TASKS MODELS FOR COMPONENT CONTEXTUALIZATION

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

Jean-Claude Tarby
Laboratoire Trigone - Institut CUEEP, Université Lille 1, 59655 Villeneuve d'Ascq, France

Keywords: Component, Integration, Tasks Models.

Abstract: This paper proposes a solution to take into account the emerging and evolving nature of users' needs in software environments. This solution consists in giving the users the means to adapt these environments by integrating new tools. Many technical solutions exist for software components integration, but their use is limited to software development experts. One reason is that current dynamic integration approaches face a semantic problem: to finely integrate a tool in an activity, its future place in this activity must be clearly identified. In order to facilitate this comprehension and the dynamic integration of software components, we propose a new approach of component design and integration, inspired by previous work on task modelling.

1 INTRODUCTION

Today, many researchers concentrate their efforts to give an answer to the users' emerging needs in the software environments supporting their activities. The great advances made in the last ten years have largely contributed to make users more and more demanding towards computer systems. If in the past the user had no choice but adapt him/herself to a rigid and concurrence-free system, the trend now tends to reverse: if the user cannot adapt the system to his/her emerging needs, s/he will probably choose another one that better suits these needs. To give an answer to this strong matter, one of the proposed approaches consists in providing means to users to adapt software environments, following the thread of their needs. This involves introducing into these environments the adequate mechanisms that will allow users to integrate the tools they download on the Web. Some advanced mechanisms can help in realizing such an integration of computer tools from a technical point of view. However, these solutions still face a semantic problem: in order to finely integrate a tool (or software component) in the environment, one must understand 1) the functioning of the tool, and 2) what will be its place in the global activity supported by the environment. We think that the current models and tools present some shortcomings regarding this aspect, and remain inaccessible to users. In order to palliate this lack, we work on a new approach for the definition, development and dynamic integration of software components. We propose to better use the tasks models coming from the CHI research field. In classical design approaches, these tasks models tend to 'evaporate' during the development process, and finally completely disappear. We think that these models could improve the understanding and facilitate the fine integration of software components by users. In the first part of this paper, we explain the problematic tied to the fine and dynamic integration of components, bringing up existing solutions and the problems they face, especially regarding their accessibility to end users. Then we propose the Task Oriented (TO) design approach aimed at adding semantic in the components thanks to the use of tasks models, in order to assist their integration — or contextualization.

2 THE PROBLEM OF CONTEXTUALIZATION

Many theoretical and empirical studies have already demonstrated the emerging nature of users' needs

Lewandowski A., Bourguin G. and Tarby J. (2007).
TASKS MODELS FOR COMPONENT CONTEXTUALIZATION.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - HCI*, pages 176-181
DOI: 10.5220/0002387301760181

towards their activities and the environments supporting them (Cubranic et al., 2004, Kuutti, 1993, Suchman, 1987). Actually, many research works tend to integrate in the software environments the mechanisms suited to support these emerging needs, and to give the users the possibility to make these environments evolve by supporting what we call the *contextualization of new tools* (or components) *in the environment*.

2.1 A SHS-based Point of View

Since a long time, the CSCW (Computer Supported Cooperative Work) research field has integrated the results coming from researches in Social and Human Sciences demonstrating clearly that users' needs cannot be completely and exhaustively defined *a priori*. Thus, the Activity Theory (Bedny and Meister, 1997) brings to the fore the fact that each human activity – and the many elements composing it – continuously evolves during its realization. This is also true for the users' needs that emerge during and from the activity. Therefore, a computer system intended to support a particular activity must also be able to evolve. Following this trend, we try to provide adequate software environments according to the *coevolution* principle — defined in details in (Bourguin et al., 2001) — by allowing end users to dynamically and cooperatively (re)define their software environment. An example of such a (re)definition can be a change in the roles played in the cooperative activity, or the dynamic integration of new tools (as components) into the working environment. Since our work is strongly inspired by the Activity Theory, we have defined this dynamic integration need in terms of *inter-activities* management (Lewandowski and Bourguin, 2005). This approach considers that each tool is intended to support a particular kind of activity. But when several tools are used in parallel by a group of actors, they generally serve a more global activity than the one they were created for. For example, let us imagine a development team that uses in parallel a synchronous discussion tool, a code editor and a file-sharing tool such as CVS. Each of these tools supports a particular activity (synchronous discussion for the chat, etc.) but they do not know each other. However, they are used in a complementary way by the group since they contribute to the realization of a particular global cooperative activity of software development. The coherence of the environment is generally mentally managed by the users themselves. Our objective is to provide an environment that creates the context of use of the many tools implied in a global activity and that supports for example a software development activity by managing the links existing between these various (sub-)activities, i.e. by managing the inter-activities. Managing the inter-activities mainly consists in managing the context of execution of the tools use brought into play by users in the realization of their global activity (Lewandowski and Bourguin, 2005). For instance, a particular context may configure the tools in order to reflect the user's role in the global activity supported by the environment. Thus, the actors in charge of the tests during a software production process will not have the rights, for example, allowing them to modify the source code of the product. The same context will also be able to pilot the tools according to the state changes in the global activity. The implementation of such scenarios in the inter-activities, that orchestrates a set of tools that do not know each other, supposes that these tools or components can be finely contextualized or integrated in the global environment.

