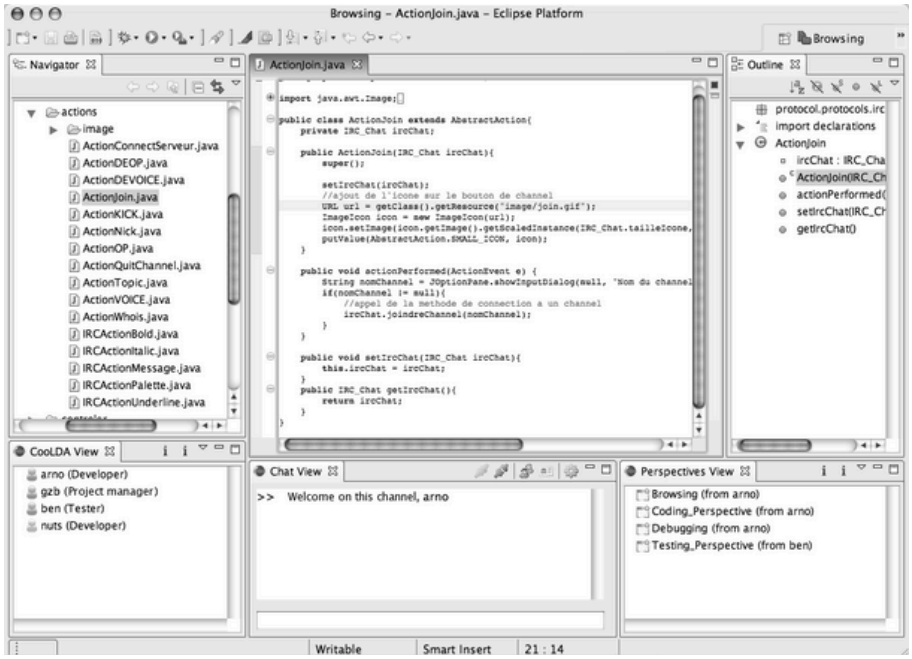

Figure 18.1. *An example of perspective for the role of developer in CoolDev*

We must note that the basic properties of *CoolDev* that are described above are elements that can be found in most environments for software development interested in collaboration. The first interest of *CoolDev* is to introduce this " classical " but missing collaborative dimension within the *Eclipse* IDE. However, our final objective is primarily to create an IDE that brings new properties offering a " better " support to collaboration. In order to define these new properties as well as the means to implement them, we got interested to the results obtained in other projects attempting to create better groupware, especially those dedicated to software development activities.

18.1.1.2. *The role of HSS in groupware design*

Several proposals [BAR 02, KOR 02, SOU 03] have recently emerged in order to better support cooperation among software development tools. If we study the approaches underlying their creation, it is interesting to note that most of them implement theories developed in HSS (AT in the examples mentioned) in order to better understand the human cooperative activities in which the development is achieved. This type of approach is very characteristic of research in the CSCW field.

It is such also from the point of view of existing problems in the necessary relationship between HSS and computer science to create better support.

What holds our attention in these very characteristic examples is that the studies carried out highlight shortcomings related to our problem, but offer results that are very difficult to reuse in our own approach. This problem is illustrated in Figure 18.2. It is established that the creation of a " good " groupware happens in the multidisciplinary encounter between HSS and computer science in the case interesting us. The meeting points between these disciplines are generally made through one-off projects. Just like numerous publications on groupware these studies are more directed towards HSS than towards computer science itself and the elements that come out of them are usually strongly linked to the analysed situation.

Thus, a question remains: what can the computer scientist gain in his way of designing *a priori*, of using and developing his own techniques? It is true that these work in situations. However, it is clear that the development of disciplines involved in a multidisciplinary approach is made, from this point of view, through indirect links, through specific works based on particular projects involved. The idea that we advocate is to imagine that we can today forge more generic links by proceeding directly to an analysis of elements that constitute the successive developments of various disciplines, often through one-off confrontations.

