KnowiXML: A Knowledge-Based System Generating Multiple Abstract User Interfaces in USIXML

Elizabeth Furtado, Vasco Furtado, Kênia Soares Sousa
Universidade de Fortaleza (UNIFOR)
Av. Washington Soares, 1321 – 60.811-905
Fortaleza (CE), Brazil
e-mail: {elizabet, vasco, kenia}@unifor.br

Jean Vanderdonckt, Quentin Limbourg
Université catholique de Louvain
Place des Doyens, 1
B-1348 Louvain-la-Neuve, Belgium
{vanderdonckt, limbourg}@isys.ucl.ac.be

ABSTRACT
This research presents a multidisciplinary approach aimed at generating multiple Abstract User Interfaces (AUIs), which are adaptable for different kinds of users, performing different tasks, using specific devices in various physical environments. The UI generation framework, called IKnowU, is based on a unified process for interactive system design, which integrates Software Engineering (SE), and Human-Computer Interaction (HCI) best practices. This framework is supported by KnowiXML, a Knowledge-Based System (KBS) that facilitates the application of models and the allocation of appropriate visual elements during the generation of AUIs. These AUIs are generated by using problem solving methods studied in Artificial Intelligence (AI). Design knowledge encoded in KnowiXML uniformly manipulates models and UI specifications through the use of an User Interface Description Language (UIDL).

ACM Classification Keywords
D.2.1 [Software Engineering]: Requirements/Specifications – elicitation methods. D.2.2 [Software Engineering]: Design Tools and Techniques – user interfaces. H.1.2 [Models and Principles]: User/Machine Systems. H.5.2 [Information Interfaces and Presentation]: User Interfaces –Prototyping, Graphical User Interfaces (GUI). I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods. I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search.

General terms
Design, Languages, Human Factors.

Author Keywords
Abstract user interface, design knowledge, expert system, knowledge base, problem solving methods.

INTRODUCTION
As users' needs become more varying, interactive application modeling and development become more complex.

Therefore, we consider the generation of multiple UIs, which leads to the need of various HCI models (e.g., task, user, environment, platform), practices, and professionals (designers, HCI experts) throughout the Software Development Process (SDP). As a result, it becomes necessary to integrate HCI models, practices, and activities with those from SE. This integration might generate more complex SDPs that lead to communication problems between the HCI and SE teams, making it more difficult to attend users' needs. Among the existing solutions that have been developed, we believe that besides using formal languages, there must be a step-by-step definition of integrated processes supported by computational tools, and on the early-use of prototypes based on Abstract UIs (AUIs) definitions, useful for user acceptance tests.

This research work presents an approach that, besides integrating SE and HCI, also integrates AI. Related to SE and HCI, we defined a Unified Process, called UPi [18], for interactive system design, which integrates SE and HCI best practices in order to generate adaptable UIs for different kinds of users, performing different tasks, using specific devices in various contexts.

Related to AI, our proposal is to develop a KBS, in which we can formalize the generation of multiple AUIs based on conceptual specifications. As a result, the designer would not need to have a high level of specialized knowledge. The KBS will be a module of the UI generation framework, which we are also proposing in this paper. The KBS will use problem-solving methods studied in AI, as well as an extensible UI conceptual specification, that is an UIDL, which will be defined using ontologies. The extensibility of the specification leads to the generation of UIs for multiple platforms that allow professionals to perform their activities more effectively when information and services are made available instantly.

This paper is organized as follows: the second section presents related work; the third section introduces the UIDL used to define the UI models in this paper; the fourth section presents the process and the framework; the fifth section presents the framework components; the sixth section presents an application of the process using the framework; and the seventh section concludes this work.

RELATED WORK

In this section, we report on initiatives that integrate HCI and SE techniques while establishing requirements and works related to the generation of multiple UIs.

Requirements Definition with HCI and SE Techniques

Functional requirements can be analyzed and documented using different techniques or artifacts. We analyzed four of them that are used to understand user's goals and tasks and to design the UIs: scenario, context of use, use case, and task analysis. A *scenario* is an informal narrative description [3] describing the human activities being performed in an environment. It has two important advantages: it is easy for the stakeholders to write stories, and it allows developers and HCI experts to concentrate on understanding what people do and the contexts with which the humans operate.

The *context of use* model defines aspects related to the system to be developed, which are: the platform, the environment, and users. This model associates the platform with an environment, anchoring the description to the physical world, besides taking into account the variations of the tasks in order to preserve usability [2]. This model reflects one way to represent textual scenarios in a defined manner, organized in the three entities mentioned previously.

The *Use Case Model* represents a set of flows of events that can happen as a result of the user interaction with the system [9]. Essential use cases [4] are a textual structured and platform-independent definition of use cases, organized in user intentions and system responsibilities.