2.2 Technical Means

The software components integration problem is a large and complex area of research and many technical solutions try to solve it. For example, distributed components such as CORBA components (Wang et al., 2001), EJB (Enterprise JavaBeans) (Blevins, 2001), or the Web Services (Ferris and Farrel, 2003) have been conceived with their future integration in mind. Some of them are associated with composition languages (Van der Aalst, 2003) that ease the fine integration of these components or services inside software applications. One can notice that such technical solutions are exclusively meant for software development experts, especially because of their complexity, of their implementation cost and of the specificity of the used techniques (Heineman and Councill, 2001). However, these different methods follow the same principle: it is possible to dynamically discover objects on the Internet, to instantiate them, to introspect their public methods and eventually their event channels, and finally to use them. These mechanisms can be very useful to manage the Inter-activities. Nevertheless, they mainly bring a solution to the technical dimension of the problem. Integrating finely and dynamically a tool supposes not only that we are able to use it, but also that we understand *how* to use it. In order to overcome this semantic problem inside components, Object Oriented (OO) methods provide some supports for

their understanding, such as the WSDL (Booth and Liu, 2006), a Web Services Description Language, or the Javadoc, a documentation of JavaBeans components generated from specific tags inserted in the source code of these components. Thus, in the example of the creation of a chat component, the associated Javadoc could only describe the set of public methods that will be used for its integration and piloting, such as `sendMessage`, `authenticate`, `showGroup`, etc. However, in order to integrate such a component, this kind of documentation that describes 'what the methods do' but not 'how to use them' is generally not sufficient. Every developer has already encountered this problem. For example, in which order these methods must be invoked to contextualize properly the chat component in the environment? A solution consists in looking for some existing examples of such integration, or in dissecting the source code of another application that uses the same component, or in the best case, in using a tutorial that will initiate us in its use.

We think that these difficulties tied to the dynamic integration of third party components in software environments are due to a semantic loss in the available means as for their documentation. Only passionate computer scientists are generally capable of properly integrating most of the components coming from the Internet because, by studying existing source codes, they have to mentally reconstruct almost completely the functioning mechanics of the tool they want to integrate. This problematic restricts reutilization to very specialized users. Moreover, even if many works in progress (Kiniry, 2005) try to palliate this semantic gap inside component models, the proposed solutions generally still remain intended to skilled developers. In the frame of our work on the *coevolution* principle (Bourguin et al., 2001), it has been demonstrated that it would be strongly valuable to facilitate this fine and dynamic integration, especially for and by users. These users, not necessarily computer scientists, are not experts in the technology used, but rather in the task they want to achieve.

2.3 What is a Tool from a Task-oriented Point of View?

In harmony with our inter-activities approach, we can consider that each tool supports the task it has been designed for. In other words, every software component can be seen as a generic support for a particular task that is more or less implicitly inscribed inside the tool. Indeed, the designer of the tool has created the underlying mechanisms and its interface in order to propose an adequate support for a given task. Thus, a *mailing* component supports the realization of *mailing* tasks; a *chat* component supports *synchronous discussion* activities, and so on. So we can consider that contextualizing a tool is nothing else than contextualizing an existing task into the framework of a more global task. In order to facilitate this contextualization and to bring an answer to the dynamic integration problems, we propose to better use the component's tasks model, a kind of missing link that generally disappears between the design stage and the delivered code.

