Towards Method Engineering of Model-Driven User Interface Development

Kênia Sousa, Hildeberto Mendonça, and Jean Vanderdonckt

Université catholique de Louvain, IAG-Louvain School of Management, Information Systems Unit (ISYS)
Place de Doyens 1, B-1348 Louvain-La-Neuve (Belgium)
{sousa, mendonca, vanderdonckt}@isys.ucl.ac.be

Abstract. Model-driven user interface development environments and their associated methodologies have evolved over time to become more explicit, flexible, and reusable but they still lack to reach a level that allows tailoring a method to the reality of software development organizations and their projects. In order to address this shortcoming, method engineering provides strategies to define and tailor software engineering methods. They should address any usability concerns, which are primordial for the integration of model-driven user interface development methods in the competitive reality of software organizations. To address the issues of explicitly defining a flexible method, we defined a strategy based on method engineering for model-driven user interface development that uses usability goals as a starting point. With the application of this strategy, we aim to help method engineers executing the method with more efficiency when defining or tailoring methods and facilitate the application of model-based user interface development methods in software organizations.

Keywords: model-driven user interface development, methodologies, method engineering, business process modeling, usability.

1 Introduction

Any development method or methodology, whether it is generic or specific for User Interface (UI) for instance, is usually decomposed into three related axes:

1. *Models* that capture different facets of the future interactive application.
2. An *Approach* which governs the actions conducted on the various models.
3. *Software* that supports executing the approach based on the models.

On the one hand, substantive efforts have been devoted to the definition and the usage of models, and extensive development of support software has been achieved. On the other hand, the approach aspect has received less attention over the past decades. Even though, there are many User Interface Development (UID) methods that use task models as a starting point to elicit user requirements and more precisely understand user cognition in order to make UIs more usable. Such a growing interest for models is due to the need to provide a more systematic approach to UID.

Professionals working in systems development usually follow a defined software development process, and when it comes to UID, many professionals do their activities more empirically because there is still resistance to the application of usability methodologies in software organizations [26], such as resource constraints and lack of knowledge about usability are the factors that most influence professionals. But, a formal UID method requires efficiency to be integrated into software development organizations. Model-based UID comes as a solution to improve efficiency by reusing models, reducing development efforts, among other benefits [3].

To make model-based UID methods applicable in the competitive reality of software development organizations, they need to be explicitly defined with the possibility of easy adaptation when it is necessary to consider constraints pertaining to specific projects [27,33]. Software organizations and their projects have specific characteristics, which require methods to be tailored, for instance, the skills and quantity of professionals affect how the method could be applied. UID is a creative process in which professionals feel the need for flexibility in their work to address the growing complexity of interactive systems. Therefore, a rigid method is no longer desired and there is a need to support method definition and adaptation. In the reality of software organizations and the need for tailoring the method for specific projects, the possibility to reuse pre-defined method specifications aids in accomplishing efficiency.

Considering this scenario, our main research question is: *How can method engineers define a model-based (or model-driven) UID method appropriate for the reality of the software organization and its projects?*

This research work aims to contribute in supporting the application of model-based UID methods efficiently by providing flexibility in its definition. Considering that the existing methods are diffused and applied in different projects around the world, such knowledge and experience acquired can not be taken for granted. Therefore, it is not the intention of this work to define a method nor to compare existing methods because we consider that a more appropriate method is adapted to the problem domain or context of the project, which has been investigated since the early 90's [17,27].

Concerning a possible automation for this support, it is important to address issues related to the creation and maintenance of a method base with propagation of changes in method specifications; how the model editors are integrated with the method tool; collaboration between professionals in the creation of models; the automatic or semi-automatic generation of UIs; coordination of the use of tools; change management of models; and support coordination of cooperative work. Solutions for these issues are appropriately addressed by technology for process automation, which allows executing methods. But such technology requires explaining many details that are not the focus of this work, but subject for another ongoing work.

This paper compares some existing solutions for the definition of methods and points out some shortcomings when considering model-based UID. In the upcoming sections, it proposes an approach for defining a model-based UID method by analyzing goals and activities, and it concludes by presenting the expected advantages and future work.

2 Related Work

A survey performed on Model-Based User Interface Development Environments (MB-UIDE) [16] showed that most of them provide a methodology for UI generation. These environments however support the execution of the methodology by automating some steps to generate a running UI or a specification of the UI; and even though some favor concurrent work or different sequence possibilities, they do not allow adapting the methodology according to the context of the project.

