Service-Oriented Architecture for Supporting Collaborative User Interface Development

Hildeberto Mendonça[1], Kênia Sousa[2], Jean Vanderdonckt[2]
Université catholique de Louvain
[1]Place du Levant, 1 - [2]Place de Doyens, 1 - 1348 Louvain-La-Neuve, Belgium
{hildeberto.mendonca, kenia.sousa, jean.vanderdonckt}@uclouvain.be

ABSTRACT
Professionals working in organizations that conduct any user interface development life cycle 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 development is decomposed into activities, which could be supported by model management operations. These operations are in turn converted into services, developed according to the service-oriented architecture. A distributed user interaction development life cycle consequently involves the following steps: a method engineer defines the activities to be conducted for the user interface of a project, the method definition is imported in the software 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. This paper also presents a taxonomy of services for supporting the user interface development life cycle that is unique.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Computer-aided software engineering (CASE), User interfaces* H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces (GUI), User Centered Design*. I.3.6 [**Methodology and Techniques**]: Device Independence, Language, Standards. H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces - *Collaborative computing, Computer-supported cooperative work.*

General Terms
Design, Human Factors, Standardization, Languages.

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

INTRODUCTION
Service-Oriented Architecture (SOA) has been successfully applied to a wide variety of domains of human activity, inside and outside computer science, such as, but not limited to: database management, distributed computing, cloud computing. One discipline of computer science that has received little or no attention with respect to SOA is Human-Computer Interaction (HCI), which is the discipline that is aimed at defining and applying a user interface development life cycle for any project. This has been for a while primarily because HCI has not evolved as fast as other disciplines in integrating SOA concepts and experience and because integrating Software Engineering §SE) techniques in HCI has been challenging for many years. This paper is aimed at addressing the need of supporting a structured user interface development life cycle using SOA.

Although still evolving, one reference framework has today received some consensus throughout the HCI community: the Cameleon Reference Framework (CRF) [5] proposes a set of models, compatible with MDA [16,24], that provide the necessary support for addressing the current challenges posed by User Interface (UI) development. This framework consists of 5 models distributed in 4 levels of abstractions that are intended to express the UI development life cycle. The CRF proposal achieves the necessary support for different platforms, devices and contexts of use and a considerable effort is running to provide computational and methodological applicability. Different implementations exist today that exhibit various levels of compliance with the CRF.

One of the bases of this work was to define a language to represent all aspects of the CRF in a computational form. The language UsiXML [15] 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 [25]. 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 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 potential benefits such as:

- 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.

RELATED WORK

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. [6] 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.

One collaborative approach to improve the user interaction design process is not a strongly explored subject as it really deserves. Actually, there exists a real desire to increase the return of investment on the HCI adoption as we have noticed in some works [3,7,10,22], which use different approaches like reducing the gap between Software Engineering and Human-Computer Interaction or creating new techniques to deliver a well-accepted user interface.

We are not trying to create a new form to reduce the gap between SE and HCI or a new technique. But, with SOA, it is possible to reduce the mentioned gap and use new techniques throughout the activities. It is fundamental to use models, represented in a structured language, such as UsiXML, to improve the performance by model transformations, avoiding tocreate models since the beginning.

MANTRA (Model-bAsed eNgineering of multiple interfaces with TRAnsformations) [4] is a model-driven approach for the development of multiple UIs. MANTRA is structured by abstraction levels similar to the CRF [5]. 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) [12], 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 contrast with our approach, MANTRA uses web services to implement the final application, not as a strategy to improve the method applicability and adaptation to the dynamism of the organization.

Wolff *et al.* [27] 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 a platform-independent UI model (PIM) is produced 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 Platform-Independent Model (PIM) to Platform-Specific Model (PSM) is achieved by the designer replacing (via "drag & drop") PIM elements by a pre-designed component, that is, UI patterns.

Similarly, Sinnig *et al.* [23] 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 User Interface Development (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 cannot 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.

UsiXML follows a multi-path UI development based on the CRF [5], which defines UI development steps for multi-context interactive applications in a simplified manner. Now, we present that this framework can also be applied with a distributed team. Figure 1 depicts that in each level of the CRF, it is possible to consider people working in different time and space, but still connected through the execution of web services that support the method.

