Service-Oriented Architecture for Supporting Distributed User Interaction Design

Hildeberto Mendonça Filho[1], Kênia Sousa [1], Elizabeth Furtado[2]
[1] Université catholique de Louvain, Unité des Systèmes d'information, Place de Doyens 1, 1348 Louvain-La-Neuve, Belgium
{mendonca, sousa, vanderdonckt}@isys.ucl.ac.be
[2] Universidade de Fortaleza, Av. Washington Soares, 1321
Fortaleza, Ceara Brasil - elizabet@unifor.br

Abstract. Professionals working in organizations conducting user interaction design are more and more involved in a collaborative setup where competences and resources are distributed in time and space. In order to support this shift of practice, a service-oriented architecture is defined and developed according to principles of model management. In this paradigm, user interaction design is decomposed into activities, which could be supported by model management operations. These operations are in turn converted into services, developed according to the architecture. A distributed user interaction design life cycle consequently involves the following steps: a method engineer defines the activities to be conducted for a user interaction project, the method definition is imported in the architecture to enact the method by assigning responsibilities to team members, and these members then perform their responsibilities through the services corresponding to the operations.

Keywords: User interaction design method, service-oriented architecture, meta-models, business process modeling, model management.

1 Introduction

The Cameleon Reference Framework [4] proposed a set of models, compatible with MDA [13, 20], that provide the necessary support for the currently user interaction challenges. This framework has 5 models distributed in 4 levels of abstractions that are intended to express the UI development life cycle. The proposal of the Cameleon Framework achieves the necessary support for different platforms, devices and contexts of use and a considerable effort is running to provide computational and methodological applicability.

One of the bases of this work was to define a language to represent all aspects of the framework in a computational form. The language UsiXML [12] was created as a XML extension to describe UIs for multiple contexts of use and it motivated the implementation of a set of tools to manipulate the language [21]. Every day, the integration between all these tools becomes more critical to guarantee the life cycle's completeness and coherence. At the same time, all possible operations in a life cycle

could not be provided by tools because there are some dimensions to be considered, as management, control, integration with external environments, and so on.

We are now proposing one approach that could standardize the integration between tools and make the application of methods possible to support the entire life cycle, including the integration of these methods in a pre-existent and deployed development process. The basic idea is to provide all model operations as services that could be reused by different tools and applications. The UsiXML language will provide the necessary support to represent models in a structured and reasonable form.

The service's concept is the same used by Service-Oriented Architecture (SOA) that considers it as a software component, but with special ability to improve the software composition and distribution. A service is published in the World Wide Web (WEB) infrastructure and can be invoked by a software to execute part of its logic. Applying it in our problem, a service can provide a set of visible benefits listed below:

- reduce the complexity of the current tools and applications available and improve the architecture of the future implementations;
- enable the collaborative work locally, distributed (e.g. intranet), and remotely distributed (e.g. internet);
- avoid the duplication of implementations and value the variety of implementation strategies;
- the number of services available is directly proportional to the number of possibilities to create new applications and methods; and
- totally based in widely accepted patterns, managed by W3C, OMG, OASIS, etc.

The research aims to provide a consistent solution to allow a collaborative work in the UI design and to bring a definitive contribution to facilitate the application of HCI practices in real world organizations. This paper intends to present the foundations and the structure of the research, talking about models, methods and technology, but every time thinking about how it can contribute to the software's end-users.

This paper is organized as follows: section 2 presents related works; section 3 shows the use of services for model management; section 4 presents the method specification; section 5 details the deployment of services to support communication; section 6 illustrates the method execution with a case study; section 7 concludes this work by presenting the contribution and future work.

2 Related Works

In order to achieve competitive results and still address constant organizational changes, SOA has been widely applied to bring agility in the business process definition and improve the communication between organizations and even between distributed people of the same organization.

A model proposed by Colombo et al. [5] shows that web services, the key technology to concretize SOA, are an effective solution to let software systems, developed by different organizations and spread across the world, interoperate. This model has support to i) Agent-actors: identification of stakeholders and roles; ii) Core Service: a particular concrete resource which is offered by a Software System; iii) Service Description: a syntax description about the service interface to show the

potential offered; iv) Service Discovery: a process to discover new services in the network to become available for services composers; v) Service Composition: the service capability to execute in cooperation with other services in a same transaction; vi) Service Publication: to make services available in the network, exposing theirs service descriptions to be recognized by other applications; and vii) Service Monitoring: production of statistics data to build quality metrics. All these concepts allow the method automation, creating services to manage artifacts, requirements, tests and so forth, putting all the services in a logic sequence and changing it whenever necessary or creating different versions of the method, considering the size or the complexity of projects.