There are many MB-UIDEs that follow a formalized method [6,28,32], but their supporting tools do not provide facilities to change the sequence of the method activities, thus restricting the possibilities to adapt the method. Fig. 1 depicts the level of method flexibility of MB-UIDEs over time: oldest systems in the 90s had no method at all, except perhaps the one induced by the software; old systems like TRIDENT [5] has a very limited method flexibility since the method is completely coupled to the software and no tailoring is possible; TEALLACH [16] offers some flexibility since the design can start from one of the task, domain, and presentation models and evolve to the other models depending on the project; Cameleon-compliant software [10] are much more numerous today ([14,17,22,28,30] among others) and provide some adaptation of the method they rely on.

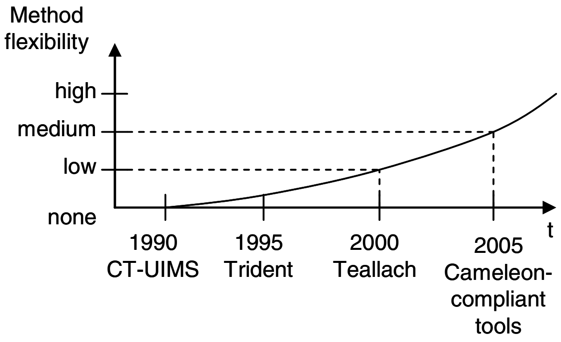

Fig. 1. The evolution of MB-UIDEs

The TEALLACH design process [16] aims to support the flexibility for the designer lacking in existing environments by providing a variety of routes in the process; from one entry point, the designer/developer can select any model to design independently or associate with other models. Even though this is a flexible approach to design UIs, it still hinders a complete flexibility because it is restrictive to the sequence of manipulation of models. Its flexibility is not extended enough to address the entire set of activities, roles, tools, and artifacts. For example, if a software organization aims at applying a method with such characteristic, it is limited by a set of models and activities implemented in the environment. Following, we present an overview of the assessment of model-based methodologies considering three main criteria:

Explicitness. Most methodologies have some kind of method definition, but not all aspects are explicitly defined, such as the association of roles, activities, models, and tools. For instance, some define the lifecycle as a sequence of transformation between models [32], some associate activities with the creation of models, but there is no association with the role responsible for executing them [6], while others have the methodology implemented in the environment, but not explicitly defined. Most of them do not mention tool in the lifecycle because their proposal is an environment to support the lifecycle.

Flexibility. The methodologies that are part of a MB-UIDE are not flexible enough [16], but TEALLACH comes as a solution to fulfill this need. Even though it provides a flexible approach, in the point of view of software development organizations, flexibility has a broader sense, which advocates the ability to change any aspect of the method and integrate with any existing process and tool.

Reuse. Some methodologies in MB-UIDE have a set of activities to be performed, within them, there is usually a set of activities that are not mandatory and can be executed or not, depending on the project's need. But, the idea of reuse is to offer a larger set of activities that provide a wider range of possibilities in different types of projects that could be selected for the method as necessary. This type of strategy is not common in MB-UIDE since the methodology is composed of a small set of activities targeted at a specific goal, such as in the use of patterns [28].

For application in real projects, existing approaches and their environments require organizations to start from scratch to apply the methodology available in the environment. To enhance the effect of methods, we need to adapt existing methods or create a new one that fits to the characteristics of each new project [27].

In a response to this demand, the term method engineering has been introduced as the "engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems." [7,8]

As an effort to address demands of flexible methods, there are several proposals to automate method engineering, as one of them, *Computer Aided Method Engineering* (CAME) supports building project-specific methods [27]. CAME has two types of tools; the first one is a method editor that creates a method and the second one is a generator of model editors based on the method meta-model to support the created method. This approach to generate CASE tools based on the method description decreases the possibilities of applying the newly created method with external tools, which are currently widely accepted for modeling software systems, as proposed in [17]. This work does not mention how this proposal applies in projects in which the software organization already has standardized a set of tools.

MetaEdit+ offers a CAME environment that allows method specification, integration, management, and maintenance [33]. It focuses on reuse and maintenance aspects for methodology specifications. It provides five strategies when requirements change may affect both the generated models and also the methodology. One detected drawback is that there is still no feature to support the reuse operation in building relationships between methodologies. We envision that during method specification it is primordial to allow integration with other methodologies because software organizations already applying a method may want to accommodate new techniques, in order not to start from scratch with a brand new method.