The MBUID method used to illustrate the workflow execution is divided 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 artifacts are a combination of SE and HCI, for instance, UML diagrams [10] and UsiXML models [22], respectively. The method engineer is the one responsible for the definition and maintenance of the method. To facilitate his/her work, standard notations and workflow editors can be used, which are presented in the next sections.

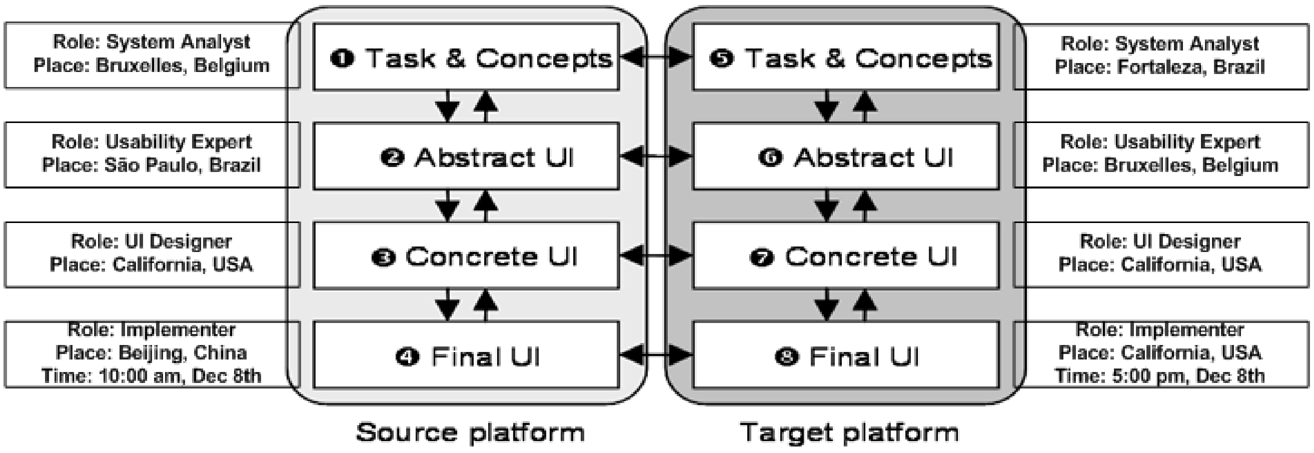


Figure 1. Distributed team applying the Cameleon Reference Framework.

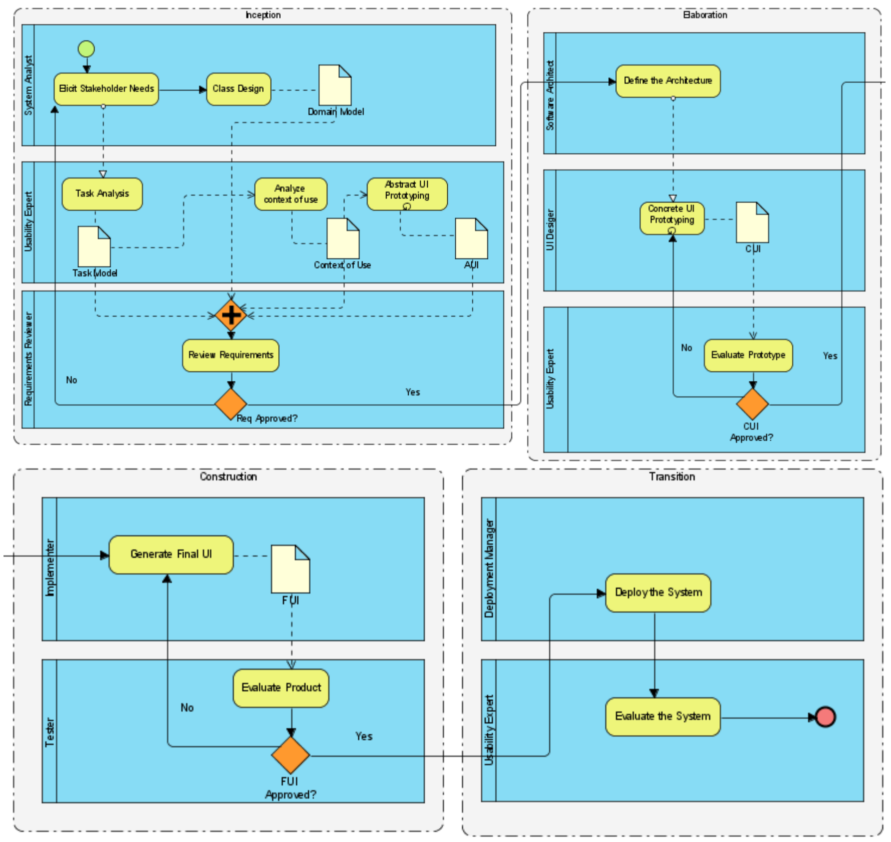


Figure 2. Example of method defined using BPMN editor.