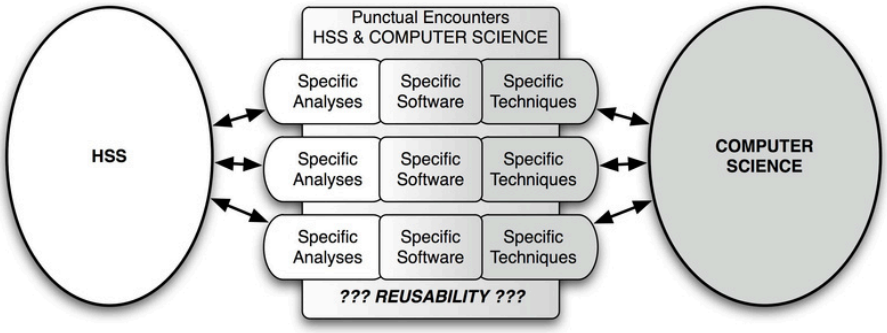

Figure 18.2. *The problem of the reuse of results provided by ad-hoc meetings between HSS and computer science*

18.2. Towards new links between HSS and computer science: application to AT

From our point of view, described in Figure 18.3, the generic link between HSS and computer science are revealed in their respective goals. HSS try to create theories, methods and tools (such as the structure of activity proposed by Engeström [ENG 87] or the checklist of Kaptelinin [KAP 99]) helping to understand human activities. Computer science tries, on its part, to develop theories, methods and tools for creating software that support them.

It appears that the interface between these disciplines is the *human activity* in its generic form, that is the description of points common to all human activities. The questions that then arise are: what are the elements linked to the generic properties of human activity that were identified in each discipline, and what links can be built up between these elements? The goal of any system being to support human activities, we are convinced that these generic links identified can easily be reused in the creation of particular systems. To our knowledge, only one team of researchers close to our area of interest has attempted to forge such type of links. It is the works initiated by Dourish and Button in what they call technomethodology [DOU 98]. However, after a leading article discussing the principle of this approach, we did not find any more advanced developments. It should also be noted that this proposition has its roots in the ethnomethodology while our own works focus on AT.

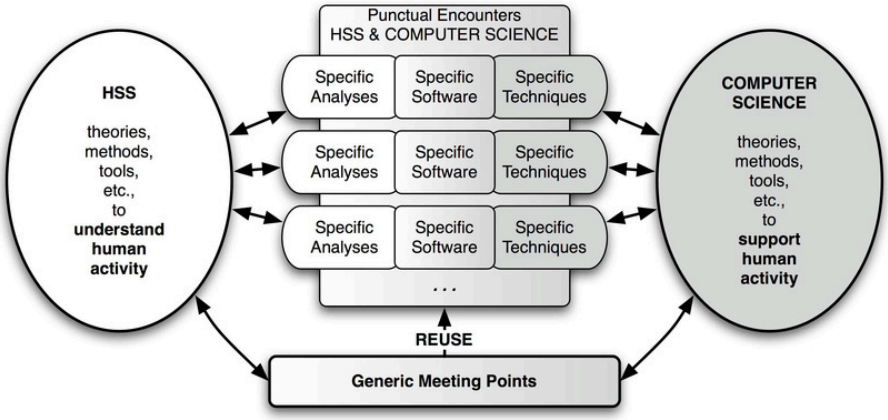

Figure 18.3. *A generic approach to multidisciplinarity*

18. 2..1*Activity Theory*

AT [BED 97, KUT 91, NAR 96] is originated from the Soviet historical and cultural school of psychology founded by Vygotski. Over the years, it has focused on mediation through the tool or instrument and has proved to be a body of concepts whose aim is to unify the understanding of human activity by providing bridges towards other approaches coming from human sciences, in particular approaches in social sciences and those of behaviour. The fundamental analysis unit of AT is human activity which is defined as a coherent system of internal mental processes, of an external behaviour and motivational processes that are combined and directed to achieve conscious goals.