Decamerone [19] provides a way to adapt and integrate methods stored in a method base. Mentor [29] provides patterns for method engineers to easily design the method. An important aspect is that the generated methods and/or model editors are aimed for information system development, such as database systems, such editors do not address the complexity and creativity necessary in model-based UID.

After analyzing some approaches, the major weaknesses in these approaches is that MB-UIDEs focus on a specific and not so flexible methodology and CAME tools, even though they provide explicitness, flexibility and reuse, they only focus on system development, letting aside the concerns of usability, therefore not fully addressing the definition of model-based UID methods. MB-UIDEs do not allow the definition or adaptation of a method according to the characteristics of the organization and project, which makes them difficult to introduce certain activities that support model-based UID, such as version control. CAME tools are limited to software engineering models and method fragments and since they use a product meta model to generate model editors, they can profit from a meta model for UI models. Therefore, there is a need of interaction between MB-UIDEs and method engineering environments.

In this paper, our goal is to suggest a Model-Based User Interface Method Engineering that can address issues related to method engineering for model-based UID. We shall investigate model-based UID activities to be performed by designers and other usability team members to envision how usability goals specified by stakeholders in the beginning of the project affect the way the usability team works. In other words, we seek to demonstrate the relationship between model-based UID method activities and the desired usability goals and how this association helps outline a method that best suits the context of the project.

3 UID Activities

Considering the evolution of MB-UIDEs and their methodologies over time, it is noticeable the increase in flexibility, as presented in Fig. 1. The Cameleon Reference Framework [10] brings a solution that supports the realization of multiples types of development paths within a single framework. This framework structures a set of models that provide a support for the current user interaction challenges. This framework has 5 models distributed in 4 levels of abstractions in order to express the UID life cycle for different contexts of use. These levels of abstraction are aligned with the model-driven approach, which aims to reduce both the amount of developer effort and the complexity of the models used [18].

The language UsiXML [22] was created as a XML extension to describe UIs for multiple contexts of use, such as graphical, auditory and vocal user interfaces, virtual reality, and multimodal user interfaces. As a language explicitly based on the Cameleon Reference Framework, it adopts four development steps: 1) Task & Concepts, 2) Abstract User Interface (AUI), 3) Concrete User Interface (CUI), and 4) Final UI. The first step generates the task model, domain model and context model, the second step generates the AUI, and the third step generates the CUI. The language does not consider the Final UI as the framework does. The UsiXML methodology is structured as presented in Fig. 2 [30].

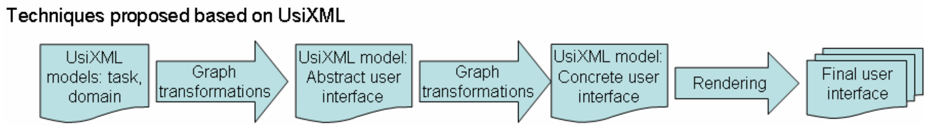
Techniques proposed based on UsiXML
UsiXML models: task, domain
Graph transformations
UsiXML model: Abstract user interface
Graph transformations
UsiXML model: Concrete user interface
Rendering
Final user interface

Fig. 2. The distribution of UsiXML models

The UsiXML language will be used to exemplify our proposal in the next sections since it provides the necessary support to represent models in a structured form and it supports the flexibility provided by the Cameleon Framework.

There is a suite of tools, automated techniques, and a framework to support the creation of models, and there is also a running effort to define a detailed model-based UID method. As follows, we explain how we intend to define such a method and how to integrate it with a software development process.

3.1 Theoretical Concepts

In this section, we describe the main theoretical concepts considered as the foundation of our proposal: model-based UID method engineering.

The proposed structure is based on the definition of method content from the Software Process Engineering Metamodel (SPEM), a meta-model for defining software development processes [25]. Considering that SPEM is "limited to the minimal elements necessary to define any software and systems development process, without adding specific features for particular development domains or disciplines" [25], we aim to add specific elements for UID. The main goal is to make usability as a central point not only for UI designers, but even before they come into action during software development processes; making usability also a concern for method engineers.

Fig. 3 depicts a class diagram with the most relevant elements for the definition of a model-based UID method. This proposal shall evolve progressively to address the organization of method activities in a process lifecycle nor does it consider the method enactment (or execution). This proposal extends the basic elements of a method engineering notation by associating usability goals with activities, which will be presented in the next sub-section. In general, a method is defined by describing Activities, which are selected for a Project based on Usability Goals. Activities are performed by Roles, and act upon Work Products using Tools to manage the work products, which can be UI Models.