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 (Figure 1):

1) *Task & Concepts* (T&C): describe the various user's tasks to be carried out and the domain-oriented concepts as they are required by these tasks to be performed.
2) *Abstract UI* (AUI): defines abstract containers (AC) and individual components (AIC) [15] two forms of Abstract Interaction Objects (AIO) [25] by grouping subtasks according to various criteria (e.g., task model structural patterns, cognitive load analysis, semantic relationships identification), a navigation scheme between the container and selects abstract individual component for each concept so that they are independent of any interaction modality, such as graphical, vocal, tactile, or haptic modalities. The AUI is said to be *independent of any interaction modality*.
3) *Concrete UI* (CUI): concretizes an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) [25] so as to define widgets layout and interface navigation. It abstracts a final UI into a UI definition that is independent of any computing platform. While the AUI is independent of any interaction modality, a CUI assumes that a particular interaction modality has been chosen, but that this CUI remains *independent of any computing platform*.
4) *Final UI* (FUI): is the operational UI i.e. any UI running on a particular computing platform either by interpretation (e.g., through a Web browser) or by execution (e.g., after compilation of code in an Integrated Development Environment (IDE).

Across these four development steps 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 [25]. Task & Concepts, respectively AUI, CUI, are the manifestations of Computing-Independent Model (CIM), respectively PIM, and PSM.

The models have a natural logic sequence. However, its sequence could not support many situations. For instance: if we already have a CUI 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 intends to be a higher level programming interface that offers a set of operations, supported by mappings between models [2]. Our approach is aimed at providing 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.

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.

Table 1. Types of services.

A service has different levels of granularity. A unique service can be responsible for an entire process, a subprocess, a process step or an operation of a process step [9]. 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 [20]. 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. [21], orchestration is a creative work and, until now, it needs to be done by human beings. 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.

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 [4,23,26]. 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) [18] 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) many software is 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 [18]. 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.

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 [19].

We associate SPEM and BPMN to help method engineers to specify the method using a workflow, and also to adhere to formal SE specifications. SPEM is a meta-model for defining software development processes and their components [18]. In a general view, the structure of the SPEM meta-model is organized as follows: *ProcessComponent* is a set of process description to assemble a process; *Process* is a process component intended to stand alone as a complete process; *Lifecycle* is the behavior of a process and it is defined as a sequence of phases; *Phase* is defined with the constraint of being executed sequentially with a series of milestones spread over time; *Iteration* is a composite of a phase with a minor milestone; *Discipline* partitions the activities of a process according to a common theme; *WorkDefinition* is the work performed in the process; *Activity* (specialization of WorkDefinition) is a piece of work performed by a process role; *Step* is an atomic element of an activity; *ProcessPerformer* performs a set of work definitions in a process; *ProcessRole* (specialization of ProcessPerformer) is responsible for specific work products by performing and assisting in specific activities; *WorkProduct* is an artifact produced, consumed, or modified by a process; and *Precondition* is a constraint that is expressed in terms of the states of the work products that are parameters of the work definition. The lifecycle, phase, and iteration can be represented as a BPMN sub-process. For instance, a lifecycle can be represented as a sub-process composed of phases. The discipline can be represented as a BPMN group in order to delimitate the activities that are part of the same discipline. The activity can be a BPMN activity, since it can represent tasks, operations, and actions. When an activity is consisted of atomic elements, a step can be represented using a BPMN task, which is an atomic activity. A process performer can be represented as a BPMN pool since both are more generic representations of process participants and a process role (a subclass of a process performer) can be represented as a BPMN lane, which is a subpartition within a pool. Work product is a BPMN data object since both are considered as artifacts in the process. Precondition can be a BPMN gateway that controls the sequence flow; or a BPMN conditional flow, which has condition expressions evaluated at runtime to determine whether or not the flow will be used.