A task specifies a set of activities the user and/or the interactive system does in order to achieve the user's goals identified. In *task models*, it is possible to represent the task decomposition (that involves breaking a complex task – either system's or user's - down into subtasks, until the lowest level task), and the structural and temporal relations as an ordering among tasks.

All of these sorts of techniques should be combined to help people to imagine what they could have on a system. This is due to the fact that some requirements are difficult to find or they are unconscious (for instance, people can be used to it or maybe they do not have a clue to see the overall picture). In addition to that, each technique is more appropriate to one kind of modeling than to another.

Although use cases are also focused on users' goals, their emphasis is on a user-system interaction rather than the user's task itself [15]. This happens because they contain certain assumptions about the UI and the kind of interaction to be designed, including the technology device the user interacts with. Essential use cases [4] try to avoid these assumptions, by defining only what the user role (not the actor) is responsible for (her/his intentions) and what the system should do.

Phillips [14] suggests the use of tabular representation of use cases in order to describe the flow of events, and the use of UI element clusters, which can be used as references to the UI prototype. Tabular use cases separate user and system actions. Lauesen [11] argues that separating users' and systems' actions as early as in requirements may be a barrier for future decisions in the project. He suggests the use of task descriptions, which specify what the users and the system shall do, not dividing the work between them.

Generation of Multiple UIs

We now compare works generating UIs from conceptual models. Some of them consider multiple AUIs, others focus on the generation of Final UIs (FUI). The Cameleon Reference Framework for multi-target UIs [2] uses three types of models: i) *ontological models* are meta-models independent from any domain and interactive system; ii) *archetypal models* depend on the domain and interactive system; and iii) *observed models* are executable models that support adaptation at run-time. The process also uses three classes of models (e.g. domain, context of use, and adaptation models) that may be ontological, archetypal or observed. Domain models cover domain concepts and user tasks; context of use models describe the user, platform, and environment [2]; and adaptation models cover evolution and transition of the UI.

UIML [1] is a UIDL for multiple devices emphasizing the separation of concerns of an interactive system in a platform-independent way. The framework for building multi-platform UIs has three models: i) a task model that is independent of the physical model; ii) a family model that describes the arrangement of the UI for each family (e.g. desktop, PDA, WAP); and iii) a platform-specific UI that uses widgets associated with the platform.

XIML [16] is a universal representation for UIs that can support multiple UIs at design time and at run-time. It is an organized collection of interface elements that are categorized into five components: task, domain, user, dialog, and presentation. The first three are in the contextual and abstract levels while the last two are in the implementation and concrete levels. It also supports relationship definition and statement for linking any component and any element.

AUIT [8] is a device-independent mark-up language useful to build adaptable UIs that augments current JSP web server implementations. It generates a thin-client UI adapted for the user, their current task context, and their display device characteristics. An AUIT screen specification contains device-independent screen element tags. At run-time, the AUIT tags are processed by JSPs that look for a corresponding tag library class, which performs adaptations and generates appropriate output for the user's device.

The adaptive task modeling [6] proposes two specification techniques. The first one is an adaptation mechanism for task models and the second one is the process that makes a transformation of an abstract interaction model into a specific UI representation. The adapted task model consists of a sequential description, in

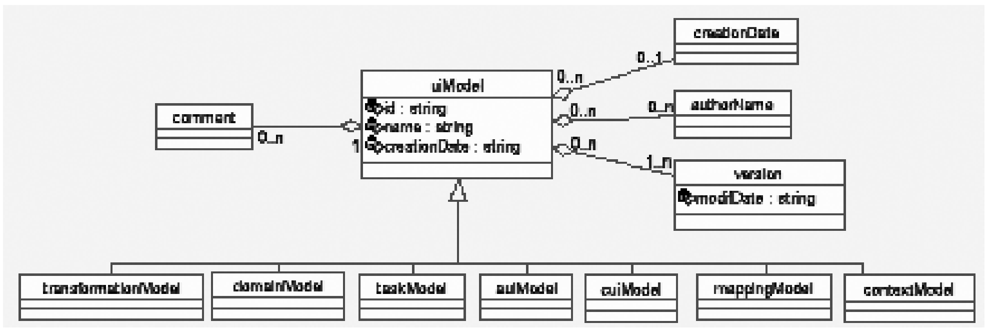

Figure 1. USIXML Model Collection.

which the operations can be performed to fulfill the whole task, with constraints for specific devices. The transformation process starts with a XML-based UI description that is mapped into a XML-based device dependent UI model based on information about specific features of devices. The next step is to create a XSL-based UI description based on design rules, then the specific UI is generated by XSL transformation.