Usability Goals should be established early in the project to drive professionals into focusing on UID efforts, and to use these goals as precise resources to evaluate their work towards accomplishing these goals. Usability goals can shorten the UID lifecycle, as stated in the Usability Engineering Lifecycle [23]. This methodology establishes usability goals in the requirements analysis phase and uses them to assess UIs during usability evaluation. In our work, usability goals have yet another purpose because they are used in the identification of activities that are appropriate for a specific project. The impact that usability goals can bring to method definition is to provide a manner to make method engineers (as well as project managers) more aligned with usability from the beginning until the end of the project, in order to make sure that all stakeholders value the importance to check whether or not such goals were accomplished in the end.

Projects are composed of activities that are performed to develop a system. *Activities* represent the work that is performed by roles when acting upon work products and using a tool. *Roles* define a set of competencies that professionals must have to execute such role by performing activities and being responsible for work products. *Work Products* are assets or artifacts that are used, produced or updated during the execution of activities using a tool. Work Products can be input or output of activities

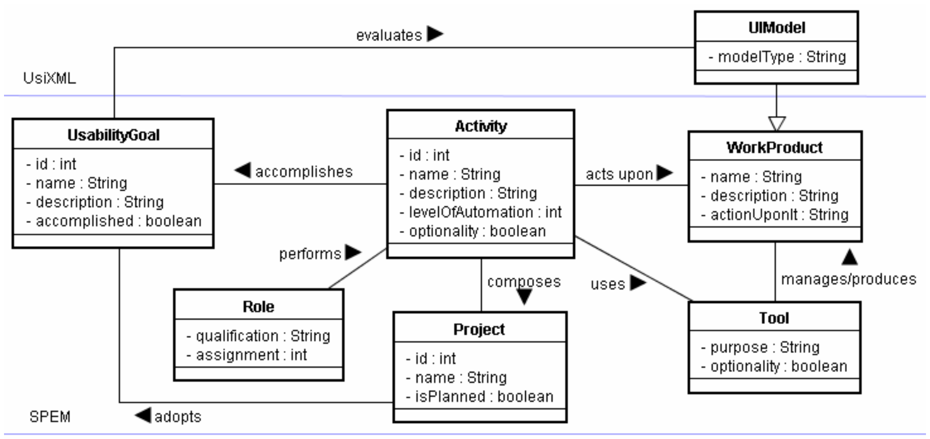

Fig. 3. Concepts for Model-Based UID Method Engineering

performed by roles. For a model-based UID method, the main work products are *UI models*. *Tools* support the execution of activities by managing work products, that is, a tool can manage one or more kinds of work products.

Activities can also be supported by other kinds of implementation besides tools, when it is necessary to implement functionalities that do not need tools or that can be available in more than one tool. In such cases and considering the current technology for process automation, we propose the use of web services.

In general, web services "allow access to a functionality via the web using a set of open standards that make the interaction independent of implementation aspects, such as the operating system platform and the programming language used" [12]. This technology promotes a high level of coherence and a low level of coupling, which contributes to assemble services to compose a method. Business Process Execution Language (BPEL) [4] was defined by W3C to promote assembling services. It has reached a good maturity and it is supported by the main architectures available in the market, such as JEE and .NET.

3.2 Strategy to Define a Method

Aiming at systematizing how a method can be defined and evolved, an evolution driven method engineering approach [2] was defined with two main goals: construct a product model and construct a process model. Focusing on the process model, this approach proposes four strategies to describe a process model:

i) *activity-based*, description of a set of actions to be carried out;
ii) *context-driven*, description considering the context, which is composed of the situation in which the product is undergoing transformation and the intention to be achieved in this situation;
iii) *pattern-driven*, use of a catalogue of patterns with the identification of generic problems and proposal of solutions applicable whenever the problem occurs;
iv) *strategy-driven*, integration of several process models into a complex multi-process model.

We selected the activity-based strategy to help method engineers in identifying activities to construct a method. We have adapted this strategy to the HCI domain, by proposing the identification of usability goals and their association with UID activities that can be included in the method to achieve the desired goals.

Depending on the usability goals presented early in the project specification and system requirements, a set of UID activities could be selected as part of the tailored method. Consequently, the activities performed by the professionals are aligned with the usability goals of the project with two main advantages. First, they are more effective in performing their work because each activity performed has a specific purpose. Second, if any non-planned goal is presented during the UID lifecycle, the method can be adapted with the selection of appropriate activities. A usability goal is a generic specification that can be addressed by one or more UID activities (see Table 1).