SPEM suggests the definition of a method using UML diagrams, in particular, the following UML notations are useful: class diagram, package diagram, activity diagram, use case diagram, sequence diagram, and state chart diagram. Method engineers can use any BPMN editor to define the process workflow and make it available for professionals in their organization as guidance for their work, as depicted in Figures 2 and 3: the business process notation is used by tools to draw the method flow and all objects shown here can be found in the left tab.

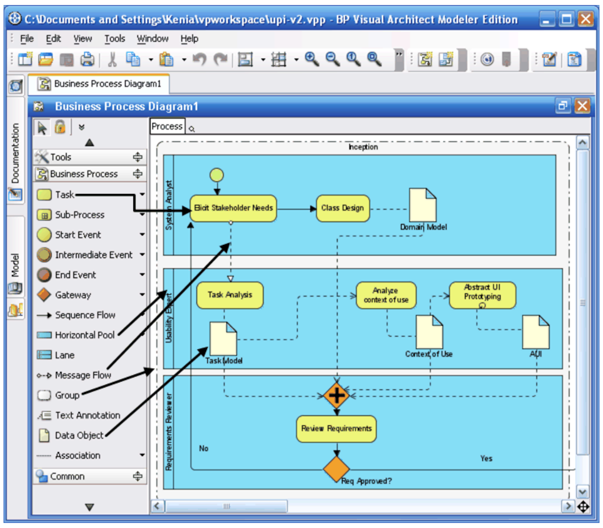


Figure 3. Defining the method with a BPMN editor.

Method Description

The method description is base on the following phases.

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.

Figure 3 illustrates the method using a flow chart organized according to the RUP structure [14] in four phases: inception, elaboration, construction, and transition. This flow was designed using the tool Business Process Visual Architect [26]. 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.

Method Implementation

Each method activity, described in BPMN, is represented as a BPEL task, which is associated with services. Many services are invoked by the engine during the BPEL execution. The engine will control the execution of the flow, the state of process, the time spent in each task, saving important data for the process analysis. A method activity can be associated with one or more services of different types and a specific service can be used by various activities. This flexibility in their association makes it easier for the method engineer to define and improve the workflow by selecting available services. The services invoked by BPEL are called web services in an IT environment. [14] considers that web services are the main element of a SOA. It represents the business functionality and application logic available in conceptual business architecture to users, costumers and other services on an IT network.

Three types of web services available in the network to support the method:

1. Transformation Services: transformation between models;
2. Operation Services: operations performed over the models;
3. Software Engineering Services: execution of activities related to SE.

The *transformation services* make transformations between UsiXML models, passing one or many UsiXML models as input and receiving another UsiXML model as output. For instance, task model and domain model as input for the transformation into an abstract UI model. The *operation services* receive as input UsiXML models created or edited using the UsiXML tools (e.g., IdealXML for task and domain model, SketchiXML or GrafiXML for Concrete UI) in order to perform some of the main model management operators: match, compose, diff, modelGen, and merge. The *SE services* receive as input UsiXML models created or edited using the UsiXML tools in order to perform the following services: version control and traceability. These services handling various models are performed by accessing them from a common repository.

Method Deployment

Figure 4 represents the infrastructure adopted for the method execution, but it can be simplified or improved, considering the availability of resources. The idea is to make clear where the method is executed and how the models are manipulated. Once the method is defined in BPMN, it can be transformed in BPEL code using a tool that implements mapping rules. The BPEL engine will effectively execute it in the organization network. The execution of BPEL and the management of instances of this execution by the BPEL engine are called Orchestration. The organizational processes are executed by a set of available services, reusing what is already done and extending the current business rules.

The process orchestration, implemented by BPEL, makes a lot of calls to web services available in the network. The web services could be available in many points of the network and the BPEL Engine needs to call many web services on different places. But, it could be a problem in the future because the connections can become interlaced, thus increasing the possibility of service coupling. To solve this problem it is recommended to use an Enterprise Service Bus (ESB) to be an intermediary between the BPEL Engine and all web services available in the network. This technology is an implementation of a broker, an architectural pattern to structure distributed software systems with decoupled components that interact by remote service invocation. The ESB is responsible for coordinating communication, such as forwarding requests, also for transmitting results and exceptions. All communication intermediated by the Enterprise Service Bus are defined by Simple Object Access Protocol (SOAP), a XML based protocol that works over TCP/IP to support web service messages between servers and clients. The entire solution is XML based. The models are defined in UsiXML, which will be transported on the network in a XML message, to be processed by services exposed using XML signatures, to be orchestrate by a BPEL (a XML to describe the execution of business logic), defined by BPMN, whose metadata is in XML format.