There are many approaches that address the challenge of efficiently managing models during the application of UID methods:

MANTRA (Model-bAsed eNgineering of multiple interfaces with TRAnsformations) [3] is a model-driven approach for the development of multiple UIs. MANTRA is structured by abstraction levels similar to the Cameleon Framework [4]. The model transformations, such as adapting the AUI according to the requirements and transforming the adapted AUI into several CUIs, are described in ATL (Atlas Transforming Language) [9], a hybrid model transformation language that allows both declarative and imperative constructs to be used in transformation definitions. The declarative style of transformation is based on specifying relations between source and target patterns. The imperative part in the transformation language is explicitly encoded by the developer. In MANTRA, web services are used to implement the application as an interface between the UIs and the application core, not as a technology to perform model transformations.

Wolff et al. [22] propose an approach and a tool to support transformations between models supported by patterns. The transformation between the dialog graph (specification of views, their association to tasks, and the transitions between tasks and views) and the AUI is done by mapping views to windows and elements of views to buttons. The connections with the task model are kept, which facilitates tracing actions on elements back to tasks and the simulation of dialog graphs. The transformations are not fully automated, but they are supported by humans using interactive tools. For instance, the transformation from AUI to CUI is done by the designer replacing (via “drag & drop”) AUI elements by a pre-designed component, that is, UI patterns.

Similarly, Sinnig et al. [19] use patterns during the application of a model-based development methodology with tool support based on XML-representation of patterns for each level, namely dialog, presentation and layout patterns.

Even though the use of patterns during UID promotes re-use, standardization, and efficiency to the transformation of models; model transformation logic are coded inside the tools, thus making it more difficult to reuse implementation strategies.

These approaches have tools to support UID methods, but the descriptions of these tools do not mention any technology that supports collaborative work. Thus, these tools support the creation of models and automated transformations in some level, but professionals can not use them to communicate and share their work.

These approaches follow a formalized method, but their supporting tools do not provide facilities to change the sequence of the method activities, thus restricting the possibilities to adapt the method. Some approaches use XML-like files to store the

models or XML-based implementation language, thus, they are aligned with standards. Following, we will demonstrate how our approach addresses these issues.

3 Services for Model Management Implementation

The UsiXML language describe the user interface for multiple context of use such as character user interfaces, graphical user interfaces, auditory and vocal user interfaces, virtual reality, and multimodal user interfaces. As a language explicitly based on the Cameleon Reference Framework, its adopts four development steps, which are 1) Task & Concepts, 2) AUI, 3) CUI, and 4) Final UI, where are distributed five models, which are task model, domain model and context model in the first step, abstract user interface (AUI) in the second step, and concrete user interface (CUI) in the third step. The language does not consider the Final UI, but offers the necessary support to concept and generate it, when it is a specific technology [21].

The models have a natural logic sequence. However, it sequence could not support many situations. For instance: if we already have a concrete user interface and a new device needs to be considered, it is necessary to transform that CUI in an AUI, to have a model platform independent. Moreover, there are many other possibilities that are treated by a model management approach. It intents to be a higher level programming interface that offers a set of operations, supported by mappings between models [2]. Our approach is to provide a set of model management operations as services to allow a collaborative work oriented by one or more methods, to integrate all available tools and to improve the architecture of the current and future tools. Other types of services are necessary to promote control and integration, as depicted in Table 1.

Table 1. Types of services

Type	Service	Description
Model management	Transform AUI to CUI, merge	Operations over models.
Control	Version check in / check out, exclude, add, etc.	Physical control over managed models.
Integration	Generate class diagram, refactoring, trace	Integration of UsiXML models with other models and processes.

A service has different levels of granularity. A unique service can be responsible for an entire process, a sub-process, a process step or an operation of a process step [7]. The granularity is inversely proportional to the possibilities of services composition. So, we choose a low level of granularity to increase the applicability of the services in different situations.