Table 1. Association of Goals and Activities

Usability Goal	UID Activity	Description
Design UIs considering users' mental models to perform their tasks	Create task model	Describe tasks in a hierarchical manner.
Design user-centered UIs	Create context of use model	Describe user's characteristics, platform used and environment.
Design UIs focused on the application domain	Create domain model	Describe the manipulated data.
Design for many devices	Create Abstract UI (AUI) model	Specify objects in a UI, independent of device.
Design focused on the look-and-feel of the system	Create Concrete UI (CUI) model	Specify positioning of objects in a UI, considering device constraints.
Adapt the user interaction according to users' personal characteristics	Create context of use model	Specify user's characteristics.
	Create task model	Specify user's tasks according to their specific characteristics.
Automate the generation of UIs considering many devices	Transform task and domain models into AUI model	Receive task model and domain model as input and generate AUI model.
	Transform AUI model into CUI model	Receive AUI model as input and generate CUI model.
Automate the generation of UIs for a specific device	Transform task and domain models into CUI model	Receive task model and domain model as input and generate CUI model.
Automate the generation of specification of UIs	Transform AUI into task model	Receive AUI as input and generate task model.

An activity can be associated with one or more usability goals, which is the case of the UID activity "Create task model". But, this does not mean that once the position and ordering of this activity has been defined, it has to be repeated twice for the different goals to be accomplished. On the other hand, it means that if a project needs to achieve both goals, the execution of this activity addresses both of them.

Depending on the usability goals, activities can be selected independently of each other, which is the case for the activities "Create task model" and "Create context of use model" with their own specific goal. But, in cases of a usability goal triggering more than one activity, their order of execution is clearly specified because one activity has a direct impact on the other, which is the case of executing the activity "Create context of use model" before the activity "Create task model" for the usability goal "Adapt the user interaction according to users' personal characteristics".

In cases when stakeholders state that they want some kind of automation in UID to achieve more productivity, certain activities can be selected depending on the goal. For instance, the activity "Transform task and domain models into AUI model" is appropriate when various devices are considered and the activity "Transform AUI model into CUI model" also aids in the productivity level of designers since they receive UIs with the necessary objects as a starting point to work on the look-and-feel. The activity "Transform task and domain models into CUI model" is useful when one specific device is the aim.

UID activities that are commonly used may already be included in software development processes, such as defining a style guide, prototyping, usability evaluation, among others. But, in cases where such activities are not yet part of the organizational software process, usability goals must be considered to correctly apply these activities. It is our intention to further improve the list in Table 1 with usability goals associated to such activities.

3.3 Tool Support

Tool support for method engineers can be very useful for their productivity when defining or customizing methods. The process of deciding which are the most appropriate activities for specific projects requires knowledge and experience, but tools can help them to maintain a base of experiences and learned lessons, when easily accessed can add value to their work. Therefore, in addition to the strategy presented in the previous section, we selected Business Process Modeling Notation (BPMN) as a standard with available tools to support method engineers.

BPMN was proposed to be applied in the representation of organizational processes [24], and we propose to use BPMN in method definition because: i) it has become a pattern for process modeling; ii) there are many tools available in the market implementing it; iii) it has been intended as a human-readable layer that hides the complexity of designing transactional business processes; and iv) BPMN can be transformed in BPEL to be automated using web services, as described at the end of section 3.1.

There are many tools available that implement BPMN, which provide the necessary support for method engineers that follow a common structure as in the tool presented in Fig. 4. But, after the assessment of model-based UID methods, we noticed the need to use method engineering techniques to improve method definition.

Therefore, we have analyzed the alignment of BPMN with a software engineering notation, more specifically with SPEM. The alignment and complementary aspect is confirmed by quoting the SPEM documentation [25]: "SPEM 2.0 does not aim to be a generic process modeling language, nor does it even provide its own behavior modeling concepts. SPEM 2.0 focuses on providing the additional information structures that you need for processes modeled with UML 2.0 Activities or BPMN/BPDM to describe an actual development process." Using a process modeling tool to define a method, we have followed three steps, as pointed out in Fig. 4:

1. *Definition of activities* – we have defined a list of activities for a model-based UID method based on the Cameleon Framework.
2. *Association of BPMN and SPEM* – we have associated BPMN elements with SPEM elements to give meaning and use business process elements in the method engineering domain.
3. *Reuse of activities* – drag and drop activities from the pre-defined list (on the left of the tool) and reuse them when defining the method for a specific project, in the desired or recommended order.