UIML and XIML can be both considered as a UIDL. In this paper, we use the USer Interface eXtensible Markup Language (USIXML) as the UIDL, which will be explained in the next section.

THE UNDERLYING USER INTERFACE LANGUAGE

USIXML was chosen because it is equipped with a collection of basic UI models (Fig. 1) [12]: task, domain, AUI, Concrete UI (CUI), context, transformation, and mapping. These models can be mapped together according to a mapping model. All models share a common syntax based on semantics defined in terms of UML class diagrams that have been transformed into XML Schemas to guide UI specifications.

UiModel is the topmost superclass containing common features shared by all component models of a UI. A uiModel may consist of a list of component model in any order and any number. TransformationModel allows a collection transformations among the UI models. DomainModel is a description of the classes of objects manipulated by a user while interacting with a system. TaskModel is a model describing the interactive task as viewed by the end user interacting with the system. A task model represents a de-composition of tasks into sub-tasks linked with task relationships. AUIModel defines interaction spaces and a navigation scheme among interaction spaces and selects abstract objects that are independent of any modality of interaction (e.g., graphical, vocal, speech, video, virtual reality) or of any context of use. CUIModel concretizes an AUI for a given context of use into concrete objects so as to define widgets layout and interface navigation. MappingModel is a model containing a series of related mappings among models or elements of models. A mapping model serves to gather a set of inter-model relationships that are semantically related. ContextModel is a model describing the context of use in which an end user is carrying out an interactive task with a specific computing platform in a given surrounding environment.

THE PROCESS AND THE FRAMEWORK

After comparing related approaches, we decided to develop a framework, called IKnowU, to semi-automatically generate usable UIs, concerned with how to provide a robust solution for the software industry. With this goal in mind, we consider the requirements established in [16]: i) define the models based on robust representation, such as CTT [13] and UML; ii) use a representation that is in sync with the needs of the software industry (e.g., portability); iii) propose a process that is compatible with acceptable SE processes (UPi is based on RUP [10]); iv) use a widely implemented foundation technology, such as XML; and v) apply the environment in a pilot program to verify its feasibility.

Fig. 2 represents the relationship among the process activities and artifacts, organized in three columns: the first one makes reference to the UPi disciplines, in which professionals execute activities (second column) to generate artifacts (third column). The main process activities and their order of execution are based on [2,6]. We focus on two UPi disciplines: Requirements, and Analysis and Design, which are directly related to the generation of AUIs. The Implementation discipline will be detailed in a future work, in which we will focus on the generation of CUIs and FUIs.

In the requirements discipline, the system analyst and the HCI expert elicit users' needs and translate such needs into system functionality, focusing on the context of use and usability requirements. We propose that the analysis of users' needs, the definition and refinement of the system are made through the definition of conceptual models, which are: task and context of use model. These models are useful to represent users' tasks, personal characteristics, environment, and platform. In addition, we use a domain model, which is useful to specify allocated entities to perform tasks, represented by the UML class diagram.

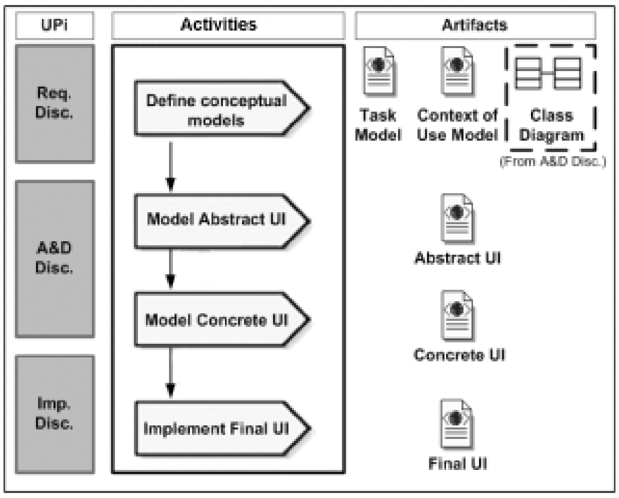

Figure 2. The Process disciplines, activities and artifacts

In the analysis and design discipline, the software architect, the designer, the UI designer, and the HCI expert design the system architecture as a solution to develop the system, model the AUI and CUI, and refine the architecture to design system components and the database. The formal definition of abstract and concrete UIs are useful to facilitate the generation of AUIs based on information in the conceptual models and the generation of CUIs based on AUIs.

With the definition of UPi, we envisioned the need to create IKnowU to support SE and HCI professionals in generating interactive systems for multiple contexts of use. IKnowU is currently in the analysis and design phase according to UPi. The framework functionality, exemplified with a UML use case diagram in Fig. 3, depicts the automation of the first two activities in Fig. 2.