3 FROM OBJECT-ORIENTED TO TASK-ORIENTED

3.1 Tasks Models during the Development Process

Software engineering provides more and more scope for tasks usage, especially in the CHI domain. Many works in this field are oriented towards the means of expressing users' tasks. This task-oriented approach generally occurs prior to or after the production process (Clerckx et al., 2004, Delotte et al., 2004, Lu et al., 2003, Luyten et al., 2003, Luyten et al., 2005, Paris et al., 2005, Paterno, 2004, Reichart et al., 2004). However, one can notice that if these methods propose to start the component design stage with a task-modelling step, this approach progressively fades during the process and finally disappears behind an object-oriented design approach inspired by the computer engineering background. This classical design approach tends to transform tasks models into objects models, from which emerges implicitly the class-based structure of the produced component (cf. top part of Figure 1). The original tasks model is swamped, implicitly inscribed in the complexity of the produced source code. In addition, task-oriented approaches are slightly used – or even not used at all – during the design and development cycle, namely after the requirements collection and analysis. Besides, most of the current well-known design tools widely used by the software engineering domain – such as Integrated Development Environments (IDE) intended to support the programming/test stages, or CASE tools (Computer-Aided Software Engineering) intended to support the whole production process – do not integrate at all the task-oriented approaches. UML itself does not integrate

the notion of tasks as it is known in the CHI research domain. For example, UML does not take into account the notions of users' goals or users' intentions; it does not permit the modelling of users (knowledge, habits, etc.); the difference between interactive task, system task, and hand operated task does not appear in UML. Nevertheless, work has been done in order to palliate this shortcoming (Bruins, 1998, Nunes et al., 2000, Pinheiro da Silva, 2002, Scogings and Phillips, 2004), taking advantage of similarities in the concepts between UML and task-oriented approach in order to create some kinds of translators.

Therefore, one can notice that the benefit acquired upstream during the tasks modelling phase totally disappears during the design and implementation stages. Even if the produced source code reflects the originally identified tasks, it is very hard to recover the tasks model from the source code. This fact is closely akin to the idea previously mentioned concerning the need for integrators to go into the code of a component in order to extract its functioning logic from a computer science viewpoint, and also its usage logic from an end user point of view, which is nothing else than reconstructing and making explicit again the underlying tasks model. This is the same reason that leads development teams, during final tests, to make use of "spywares", interviews, and other methods in order to recover parts of this tasks model. Consequently, we are convinced that keeping explicit the tasks model inside the final software component will be useful for the dynamic integration of tools in the users' activities. Furthermore, tasks models also often serve, as shared objects, to help a better communication between the diverse actors (including the future users) implied in the complex software development process. Therefore, putting the tasks model to the users' disposal should not only help in the understanding of the functioning of the tool, but also facilitate – by providing the tool usage logic – its integration in the global task by the end users themselves. This is what we propose to realize thanks to a new design approach.

3.2 A New Design Approach

As previously stated, there is a real benefit to use tasks models in every step of the software component's life cycle, from its design to its development and even in its integration. Our proposition consists in a Task-Oriented (TO) design approach, which is nothing else than a 'classical' design approach extended with the explicit keeping of the links between the source code and the tasks model it is based on. This approach does not require a particular formalism for the tasks modelling, since we focus on linking tasks models concepts (that are independent from the formalism used) with source code. In the same way, this design approach does not modify the class-based structure of the component. Indeed, our approach precisely consists in identifying the links between the code (in other words the structure of the component) and the original tasks model. These links will be written in the source code as tags during the implementation stage — technical aspects about how these tags are written will not be explained in this paper due to the lack of space, but these tags are similar with the Javadoc tags that are inserted into the code as comments. This approach leads to provide software components with their tasks model inside and linked to the code (cf. bottom part of Figure 1).

The benefit of this method stands in the fact that it is a very low cost approach that induces only a small extra work for designers and developers. Indeed, supposing that the tasks model has already been realized during the requirements analysis stage, the extra work will only consist in writing into the code the links mentioned above, and in delivering the final component with its tasks model inside. On the other hand, the gain in terms of semantic regarding the functioning of the tool is noticeable and will help in its understanding. As for the contextualization, we propose to provide handling means that will associate introspection and classical documentation mechanisms with the component's tasks model.

3.3 Perspectives for Contextualizing to Components

From a TO component, we can apply the classical introspection mechanisms to retrieve the public methods we can use for its integration, the Javadoc that describes the functioning of these methods in an Object-Oriented way, but also the direct links between these methods and the underlying tasks model. In order to facilitate the contextualization of this component in the global activity, we currently develop a prototype that will allow this introspection from the tasks model point of view, by navigating between this model's different hierarchical levels, and by linking it with the global activity's model. The prototype provides a global view of the tasks model related to the component we want to integrate, and in this way it will help in its