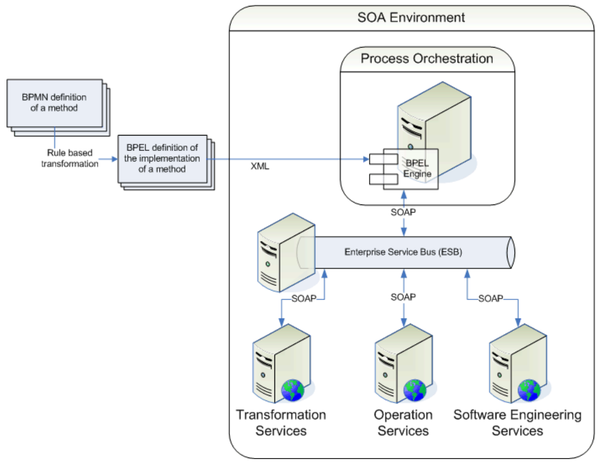

SOA Environment
Process Orchestration
BPMN definition of a method
Rule based transformation
BPEL definition of the implementation of a method
XML
BPEL Engine
SOAP
Enterprise Service Bus (ESB)
SOAP
SOAP
SOAP
Transformation Services
Operation Services
Software Engineering Services

Figure 4. The architecture proposed.

Method Execution

Methods are then executed according to the normal SOA principles of orchestration. Table 2 presents the services created for each method activity.

Activity	Type of Service	Service Interface
Task analysis	Software Eng.	controlModel(taskModel)
Class design	Transformation	transformDomainModel(classDiagram)
Analyze Context of Use	Operation	addContextModel(contextModel)
Review Requirements	Software Eng.	evaluateModels(evaluatedModels)
Abstract UI Prototyping	Software Eng.	controlModel(auiModel)
ConcreteUI Prototyping	Software Eng.	controlModel(cuiModel)
Generate Final UI	Transformation	generateFUI(language, cuiModel)

Table 2. Association of Method Activities and Services.

SUPPORT FOR COMPUTER-MEDIATED COMMUNICATION

Technically speaking, the implementation of services in a specific platform is called Web Services [13]. 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 [9], "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 [11], 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.

Figure 5 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.

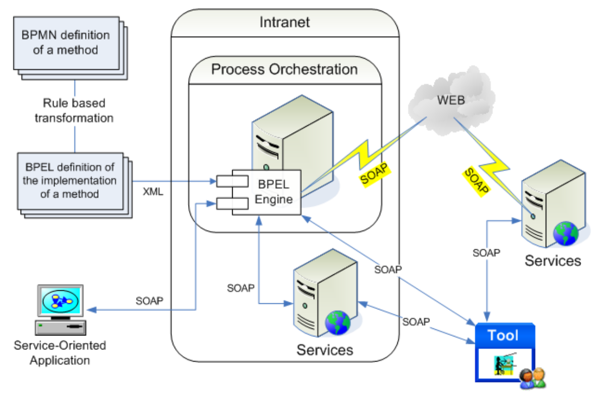
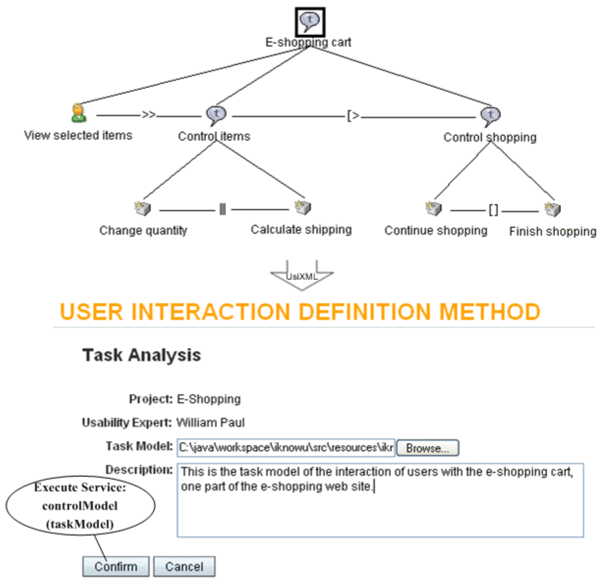

BPMN definition of a method
Rule based transformation
BPEL definition of the implementation of a method
XML
Intranet
Process Orchestration
BPEL Engine
SOAP
WEB
SOAP
Services
SOAP
SOAP
SOAP
Services
SOAP
Service-Oriented Application
SOAP
Tool

Figure 5. The SOA support for methods and tools.