Engeström [ENG 87] defined the " basic structure of an activity ", which is the most classical reference to AT. This model expresses the relation between the subject and the object of the activity. This relationship is reciprocal: the subject realizes the object of the activity, but at the same time, the properties of the object transform the subject by increasing its experience. It is furthermore mediated by the concept of tool representing everything that is used in the realization process of the object.

The tool both allows and limits: it allows the subject to realize the object of its activity, but limits by masking some of the potential of transformation. On one hand, the tool assists in the realization of the object by subjects that use it because it carries with it implicit goals that were set by its developers. On the other hand, it is itself transformed and (re)constructed during the activity. It is often modified or adapted by subjects in a reflexive process and in response to contradictions that may emerge between elements participating in the activity, depending on their emerging needs, their goals, and experience. Thus, this subjects' experience crystallizes in the tool that thus carries in it the cultural heritage of the situation. Mediation by the tool corresponds then to a means of transmitting a certain culture and experience. Moreover, the individual is not isolated but forms part of a community that share the same object of activity. Community-subject and community-object relations are mediated by concepts of rules and division of labour also containing the cultural heritage of the situation. As with the tool, these mediators are open to new developments. More detailed descriptions on our understanding of AT can be found in [BOU 05].

18.2.2.*In search of generic tools for human activities*

The AT has often been criticised [HAL 02], in particular by HSS researchers, because of its approach sometimes considered too general not allowing to reach a certain level of details in the description and understanding of analysed activities.

Paradoxically, it also seems to be the most approved theory by computer technologists, hence its success in CSCW. Indeed, beyond the creation of computer supports for particular activities, one of the major motivations that guides computer scientists is genericity. The creator of software tries in most cases to create a tool aimed to support a more or less generic task that can be instantiated for various users involved in similar activities. For example, in the case of *CoolDev*, our goal is to support various activities of software development so that the tool we create is potentially useful to a maximum of users. In the same way, computer science itself tries to address the generic problems of support of human activities. Thus, part of what the computer scientist seeks in the answers of HSS is a generic description of properties of the activity he tries to support. From our point of view, this corresponds to what provides AT by proposing a model that describes the components and mechanisms common to all activities.

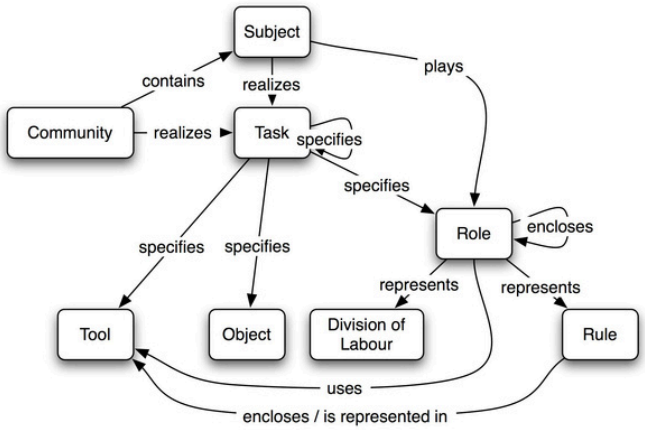

Figure 18.4. *Elements of an ontological model based on AT*

A typical example of this search for generic solution in HSS by creators of groupware is their extensive use of the basic structure of activity. Many works claim being inspired from AT because they propose a conceptual model inspired by the structure of Engeström, very useful for the conceptualisation of a generic support for cooperative activities, as this corresponds to what it describes. We ourselves have used this structure to conceptualise *CoolDev* [LEW 05]. The concepts of role, tools, etc. described in Section 1.1 are to be directly related to the entities described in AT (see Figure 18.4). However, although this example shows that the structure of Engeström allows the information to use AT directly in the design, we do not think that it is a real generic link between foundations of activity and computer science. On the contrary we are convinced that these links can be identified, not by considering this

structure, but by examining the properties and mechanisms that are associated to it in AT.