In some situations, a specific service can be invoked separately by a tool, but, in most of the situations, an atomic service collaborates with other services to produce significant results. For example: a tool can call a specific transformation service to transform an AUI model in a CUI model. It will be done, but in a real application, a CUI Model could already exist and it is necessary to use a comparison service to

check all changes made over the original CUI model and call a refactoring service to propagate each change in all managed models.

A SOA provides support for process orchestration, which is used to describe how services can interact with each other to provide new applications for each specific situation in the organization [17]. Orchestration promotes the services composition to make possible simple logics, as explained in the last example, and complex process definition, such as a software development process.

But, as described by Sadiq et al. [18], orchestration is a creative work and, until now, it needs to be done by human beans. So, this concept is not self evident and to be applied, we need to consider a specific domain, as HCI, a specific application, as model management, and a method to describe in details how everything works.

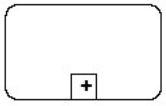
4 Service Orchestration to Support UID Methods

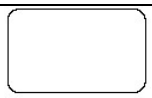
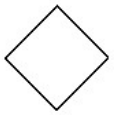
A method supporting model management addresses model transformations as a means to produce UIs for a variety of platforms, such as in [3, 19, 22]. But, we intend to encompass a broader definition by considering the application of standards in the method definition.

To make a representative method, that is, to make it clearer to understand, apply and implement, it is important to present it in a flow format. Business Process Modeling Notation (BPMN) [15] was proposed to be applied in the representation of organizational processes. The use of BPMN in the method definition is important 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) there is a strong integration with the SOA.

There are a lot of elements in the BPMN notation. To take a complete overview of the notation, the specification is available on [15]. We will present a subset of these elements, which are necessary to make comparisons with an interactive software development lifecycle and to present the method's structure. The elements are described in Table 2. Since BPMN can be used in any business process modeling, we want to demonstrate that it is in accordance to Software Process Engineering Metamodel (SPEM), as described in the next sub-section, before actually presenting the method using BPMN elements.

Table 2. Description and notations of the BPMN elements, adapted from [15].

Element	Description	Notation
Process	A set of sub-processes, activities or task linked by different types of connectors to represent an organizational routine.	No specific visual notation, but consider a diagram as a notation.
Sub-Process	It is a process inside another one. It exists for two reasons: to reduce the complexity of an activity, dividing it in smaller ones or to reuse a low level process in a related process.	
Activity	It composes a process or a sub-process, representing the steps to complete the work successfully. It can be represented by a	The notation of an activity can be the notation of a sub-process or of a task.

	process, sub-process or task.	
Task	It is an atomic activity, a work that can not be broken into parts. If it is necessary to break it, it can become a sub-process.	
Group	It is used to bring a new perspective for pieces of the flow, but it does not affect the sequence flow, it just adds documentation and analysis information.	
Pool	It represents a participant (entities of roles) with responsibility over the pool's elements in a process.	
Lane	It organizes activities within a pool and represents sub-entities or sub-roles. There is at least one lane within a pool.	
Data Object	It is an artifact, used as input or output of activities. It has a state that indicates the impact of the process execution on it.	
Gateway	It is a mechanism that controls the passage in the process flow. There are different kinds: exclusive, inclusive, complex decision, and parallel fork/join.	
Conditional Flow	It represents a condition in the process that needs to be evaluated in order to decide the flow to be taken.	

We associate SPEM and BPMN to help method engineers specify the method and also to adhere to formal SE specifications. SPEM is a meta-model for defining software development processes and their components [16].

In a general view, the structure of the SPEM meta-model is organized with the elements presented in Table 3, which demonstrates the association of SPEM components with BPMN components.

Table 3. Association of SPEM and BPMN elements

SPEM	BPMN	Comments
Process	Process	Group of process elements to achieve a common goal.
Lifecycle	Sub-process	They can be detailed with a set of sub-activities.
Phase		
Iteration		
Discipline	Group	Organization of activities in a same group.
Activity	Activity	An atomic or non-atomic work.
Step	Task	Represent an atomic activity.
Process Performer	Pool	Represent the responsible for activities.
Process Role	Lane	Represent a specialization of a process performer or pool.
Work Product	Data object	Considered as artifacts.
Precondition	Gateway Conditional flow	Represent a condition or a decision point in the flow.

After we analyzed these components, we realized that it is possible to represent the concepts defined in SPEM using BPMN. This means that a method defined according to the SPEM specification can be graphically presented using the BPMN formalism, as presented in the next section.