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. We now 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, the usability expert uploaded the UsiXML task model file (Fig. 6). 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.



USER INTERACTION DEFINITION METHOD

Task Analysis

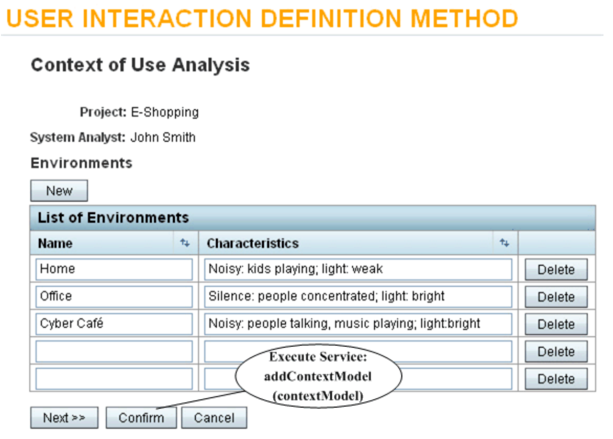
Project: E-Shopping
Usability Expert: William Paul
Task Model: C:\java\workspace\iknowu\src\resources\ikr Browse...
Description: This is the task model of the interaction of users with the e-shopping cart, one part of the e-shopping web site.

Execute Service: controlModel (taskModel)

Confirm Cancel

Figure 6. Task Analysis.

During the Analyze Context of Use activity of the inception phase, the usability expert can inform the data about the context of use (Figure 7). 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. During the Abstract UI Prototyping activity of the inception phase (Figure 6), 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, the system analyst can create the class diagram using any UML editor, upload the XMI file of the class diagram, and then request the generation of the UML class diagram into a UsiXML domain model (Figure 8).



USER INTERACTION DEFINITION METHOD

Context of Use Analysis

Project: E-Shopping
System Analyst: John Smith

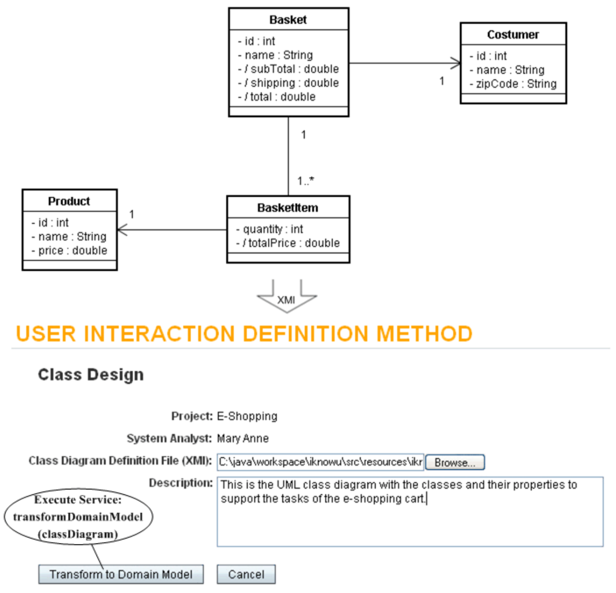
Environments

List of Environments

Name	Characteristics	
Home	Noisy: kids playing; light: weak	Delete
Office	Silence: people concentrated; light: bright	Delete
Cyber Café	Noisy: people talking, music playing; light:bright	Delete
		Delete
		Delete

Figure 7. Analyze Context of Use.

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. With this application, we reinforce consistency in applying the method by providing such services for specific transformations.