18.3. Generic links between AT and computer science: application in *CoolDev*

Every software is a tool whose purpose is to equip a human activity. Thus, in AT, the description of the mechanisms surrounding the tool, mediator of activity, and the properties it is supposed to present can help the computer scientist on the fundamental properties of software tools he attempts to create. If we consider that certain properties of software come from the very foundations that were used in their creation, that is techniques that constitute computer science, we then therefore create generic links between computer science and HSS, and hope that techniques supporting generic properties of tools in human activity would allow producing better software. We will now exemplify this subject by highlighting a few particular points from the application of this approach in *CoolDev*.

18.3.1.*Link between generic properties of AT and techniques of computer science*

We have identified certain connections between the properties of the human activity described in AT and computer techniques that allow supporting them. These techniques have been implemented in *CoolDev* following its theoretical analysis (as opposed to an empirical analysis) as part of AT. The outlines of this analysis are the following: *CoolDev*, as a *tool* to support development activities, may be changed by its users (the *subjects*) according to their emerging needs and their *experience* in development. This *experience* should be able to *crystallise* within *CoolDev* so that it can then be transmitted to other users.

Finally, *CoolDev*, as a groupware, contains a representation of other elements of the activity as the roles of actors who reflect certain *rules* and *division of labour*. These elements being also subject to development and to the crystallisation of the experience, their computerized representation should reflect these properties. The techniques implemented to support these generic properties are listed in Table 18.1.

Activity Theory	Computer techniques
Reflective attitude	Introspection, causal relation
Understanding by the subject	Framework, meta-modelling
Evolution, crystallisation of the experience	Intercession, customization, integration, extension
Development of experience	Prototyping
Transmission of experience	Model-driven engineering, components

Table 18.1. *A connection between properties from AT and computer techniques*

To enable subjects to understand the functioning of their tool in the event of contradictions, *CoolDev* uses *introspection* [MAE 87] that allows the system to provide a representation of its own functioning, during the execution. *CoolDev* proposes for example a point of view on the scenarios, the actions available for the roles, etc.

However, these elements correspond to computer objects that are hard to understand for non-computer scientists. This is why techniques of *framework* and *meta-modelling* are very useful. A framework provides a set of computer elements that guide the application towards a particular area.

In our case, the framework of *CoolDev* is a set of objects (in terms of object oriented languages) that participates in the support of development activities. Meta-modelling allows defining one or more languages oriented towards subjects that will allow them to manipulate elements defined in the framework. In *CoolDev*, the meta-model defines for example what a role is, what a tool is, and how these entities can be composed to create a particular activity support. At runtime, *CoolDev* performs the translation between the computer entities of the framework and entities of the language defined by the meta-model.

This translation is made in both directions: from the framework to the meta-model to provide a representation of the functioning of a specific activity support understandable by the subjects; from the meta-model to the framework to make effective within the system the transformation of specifications of an element by the subjects (see Figure 18.5). The maintenance of this link between the representation

provided to subjects and the computer representation that allows the application to run corresponds to a *causal relation*, as defined in computer reflective systems [MAE 87].

With these techniques, subjects have the opportunity to reflect on their activity support and to identify elements causing problems. The intercession technique [MAE 87] makes it possible for this support to evolve, that is the mediators of the activity, by transforming elements that constitute it, under the framework, through a meta-model and during the execution of the system. This intercession can take different forms whose complexity is inversely proportional to the power of transformation. These are the techniques of *customization*, *integration* and *extension* described by A. Morch [MOR 97].