The method defined on the right side of the tool in Fig. 4 is clearly related with the concepts defined in Fig. 3. For example, the Role "Usability Expert" performs the Activity "Create AUI" and acts upon (by creating) the Work Product, which in this case is a UI Model "AUI Model" by using the Tool "IdealXML". To complete, this activity is present in this method because the stakeholders stated the Usability Goal "Design for many devices", which is directly associated with the activity "Create AUI".

After analyzing which activities are important to achieve certain usability goals and selecting the appropriate ones, it becomes easier to define a method. We must furthermore be able to define methods that are applicable in software development

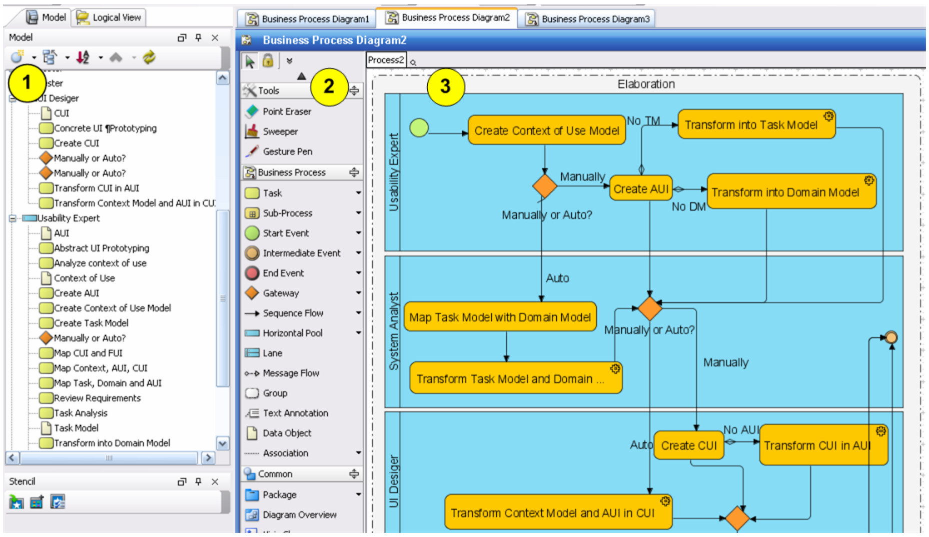

Fig. 4. Activity selection using a process modeling tool [31]

projects and also provide support for model-based UID. Following, we demonstrate an example of integration of model-based UID activities in a software development process.

4 Integration of Methods

In an attempt to make UID methods really effective in real projects, there have been various efforts to bridge the gap between software engineering and HCI. Some proposals focus on user involvement [15], on how to help software engineers execute usability techniques [13], on addressing usability issues using architectural patterns [20], others are product-oriented and adapt an object-oriented notation to support HCI techniques [11], but all aim at making usability techniques applicable in real-life software development projects.

The technique to define project-specific methods from parts of existing methods is called method assembly [8], which can produce a powerful new method. Using this technique, we integrate the best from both domains: activities from a world-wide accepted commercial software development process, the Rational Unified Process (RUP) [21]; and activities for creating UI models. Works, such as [9], demonstrate that the integration with RUP can make model-driven methods in general more accessible to a wider audience of software engineers.

While some HCI methods have specific and unique structures, like the Usability Engineering Lifecycle [23], many proposals that integrate SE and HCI are based on the RUP structure, such as the integration of development activities with usability techniques [13] is based on the RUP process structure; and the UCD [15] creates a new discipline for usability design in the RUP.

This is an example of the integration of a model-based UID method and a software development process. Picture a software organization that already has a well-deployed software development process, such as the RUP and wants to focus on UID. For