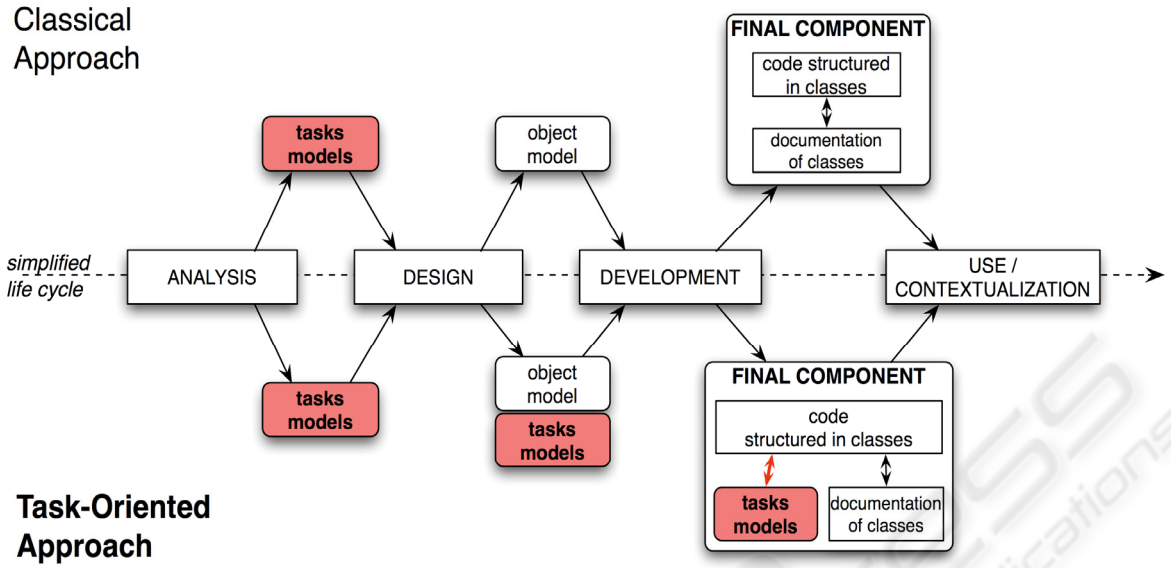

Figure 1: Schematization of the classical design approach, and of the Task-Oriented approach that tends to preserve the tasks model all along the development process, for a better contextualization of the produced component.

understanding. Furthermore, it will give the means to specify how this component will be integrated in the global task, through the definition of links between the component's tasks model and the global activity's model.

We now work on the validation of the TO approach and expect that it will give the following benefits. First, as we mentioned before, this approach should induce only a small amount of extra work for designers and developers, provided that the previous requirements analysis stage has produced a usable tasks model. On the other hand, the addition of a tasks model describing the component's functioning and usage to other documentation means (Javadoc, etc.) should make its understanding more intuitive. Its contextualization in the global environment by the creation of fine links will thus become easier. We also pursue our efforts in order to make these features (the introspection mechanisms on the tasks models and the tools manipulating them and assisting components contextualization) more accessible to users.

4 CONCLUSION

One of the solutions considered in order to answer to the emerging nature of users' needs consists in proposing the dynamic integration of new tools in their software environment. From a human activity point of view, this dynamic integration raises a semantic problem. Indeed, the fine integration of a component implies the understanding of the task it is intended to support, in order to define the place it will hold in the global activity. Unfortunately, the current means induce a semantic loss during the development process of components. The Task-Oriented (TO) approach we propose takes benefit from the tasks modelling process that occurs during the early requirements analysis stage of the development process. Generally, these tasks models are finally diluted in the source code of the delivered component. Our approach tends to preserve these models all along the development process, and to pack them with the final component, keeping the fine links existing between the original tasks model and the corresponding source code that realizes it written in the code. We have verified the feasibility of such an approach by developing a chat component according to the TO approach. We are going to improve and complete these new introspection means and their tooling in order to facilitate the dynamic integration of tools or components by end users, to reach a better co-evolution (Bourguin et al., 2001).

ACKNOWLEDGEMENTS

We thank the organisms supporting this work, in particular the French Research ministry for the ACI Jeunes Chercheurs CooLDev, the TAC (Advanced

Technologies for Communication) program financed by the Région Nord/Pas-de-Calais and by the French State in the framework of the NIPO/MIAOU and the EUCUE projects, and the FEDER.

REFERENCES

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

Blevins, D. Overview of the Enterprise JavaBeans Component Model. In (Heineman and Councill, 2001), pp. 589-606.

Booth, D. and Liu, C.K. Web Services Description Language (WSDL) Version 2.0 6 January 2006.