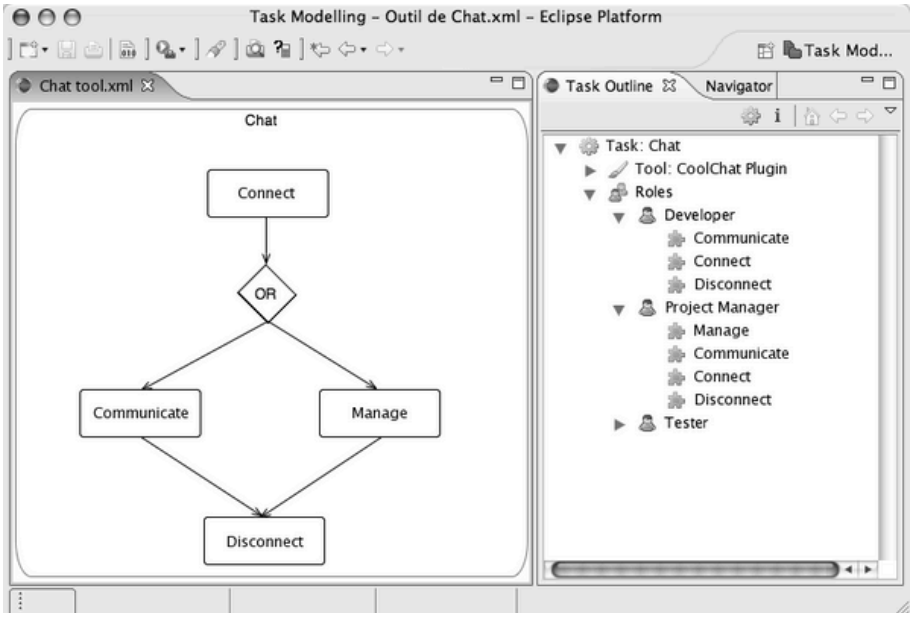

Figure 18.5. *An example of access to (re)definition: the sub-activity of Chat.*
(explanation of actions (as tasks) available for the roles)

An example of customization in *CoolDev* is the possibility offered to subjects, according to their role, to choose a perspective (as the arrangement shown in Figure 18.1) on their activity. *CoolDev* being built as an extension of the Eclipse platform, the concept of perspective corresponds to that of the same name in Eclipse: it is an arrangement of tools such as an editor, a chat or other. The integration technique allows subjects to introduce in the activity support new tools in the environment.

In *CoolDev*, these tools correspond to any Eclipse plugin2 [IBM 06] available on the Internet. Finally, the extension mechanism of *CoolDev* allows for example (re)specifying the rights of roles of actors on tools involved in this activity, this through the metamodel and within the framework.

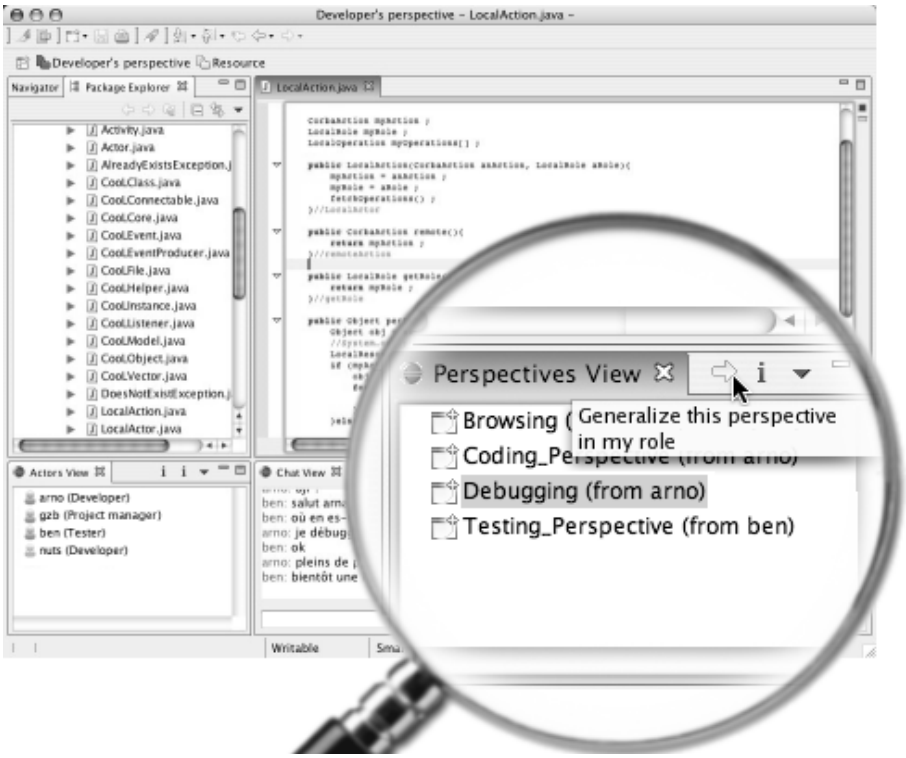

Figure 18.6. *Experience crystallisation of by generalisation of perspective in the models*

All these transformations of the environment that reflect the reflexive attitude of subjects are carried out according to their experience. However, every transformation is not necessarily the reflection of an experience suited to crystallise within the tool. Indeed, subjects may need to carry out tests before wanting to share their experience.