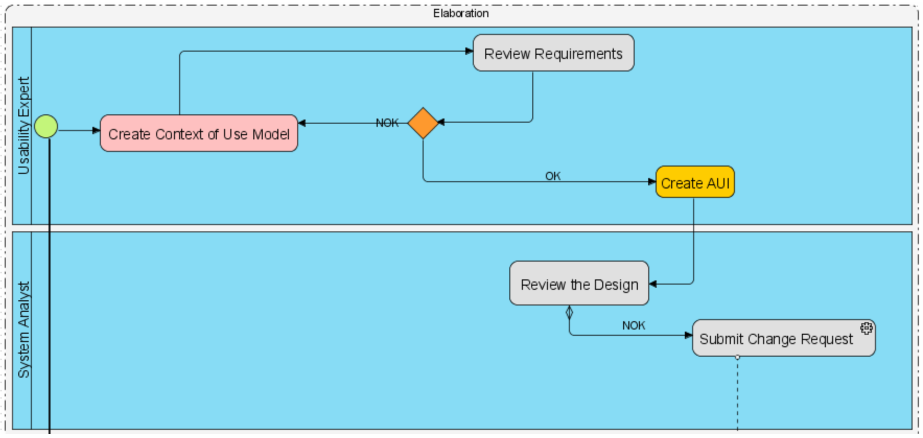

Fig. 5. Integration of software and UID activities

instance, when the organization already has a standard way to do tests, reviews, and controls of change requests, but it wants to increment its way of working with models, it is possible to make a smooth integration. In Fig. 5, we present activities related to model-based UID: create context of use model and create AUI, and SE activities: review requirements, review the design, and submit change request.

Our proposal to support the integration scenario is provided with the association of goals with activities that can be appropriately allocated in the method. For instance, if a new project aims at designing UIs for many devices, the activity "Create AUI" is included in the organizational software process to accomplish this usability goal, as specified in Table 1. In addition, the method engineer might also need support in defining the sequence of the activities; therefore, a proposed model-based UID method that integrates UID activities and RUP activities can be provided as a source of guidance, which is subject for future work.

5 Conclusion

The main goals we intend to achieve with our proposal of a model-based UID method engineering is to aid method engineers when creating methods more efficiently and also to make model-based UID methods applicable in the competitive reality of software development companies.

Method engineers can define a model-based UID method appropriate for the reality of the software organization and its projects using an activity-based strategy. This strategy is founded on usability goals and brings together two different domains: method engineering and UID methods. In other words, when method engineers rely on usability goals to define a method, they also profit from clearly specifying goals that must be accomplished after each activity is concluded.

Our ongoing and future works are related to extending this proposal to address the organization and sequence of UID activities in a process lifecycle, such as the organization of activities in phases and disciplines; to provide guidance for the integration of UID and software engineering activities; to define activities related to UID, but not necessarily to model-based design and associate them to usability goals; and to propose a solution to execute the method and a strategy for model traceability [1].

Acknowledgments. We gratefully acknowledge the support of the Program Alban, the European Union Program of High Level Scholarships for Latin America, under scholarship number E06D103843BR and the Similar network of excellence (http://www.similar.cc), the European research task force creating HCI similar to human-human communication of the European Sixth Framework Program.

References

1. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Systems Journal 45(3), 515–526 (2006)
2. Ayed, M.B., Ralyte, J., Rolland, C.: Constructing the Lyee method with a method engineering approach. Knowledge-Based Systems 17(7-8), 239–248 (2004)

3. Barclay, P.J., Griffiths, T., McKirdy, J., Kennedy, J.B., Cooper, R., Paton, N.W., Gray, P.: Teallach - a flexible user-interface development environment for object database applications. Journal of Visual Language and Computing 14(1), 47–77 (2003)
4. BEA Systems, IBM Corporation, Microsoft Corporation, SAP AG, Siebel Systems: Business Process Execution Language for Web Services, V1.1 (May 2003)
5. Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Vanderdonckt, J.: Computer-Aided Window Identification in Trident. In: Nordbyn, K., Helmersen, P.H., Gilmore, D.J., Arnesen, S.A. (eds.) Proc. of 5th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact 1995, Lillehammer, July 1995, pp. 331–336. Chapman & Hall, London (1995)
6. Botterweck, G., Hampe, J.F.: Capturing the Requirements for Multiple User Interfaces. In: Proc. of 11th Australian Workshop on Requirements Engineering AWRE 2006, Adelaide, December 9, 2006, Univ. of South Australia (2006)
7. Brinkkemper, S.: Method engineering: Engineering of information systems development methods and tools. Information Software Technology 38(4), 275–280 (1996)
8. Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-Modelling Based Assembly Techniques for Situational Method Engineering. Information Systems 24(3), 209–228 (1999)
9. Brown, A.W., Iyengar, S., Johnston, S.: A Rational approach to model-driven development. IBM Systems Journal 45(3), 463–480 (2006)
10. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15(3), 289–308 (2003)
11. Costa, D., Nóbrega, L., Nunes, N.: An MDA Approach for Generating Web Interfaces with UML ConcurTaskTrees and Canonical Abstract Prototypes. In: Proc. of 5th Int. Workshop on Task Models and Diagrams for user interface design Tamodia 2006. LNCS, vol. 4385, pp. 95–102. Springer, Heidelberg (2006)
12. Fensel, D., Lausen, H., Polleres, A., Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services - The Web Service Modeling Ontology. Springer, Berlin (2007)
13. Ferré, X., Juristo, N., Moreno, A.M.: Framework for Integrating Usability Practices into the Software Process. In: PROFES 2005. Proc. of 6th Int. Conf. on Product Focused Software Process Improvement, Oulu, June 13-18, 2005. LNCS, vol. 3547, pp. 202–215. Springer, Heidelberg (2005)
14. Furtado, E., Furtado, J.J.V., Silva, W.B., Rodrigues, D.W.T., Taddeo, L.S., Limbourg, Q., Vanderdonckt, J.: An Ontology-Based Method for Universal Design of User Interfaces. In: Seffah, A., Radhakrishnan, T., Canals, G. (eds.) Proc. of Workshop on Multiple User Interfaces over the Internet: Engineering and Applications Trends MUI 2001 (Lille, September 10, 2001)
15. Göransson, B., Gulliksen, J., Boivie, I.: The usability design process - integrating user-centered systems design in the software development process. Software Process: Improvement and Practice 8(2), 111–131 (2003)
16. Griffiths, T., Barclay, P.J., McKirdy, J., Paton, N.W., Gray, P.D., Kennedy, J.B., Cooper, R., Goble, C.A., West, A., Smyth, M.: Teallach: A Model-Based User Interface Development Environment for Object Databases. In: Proc. of UIDIS 1999, pp. 86–96. IEEE Computer Society Press, Los Alamitos (1999)
17. Grundy, J.C., Venable, J.R.: Towards an integrated environment for method engineering. In: Proc. of IFIP WG 8.1 Conf. on method Engineering, pp. 45–62. Chapman and Hall, Sydney, Australia (1996)
18. Hailpern, B., Tarr, P.: Model-driven development: The good, the bad, and the ugly. IBM Systems Journal 45(3), 451–461 (2006)