4.1 Method Description

We are currently working on the definition of a method; therefore; this is just an illustrative method to validate the proposed application with SOA. Some activities of the method can be the following.

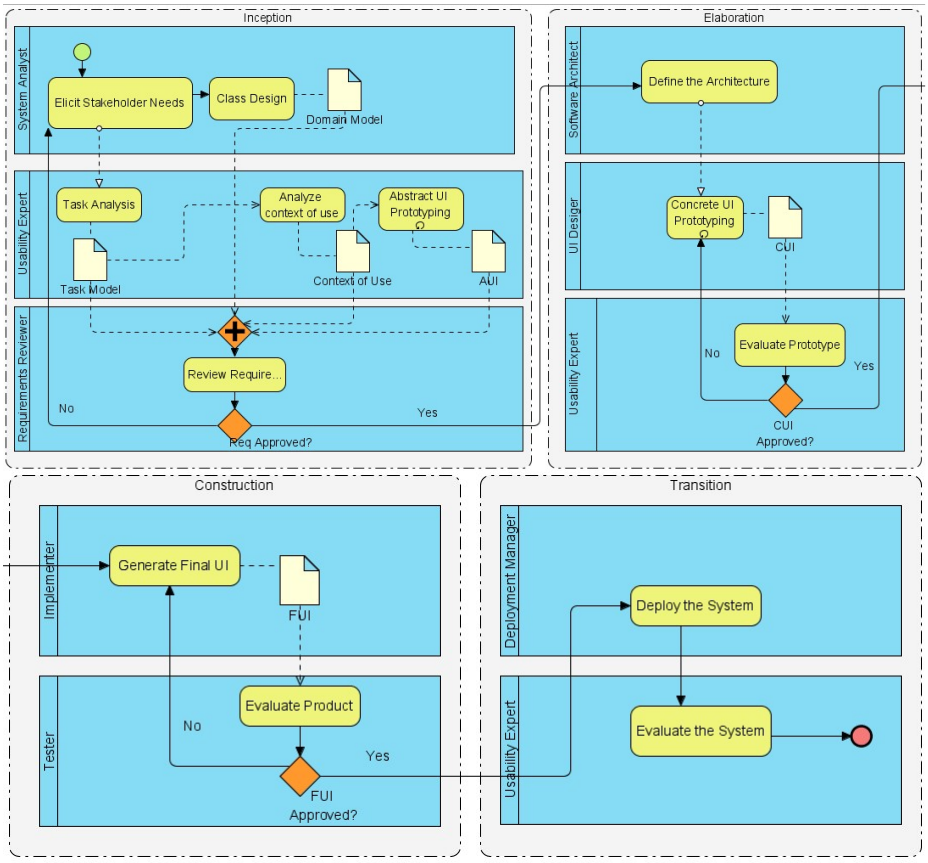


Fig. 1. Example of the method defined using a BPMN editor.

In the inception phase, the system analyst elicits requirements, then the usability expert describes the tasks, the context of use, and the prototype in abstract level while the system analyst performs class design, then the requirements reviewer analyzes all the models that have to be approved before going to the next phase. In the elaboration phase, the software architect defines the architecture and the UI designer designs the UI in a concrete level, then the UI is evaluated and approved by the usability expert

before going to the next phase. In the construction phase, the implementer implements the UI in the final level. The tester evaluates the UI before approving it for the final phase. In the transition phase, the deployment manager installs the system in the environment where the usability expert evaluates the system with end users, who have to approve it to consider the project finalized. Throughout the method, when an artifact is not approved, a new iteration can be executed until an acceptable version of the artifacts is produced.

Fig. 1 illustrates the method using a flow chart organized according to the RUP structure [11] in four phases: inception, elaboration, construction, and transition. These phases are composed of activities, organized in five disciplines: requirements, analysis and design, implementation, deployment, and test. The main roles are: system analyst, usability expert, requirements reviewer, UI designer, software architect, implementer, tester, and deployment manager. The main UsiXML models are: task model, domain model, context model, abstract UI (AUI), and concrete UI (CUI). With the use of a UID method for the application of services for model management, we can present how SOA provides support for professionals working in a distributed manner towards the creation of consolidated artifacts.