Bourguin, G., Derycke, A., Tarby, J.C. Beyond the Interface: Co-evolution Inside Interactive Systems – A proposal Founded on Activity Theory. Springer Verlag, Vanderdonckt, Gray (eds), 2001, pp. 297-310.

Bruins, A. The Value of Task Analysis in Interaction Design. In Task to Dialogue: Task-Based User Interface Design, Workshop, CHI'98, Los Angeles, April 18-23, 1998.

Clerckx, T., Luyten, K., Coninx, K. DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In The 9th IFIP Working Conference on Engineering for Human-Computer Interaction, jointly with the 11th International Workshop on Design, Specification and Verification of Interactive Systems, Trems-büttel Castle, Hamburg, Germany, July 11-13, 2004, Pre-Proceedings, pp. 142-160, 2004.

Cubranic, D., Murphy, G.C., Singer, J., Booth, K.S. Learning from project history: a case study for software development. In Proceedings of CSCW04, Chicago, Illinois, USA, ACM Press, 2004, pp. 82-91.

Delotte, O., David, B.T., Chalon R. Task Modelling for Capillary Collaborative Systems based on Scenarios. In (Slavík and Palanque, 2004), pp. 25-31.

Diaper, D. and Stanton, N. Handbook of Task Analysis for Human-Computer Interaction. Lawrence Erlbaum Associates Pubs, London, 2004.

Ferris, C. and Farrel, J. What are web services? Communications of the ACM, 46(6), 2003, pp. 31.

Heineman, G.T. and Councill, W.T. Component-based software engineering: putting the pieces together. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2001.

Kiniry, J.R. Semantic Component Composition. Third International Workshop on Composition Languages, in conjunction with 17th European Conference on Object-Oriented Programming (ECOOP), Darmstadt, Germany, 2003.

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

Lewandowski, A. and Bourguin, G. Inter-activities management for supporting cooperative software development, in Proc. of the 14th Int. Conf. on Information Systems Development (ISD'2005), Karlstad, Sweden. Springer Verlag, 12p.

Lu, S., Paris, C., Vander Linden, K. and Colineau, N. Generating UML Diagrams From Task Models. In Proc. of CHINZ'03, July 3-4 2003, Dunedin, New Zealand.

Luyten, K., Clerckx, T., Coninx, K., Vanderdonckt, J. Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In Interactive Systems: Design, Specification, and Verification, 10th International Workshop DSV-IS, Funchal, Madeira Island, Portugal, June, 2003.

Luyten, K., Vandervelpen, C., Coninx, K. Task Modeling for Ambient Intelligent Environments: Design Support for Situated Task Executions. 4th Int. Workshop on TAsk MOdels and DIAgrams for user interface design (TAMODIA 2005), Gdansk, Poland, 2005, pp. 87-94.

Nunes, N. Falcão e Cunha, J. Towards a UML profile for interaction design: the Wisdom approach. In Proc. of the UML 2000 Workshop, 2000.

O'Neill, E., Johnson, P. Participatory task modelling: users and developers modelling users' tasks and domains. In (Slavík and Palanque, 2004), pp. 67-74.

Paris, C., Colineau, N., Lu, S., and Vander Linden, K. Automatically generating effective online help. International Journal on Elearning, ISSN 1537-2456, Volume 4, Issue 1, 2005, pp. 83-103.

Paterno, F. ConcurTaskTrees: An Engineered Nota-tion for Task Models. In (Diaper and Stanton, 2004), pp. 483-501.

Pinheiro da Silva, P. Object Modelling of Interactive Systems: The UMLi Approach. PhD's thesis, University of Manchester, United Kingdom, 2002.

Reichart, D., Forbrig, P., Dittmar, A. Task models as basis for requirements engineering and software execution. In (Slavík and Palanque, 2004), pp. 51-58.

Richard, J-F. Logique de fonctionnement et logique d'utilisation. Rapport de recherche INRIA N° 202, Avril 1983.

Scogings, C., Phillips, C. Linking Task and Dialogue Modeling: Toward an Integrated Software Engineering Method. In (Diaper and Stanton, 2004), pp. 551-566.

Slavík, P., Palanque, P. (eds.). Proc. of the 3rd Int. Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2004. November 15 - 16, 2004, Prague, Czech Republic, ACM 2004.

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

Van der Aalst, W. Don't go with the flow: Web services composition standards exposed. Trends Controversies Jan/Feb 2003 issue of IEEE Intelli-gent Systems, 2003.

Wang, N., Schmidt, D.C., O'Ryan, C. Overview of the CORBA Component Model. In (Heineman and Councill, 2001), pp. 557-572.