For example, in *CoolDev*, each actor has a role, instance of a shared model. We have implemented prototyping mechanisms allowing instances of roles to detach

[2] Integral component extending the functionalities of the platform.

from their model in order to allow particular experiments, such as the choice of arrangement of tools, developed during the activity. This mechanism corresponds to a controlled release of the causal relation between the role model and its instances. Thus, in our example, each developer has the opportunity to integrate and arrange new tools in his environment. Hence, all the developers do not have exactly the same perspective, even if they share the same role. However, each instance of role functions as a prototype that represents the preferences but also the experience of each actor in his role. Finally, when a perspective or an instance of role represents an interesting experience, this instance can be crystallised, that is generalised at the level of the role model (Figure 18.6). Thus, the roles of all the actors that follow this model can be influenced and the future actors who will be assigned this role will directly benefit from the crystallized experience.

Finally, we have identified several computer techniques that indeed allow crystallizing and sharing the experience developed by subjects. When compared to the techniques that we just mentioned, the *components*[3] approaches and techniques related to *model-driven engineering* allow the sharing of experience. Indeed, the very purpose of a component is its reuse by others. However, any component contains experience that was recorded by its developers. In a similar approach, each model can be seen as a component intended to specialize a generic structure. Thus in *CoolDev,* a particular scenario of software development activity, transformed and developed during its implementation, is a model that has crystallised the experience developed by those who have used it and caused it to evolve. This model can then be reinstantiated in the platform in order to support a different (involving for example other actors), but similar (starting with the same definitions of roles, the same tools) activity.

18.4. Discussion: towards new developments

Faced with the problem of the reuse of results from ad-hoc confrontations between HSS and computer science, we have proposed a first comparison between the generic properties of human activity described in AT and the techniques of computer science that allow supporting them. It can be noted that these computer techniques have, at least in some case, already been implemented in particular situations. Introspection, intercession and approach by models have already been used to alleviate problems of rigidity of workflows [KAM 00] and intercession techniques have been used to address the problem of malleability [MOR 97]. This is not surprising since, as we have pointed out, computer science and HSS fed on their respective experiences.

---

[3] Computer components are software elements defined in order to facilitate their reuse since they offer a particular interface hiding the complexity of their implementation.

However, the fact of addressing these comparisons in a generic way allows approaching tools development in a more theoretical way by directly identifying the generic properties of human activity that we want to support and techniques that have emerged from computer science and that allow to do so. Moreover, this approach could allow revealing defects in each discipline. For example, in computer science, components approaches exist for many years. However, even if technical solutions exist for their implementation, they are much less well defined from a semantic point of view.

Considering a component in terms of human activity highlights the fact that it is today very difficult to understand what will be its place in the activity that wants to integrate it. Only experienced and motivated computer scientists are able to really reuse most computer components. However, as we saw previously, this technique proves to be very beneficial for the reuse of experience *in situ* and by end users. This problem of semantics is due to the fact that with current techniques, it is necessary to almost completely rebuild the task model of a component to be able to contextualize it in a particular activity.

Indeed, its task is diluted in the code and the documentation means generally provided, as the Javadoc, do not really allow understanding its underlying overall logic. This is why, in order to facilitate the dynamic integration of components, we are working today on a new approach inspired by works on AT and task modelling from the field of HCI [LEW 06]. This example demonstrates how the fact of considering computer science from the point of view of HSS can lead to transformations in its techniques. Conversely, the study of computer science, crystallizing the experience developed by computer scientists, can certainly also feed the HSS. It is still too early for us to really measure the impact of this approach. However, we hope that it will lead in the future to better exchange between these disciplines.