5 Support for Computer-Mediated Communication

Technically speaking, the implementation of services in a specific platform is called Web Services [10]. The added term "Web" means that the service is available in an internet environment, and it can be invoked locally or remotely by other applications. All the communication is managed by a protocol called Simple Object Access Protocol (SOAP), based on XML, which is responsible for the encapsulation and transport of request messages from the client to the server and response messages from the server to the client.

The availability of a web service can improve substantially the management of the UsiXML models. One of the first steps is to migrate all the code in the tool implementation that manipulates UsiXML models to web services. It will reduce the complexity of the tool's code, make the implementation available for other tools, and improve the architecture of future tool implementations. A catalog of available services, among other things, will avoid the duplication of implementations.

A SOA is the basis to enable a correct design and implementation of all model management web services. According to [7], "SOA represents a open, agile, extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services". Part of the above advantages is supported by web services and orchestration, a concept introduced in section 3.

As we discussed in section 4, BPMN will provide the higher level definition of a SOA application to execute methods to offer support in the UID. Currently, BPMN can be transformed in the Business Process Execution Language (BPEL) [1]. In fact, most of the tools that work with BPMN can generate BPEL [8], reducing the effort of achieving an automated process. BPEL is a XML based language that can be executed by a BPEL engine that allows composition of web services and communication between them.

Each method activity, described in BPMN, is represented as a BPEL task, which is associated with web services. The engine controls the execution of the flow, the state of process, the time spent in each task, saving important data for the process analysis.

Fig. 2 represents the SOA support for methods and tools. We can see the transformation of BPMN in BPEL and the deployment of a BPEL in its engine. The engine can orchestrate services inside and outside the organization, widely distributed. A service-oriented application client will execute instances of the method and tools can make use of a particular web service and participate of the method execution, offering the support for a particular activity.

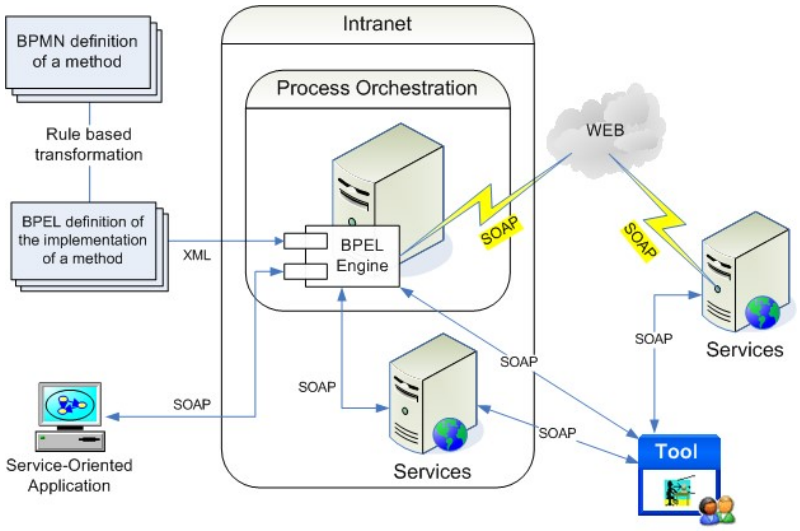


Fig. 2. The SOA support for methods and tools.

6 Case Study

Consider a software organization, located in Europe, which is responsible for a project for the design of UIs for an e-commerce web site for desktops and Palm Tops. But, the organization needs an expert in UI design for Palm Tops. Therefore, the directors decide to hire such an expert for this specific project, but who works remotely from South America.

In order to improve their communication, the project manager trains the hired professional on the method and on the method execution application so all the outcomes are shared, that is, they are available for all professionals at anytime, anywhere.

Following, we present how the execution of the method can be automated with SOA by depicting prototypes of the method execution application, a service-oriented application, and images of the models created during the execution of certain activities.

During the *Task Analysis* activity of the inception phase (Fig. 1), the usability expert uploaded the UsiXML task model file (see Fig. 3), created with IdealXML [14]

(see Fig. 3) or another tool that creates the UsiXML file. Whenever the usability expert makes changes in this model, he/she can upload the new version and a service is called to perform version control operations.