19. Harmsen, F.: Situational Method Engineering. Moret Ernst & Young Management Consultants (1997)
20. Juristo, N., López, M., Moreno, A.M., Sánchez-Segura, M.I.: Improving software usability through architectural patterns. In: ICSE Workshop on SE-HCI 2003, pp. 12–19 (2003)
21. Kruchten, Ph.: The Rational Unified Process - An Introduction. Addison-Wesley, New Jersey (2000)
22. Limbourg, Q., Vanderdonckt, J.: UsiXML: A User Interface Description Language SupPorting Multiple Levels of Independence. In: Matera, M., Comai, S. (eds.) Engineering Advanced Web Applications, pp. 325–338. Rinton Press, Paramus (2004)
23. Mayhew, D.: The Usability Engineering Lifecycle - A Practitioner's Handbook for User Interface Design. Morgan Kaufmann Publishers, San Francisco (1999)
24. OMG, Business Process Modeling Notation Specification, V1.0 (February 2006)
25. OMG, Software Process Engineering Metamodel Specification, V2.0 (February 2007)
26. Rosenbaum, S., Rohn, J.A., Humburg, J.: A toolkit for strategic usability: Results from Workshops, Panels and Surveys. In: Proc. of ACM Conf. on Human Factors in Computing Systems Proceedings CHI 2000, pp. 337–344. ACM Press, NY (2000)
27. Saeki, M.: Came: The first step to automated software engineering. In: Proc. of the OOPSLA 2003 Workshop on Process Engineering for Object-Oriented and Component-Based Development, pp. 7–18 (2003)
28. Sinnig, D., Gaffar, A., Reichart, D., Seffah, A., Forbrig, P.: Patterns in Model-Based Engineering. In: Proc. of CADUI 2004, pp. 195–208. Kluwer Academic Publishers, Dordrecht (2004)
29. Si-Said, S., Rolland, C., Grosz, G., MENTOR,: A Computer Aided Requirements Engineering Environment. In: Constantopoulos, P., Vassiliou, Y., Mylopoulos, J. (eds.) CAiSE 1996. LNCS, vol. 1080, pp. 22–43. Springer, Heidelberg (1996)
30. Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 16–31. Springer, Heidelberg (2005)
31. Visual Paradigm. Business Process Visual Architect. Available at: http://www.visual-paradigm.com/product/bpva/
32. Wolff, A., Forbrig, P., Dittmar, A., Reichart, D.: Linking GUI elements to tasks: supporting an evolutionary design process. In: Proc. of TAMODIA 2005, pp. 27–34. ACM Press, New York (2005)
33. Zhang, Z., Lyytinen, K.: A Framework for Component Reuse in a Metamodelling-Based Software Development. Requirements Engineering 6(2), 116–131 (2001)