18.5. Bibliography

[BAR 02] BARTHELMESS P., ANDERSON K.M., " A view of software development environments based on activity theory ", *Computer Supported Coop. Work 11(1-2),* p.13-37, 2002.

[BED 97] BEDNY G., MEISTER D., *The Russian theory of activity, Current Applications to Design and Learning*, Lawrence Erlbaum Associates, Mahwah, 1997.

[BOU 05] BOURGUIN G., DERYCKE A., " Systèmes Interactifs en Co-évolution, Réflexions sur les apports de la Théorie de l'Activité au support des Pratiques Collectives Distribuées ", *Human Computer Interaction Review(HCIR),* vol.6-1, AFIHM Europia, p. 1-31, June 2005.

[DOU 98] DOURISH P., BUTTON G., " On "Technomethodology": foundational relationships between ethnomethodology and system design ", *Human-Computer Interaction*, vol. 13, Lawrence Erlbaum Associates, p. 395- 432, 1998.

[ENG 87] ENGESTRÖM Y., *Learning by expanding*, Orientakonsultit, Helsinki, 1987.

[HAL 02] HALVERSON C., " Activity Theory and Distributed Cognition: Or What Does CSCW Need to do with Theories? ", *Computer Supported Cooperative Work (CSCW),* vol. 11, n° 1-2, p. 243-267, 2002.

[IBM 06] IBM Corp., Eclipse Platform Technical Overview, http://www.eclipse.org/articles/, 2006.

[KAM 00] KAMMER P.J., BOLCER G.A., TAYLOR R.N., HITOMI A.S., BERGMAN M., " Techniques for supporting Dynamic and Adaptive Workflows ", *Computer Supported Cooperative Work,* 9(3-4), p. 269-292, 2000.

[KAP 99] KAPTELININ V., NARDI B., MACAULAY C., " The Activity Checklist: a Tool for Representing the "Space" of Context ", *Journal of Interactions*, p. 27-39, July/August 1999.

[KOR 02] KORPELA M., MURSU A., SORIYAN H.A., " Information Systems Development as an Activity ", *Computer Supported Cooperative Work,* 11(1-2), p.111-128, 2002.

[KUT 91] KUUTTI K., " The concept of activity as a basic unit of analysis for CSCW research ", *Proceeding of the second ECSCW'91 conference*, Kluwers Academics Publishers, p. 249-264, 1991.

[LEW 05] LEWANDOWSKI A., BOURGUIN G., " Inter-activities management for supporting cooperative software development ", *Proceedings of the Fourteenth International Conference on Information Systems Development (ISD'2005)*, Karlstad, Sweden, 15-17 August 2005.

[LEW 06] LEWANDOWSKI A., BOURGUIN G., TARBY J-C., " Towards Task Oriented Software Components ", *Proceedings of the 18th French-speaking conference on Human Computer Interaction (HCI'06)*, Montreal, Canada, 18-21 April 2006.

[MAE 87] MAES P., Computational Reflection, Thesis, V.U.B, Brussels, 1987.

[MOR 97] MORCH A., Method and Tools for Tailoring of Object-oriented Applications : An Evolving Artifacts Approach, part 1, Thesis, 241, University of Oslo, 1997.

[NAR 96] NARDI B.A., *Context and Consciousness: Activity Theory and Human-Computer Interaction*, MIT Press, Cambridge, MA, 1996.

[SOU 03] DE SOUZA C.R.B., REDMILES D.F., " Opportunities for Extending Activity Theory for Studying Collaborative Software Development ", *Workshop on Applying Activity Theory to CSCW Research and Practice, in conjunction with ECSCW 2003*, Helsinki, 2003.

[SRI 04] SRIDHARAN P., Visual Studio 2005 Team System: Overview, http:// msdn.microsoft. com/library/default.asp?url=/library/en-us/dnvsent/html/vsts-over.asp.