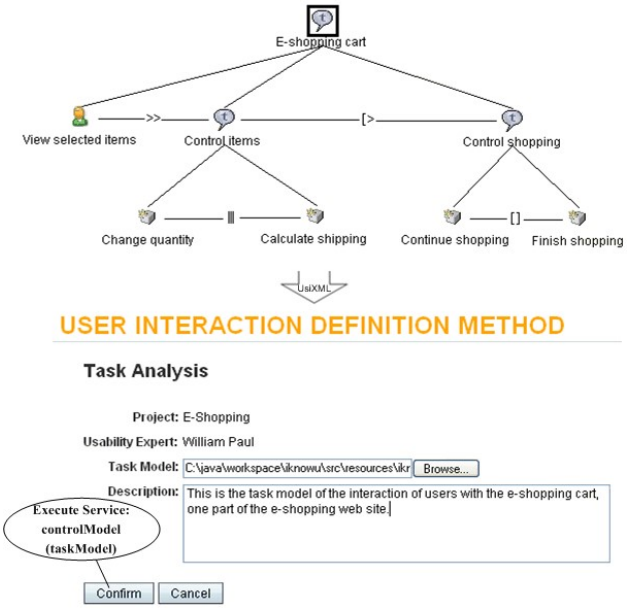


Fig. 3. Task Analysis

During the *Analyze Context of Use* activity of the inception phase (Fig. 1), the usability expert can inform the data about the context of use (see Fig. 4 for environment data). In this application, there are also pages devoted for user profile and platform. After requesting the data to be saved, a service is called to perform a model management method that creates the UsiXML context of use file.

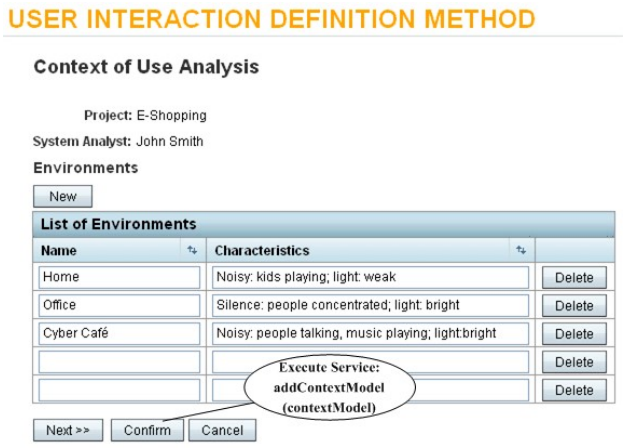


Fig. 4. Analyze context of use.

During the *Abstract UI Prototyping* activity of the inception phase (Fig. 1), the usability expert uploads the UsiXML AUI model file, which was generated using IdealXML. The AUI could also have been automatically generated by another tool, which can call specific services that perform the transformation from conceptual models (task, domain, and context of use) into the AUI. When the resulting UsiXML AUI is uploaded, a service is called to execute version control operations. If a new version of the AUI is uploaded, a service for refactoring is called to make the appropriate changes in the models associated to this AUI model, such as the conceptual models or even in the CUI, in cases when it was created, for instance, in a previous iteration of the lifecycle.

During the *Class Design* activity of the inception phase (Fig. 1), the system analyst can create the class diagram using any UML editor (see Fig. 5), upload the XMI file of the class diagram, and then request the generation of the UML class diagram into a UsiXML domain model (see Fig. 5). This service performs a specific model management operation, which facilitates the integration of artifacts prepared using another modeling language with UsiXML models. It is visible here that any modeling tool can be used, and each one can have a different specification, visualization, among other aspects. But, with this service-oriented application, we reinforce consistency in the application of the method by providing such services for specific transformations.