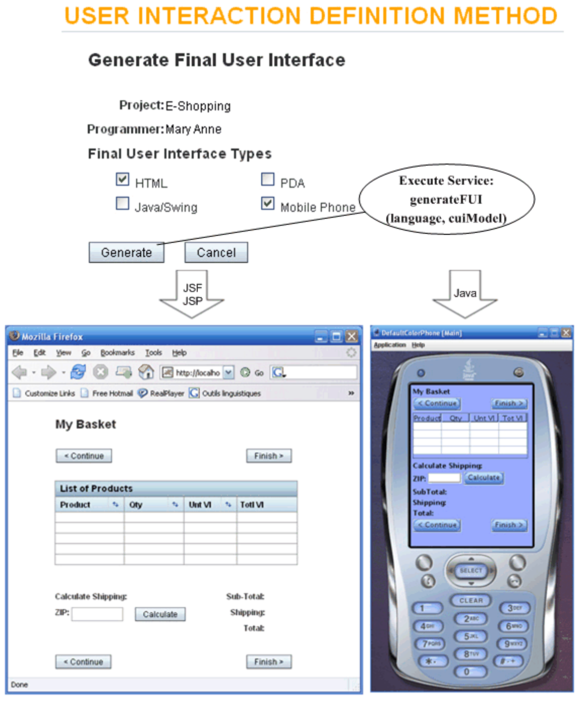
USER INTERACTION DEFINITION METHOD

Class Design

Project: E-Shopping
System Analyst: Mary Anne
Class Diagram Definition File (XMI): C:\java\workspace\iknowu\src\resources\ikr
Description: This is the UML class diagram with the classes and their properties to support the tasks of the e-shopping cart.

Figure 8. Class Design.

During the *Review Requirements* activity of the inception phase, 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, the usability expert creates the CUI model. 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, 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, the implementer selects the language to generate the UI. 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 (Fig. 9). Consider that this is an automatically generated UI, in which the designer will still incorporate the organization's style guide into it. For the moment, (X)HTML, Java, and XUL codes are automatically generated.



USER INTERACTION DEFINITION METHOD

Generate Final User Interface

Project: E-Shopping
Programmer: Mary Anne

Final User Interface Types

[✓ HTML] PDA
Java/Swing [✓ Mobile Phone]

Figure 9. 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.

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 services, 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.

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. Bias, Randolph G.; Mayhew, Deborah J. Cost-Justifying Usability: An Update for the Internet Age, Morgan Kaufmann Publishers Inc., San Francisco, CA. (2005).
4. Botterweck, Goetz and J. Felix Hampe. Capturing the Requirements for Multiple User Interfaces. 11th Australian Workshop on Requirements Engineering (AWRE'06), Adelaide, Australia, 2006
5. 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 (June 2003), pp. 289-308.
6. 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'2005, 48-60.
7. Constantine, Larry and Lockwood, L. Software for Use: A Practical Guide to Models and Methods of Usage-Centered Design. Addison-Wesley, Reading.
8. Blind reference
9. Erl, Thomas. Service-Oriented Architecture – Concepts, Technology, and Design. Prentice Hall, Crawfordsvile, Indiana (2005).
10. Göransson, Bengt; Gulliksen, Jan; Boivie, Inger. The usability design process - integrating user-centered systems design in the software development process. Software Process: Improvement and Practice 8(2): 111-131 (2003).
11. Harvey, Michael. Essential Business Process Modeling. O'Reilly Media (2005).
12. Jouault, Frédéric; Kurtev, Ivan. Transforming Models with ATL. Model Transformations in Practice (Workshop at MoDELS 2005), Montego Bay, Jamaica (2005).
13. Kreger, Heather. Web Services – Conceptual Architecture. IBM Software Group (2001).
14. Kruchten, Philippe. The Rational Unified Process - An Introduction. 2 ed. Addison-Wesley, New Jersey (2000).
15. 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", Rinton Press, Paramus, 2004, pp. 325-338.
16. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley, NY, 2004.
17. 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.
18. OMG, Business Process Modeling Notation Specification, 1.0, February, 2006.
19. OMG, Software Process Engineering Metamodel Specification, Jan. 2005.
20. Peltz, C. Web Services Orchestration and Choreography. IEEE Computer, Vol. 36, 10 (2003) 46-52.

21. Sadiq, Waqar; Racca, Felix. Business Services Orchestration - The Hypertier of Information Technology. Cambridge University Press, Cambridge (2003).
22. Schaffer, Eric. Institutionalization of Usability: A Step-by-Step Guide, Addison Wesley, (2004).
23. 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.
24. Soley, Richard. Model Driven Architecture. Available at: http://www.omg.org/mda/presentations.htm. OMG White Paper (2000).
25. Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005),* O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16-31.
26. Visual Paradigm. Business Process Visual Architect. Available at: http://www.visual-paradigm.com/product/bpva/. Accessed on March 30th, 2007.
27. 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