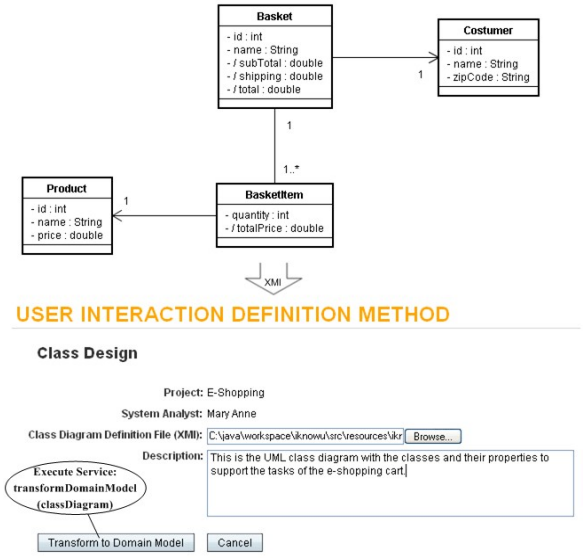


Fig. 5. Class Design

During the *Review Requirements* activity of the inception phase (Fig. 1), the requirements reviewer evaluates each of the models mentioned previously. When the reviews are saved, a service is called to send e-mails with the status of the evaluation and the list of possible changes for the professionals responsible for the models that need changes. This is an example of a service that facilitates the coordination between professionals. This service is associated to the other activities that evaluate artifacts, such as the *Evaluate Prototype, Evaluate Product,* and *Evaluate the System* activities.

During the *Concrete UI Prototyping* activity of the elaboration phase (Fig. 1), the usability expert creates the CUI model using SketchiXML [6]. When he/she uploads it, a service for version control is called. Similarly to all other models, the upload of a new version of the model triggers the service for refactoring. Since the models are in a common repository, it is easier to perform such a service.

During the *Evaluate Prototype* activity of the elaboration phase (Fig. 1), the expert in UIs for Palm Tops also plays the role of a reviewer and verifies if specific guidelines for this device are addressed in the CUI. This is one of the method activities that triggers refactoring services, since changes made in one model have to reflect on all associated models.

During the *Generate Final UI* activity of the construction phase (Fig. 1), the implementer selects the language to generate the UI (see Fig. 6). When the implementer requests the generation of HTML and WAP files, the service is called to generate the Web page and the page in a Mobile Phone (see Fig. 6). Consider that this is an automatically generated UI, in which the designer will still incorporate the organization's style guide into it.

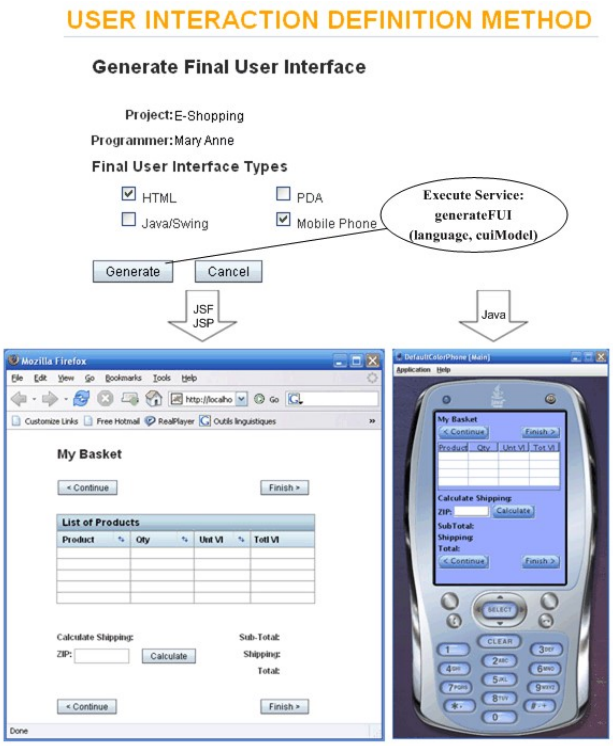


Fig. 6. Generate Final User Interface

These screens depict the most common interactions with the application that manages the method lifecycle. When professionals perform these activities, they interact with the application to: document the outcomes of their work (i.e. uploading files), perform their work (i.e. defining the context of use), receive outcomes that help

in their work (i.e. use the generated FUI and add business rules in it), control the status of their work (i.e. view which activity of the process is being executed, receive notifications of which activity they need to perform), etc.

This case study demonstrates that the execution of a method supported by a service-oriented application reinforces consistency in a project and even across different projects. Also note that if the method flow of activities is changed in the BPMN editor, so is the associated BPEL, therefore, the services will be invoked in a different sequence, accordingly to the method, thus, increasing the flexibility to make changes in the method, whenever appropriate.

7 Conclusion

This work presented an approach that uses SOA to support the transformation between models during the UID lifecycle. The specific issue we address is the support for communication through the use of an application that allows sharing and integrating the work performed by the software organization teams either locally (e.g. professionals designing the UI physically located in the same organization), distributed (e.g. professionals accessing the organization intranet), or remotely distributed (e.g. professionals accessing the internet to use the common application).

Considering the existence of a set of tools that manipulate UsiXML during the UID lifecycle, we provide the foundation to integrate them through the reuse of services that manipulate UsiXML models to perform transformations.

With the case study, we exemplified how real world organizations can use a standardized architecture to reuse implementation strategies in different projects to promote the institutionalization of UID by supporting collaborative work.

There are some topics that are open for future work. For the *method*, we will work on its specification in details, and define how to make adaptations, depending on the organization or project. For the *service*s, the next step is to implement them. Concerning *project management*, we intend to monitor the method using an engine that generates statistical data (e.g. comparison of project planned dates with real dates), and improve the method based on the results of the statistical data.

Acknowledgments. We gratefully acknowledge the support of 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 and the support by the Programme Alban, the European Union Programme of High Level Scholarships for Latin America, scholarship number E06D103843BR.

References

1. BEA Systems, IBM Corporation, Microsoft Corporation, SAP AG, Siebel Systems, Business Process Execution Language for Web Services, 1.1, May 2003
2. Bernstein, P. Applying Model Management to Classical Meta Data Problems. In: Conference on Innovative Data Systems Research (2003) 209-220

3. Botterweck, Goetz and J. Felix Hampe. 2006. Capturing the Requirements for Multiple User Interfaces. 11th Australian Workshop on Requirements Engineering (AWRE'06), December 9, 2006, Adelaide, SA, Australia
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers (2003) 289–308
5. Colombo M., Di Nitto E., Di Penta M., Distante D., Zuccalà M. Speaking a Common Language: A conceptual Model for Describing Service-Oriented Systems. In ICSOC, Amsterdam (2005) 48-60
6. Coyette, A., Vanderdonckt, J., Limbourg, Q., SketchiXML: An Informal Design Tool for User Interface Early Prototyping. In Proc. of RUIPICAS'2006, M. Risoldi, V. Amaral (Eds.), Geneva (2006)
7. Erl, Thomas. Service-Oriented Architecture – Concepts, Technology, and Design. Prentice Hall, Crawfordsvile, Indiana (2005)
8. Harvey, Michael. Essential Business Process Modeling. O'Reilly Media (2005)
9. Jouault, Frédéric; Kurtev, Ivan. Transforming Models with ATL. Model Transformations in Practice (Workshop at MoDELS 2005), Montego Bay, Jamaica (2005)
10. Kreger, Heather. Web Services – Conceptual Architecture. IBM Software Group (2001)
11. Kruchten, Philippe. The Rational Unified Process - An Introduction. 2 ed. Addison-Wesley, New Jersey (2000)
12. Limbourg, Q., Vanderdonckt, J. UsiXML: A User Interface Description Language Sup-Porting Multiple Levels of Independence. In Matera, M., Comai, S. (Eds.), Engineering Advanced Web Applications, Rinton Press, Paramus (2004) 325-338
13. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley, New York (2004)
14. Montero, F., Víctor López Jaquero, V., IDEALXML: An Interaction Design Tool and a Task-based Approach to User Interface Design, Proc. of CADUI'2006, Chapter 20, Springer-Verlag, Berlin (2006) 245-252
15. OMG, Business Process Modeling Notation Specification, 1.0, February, 2006
16. OMG, Software Process Engineering Metamodel Specification, 1.1, January 2005
17. Peltz, C. Web Services Orchestration and Choreography. IEEE Computer, Vol. 36, 10 (2003) 46-52
18. Sadiq, Waqar; Racca, Felix. Business Services Orchestration - The Hypertier of Information Technology. Cambridge University Press, Cambridge (2003)
19. Sinnig, Daniel; Gaffar, Ashraf; Reichart, Daniel; Seffah, Ahmed; Forbrig, Peter. Patterns in Model-Based Engineering. In Proc. of CADUI'2004, Madeira, Portugal (2004) 195-208
20. Soley, Richard. Model Driven Architecture. Available at: http://www.omg.org/mda/presentations.htm. OMG White Paper (2000)
21. Vanderdonckt, Jean, A MDA-Compliant Environment for Developing User Interfaces of Information Systems, Proc. of CAiSE'05, O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin (2005) 16-31
22. Wolff, Andreas; Forbrig, Peter; Dittmar, Anke; Reichart, Daniel. Linking GUI elements to tasks: supporting an evolutionary design process. Proc. of TAMODIA'2005, Gdansk, Poland (2005) 27-34