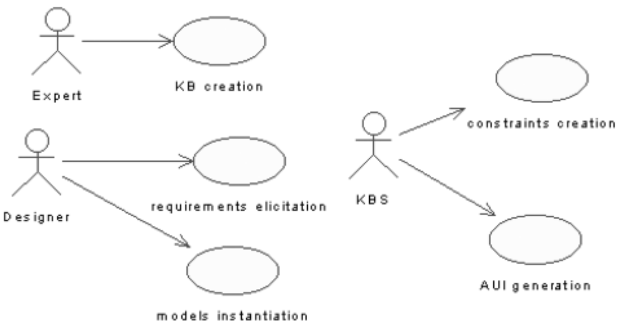

Figure 3. The framework functionality.

The domain expert and the designer are professionals, and KnowiXML is a module of IKnowU. The domain expert is responsible for creating the KB by providing a set of guidelines and informing transformation rules. Therefore, this professional must be an expert in the HCI domain in order to perform those tasks (the domain is not making reference to the knowledge on the domain of the system under development). The designer elicits usability requirements and instantiates conceptual models. KnowiXML creates constraints concerning abstract objects in order to generate the AUI by analyzing users' preferences and constraints, abstract objects constraints, and selecting abstract objects based on this analysis.

STEP 1 – KNOWLEDGE BASE CREATION

As we have mentioned previously, the domain expert provides guidelines and transformation rules. Guidelines are usability rules that represent users' preferences and constraints, as well as correction actions (fixes) concerning the system they want to use (Fig. 4). Guidelines are associated to abstract objects and serve as the basis for the definition of usability requirements, which are also related to abstract objects.

One example of transformation rules are task rules that represent the relationship among tasks in the task model and abstract objects in the AUI. For instance, tasks that request information from the user are associated to input abstract objects. Even though, the framework includes an initial KB, the domain expert can enhance it with more information.

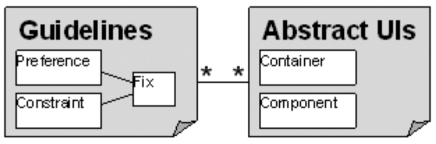

Figure 4. Relationship among guidelines and abstract UIs

STEP 2 – REQUIREMENTS ELICITATION

The designer elicits usability requirements with the user by using interview techniques, considering different user profiles. These requirements are elicited from a set of preferences and constraints that must be addressed in the abstract UI. For instance, preferences are: 'maintain system consistency', 'provide feedback', 'provide help', and constraints are: 'provide help *only* upon request'. In some cases, it is necessary to use actions (fixes) to correct situations when certain preferences and constraints are conflicting. For instance, concerning the preference and constraint related to help, the fix would be 'offer user explicit control when providing help'. Because of the association of guidelines with abstract objects, the usability requirements (preferences, constraints or fixes) represent actions upon the abstract UI that include or exclude abstract objects (abstract containers or abstract individual components in USIXML [12]).

Step 3 – Models Instantiation

The designer instantiates users' usability requirements and the following models: use case, task, domain, and context of use, which are required to generate the AUI.

Step 4 – Constraints Creation

When the designer requests the generation of the AUI, KnowiXML starts an analysis of a set of task rules against information from instantiated models (such as the task and context of use model) in order to create a set of AUI constraints concerning the allocation of objects in the AUI. Such constraints allow the definition of which (e.g., input, output, control, navigation) and how many abstract objects will be allocated in a certain abstract container.

Step 5 – Abstract User Interface Generation
As a result of analyzing users' preferences, constraints, and AUI constraints, KnowiXML generates the AUI. The abstract objects are in accordance to usability requirements and to the task, context of use, and domain models. The accordance to the task model is achieved with the use of task rules that result in the allocation of abstract objects on the AUI in order to facilitate the users to perform their tasks.

Step 6 – Progressive generation of CUI and FUI
Once the AUI is generated, it is supposed to be a UI that remains independent of any modality and computing platform as it is expressed only in terms of abstract containers and abstract individual components. Therefore, the ultimate step in the process is to progressively reify the AUI into a CUI once a target computing platform has been selected. This CUI will be in turn the source for reification into a Final UI (FUI) once a particular language of the platform has been identified. These two reifications are ensured by TransformiXML, a transformation engine that supports multi-path development of UIs based on USIXML. In principle, a CUI is independent from any programming or markup language. But in practice, this reification depends on the availability of USIXML code generators [17] and interpreters.

GENERATION OF MULTIPLE ABSTRACT UIS
Now, we detail how IKnowU functions in terms of its components, as depicted in Fig. 5 according to the UML component diagram. It consists of three main components:

Component 1: the Ontology Editor
The HCI expert instantiates conceptual models allowing the generation of UIs using ontologies. Ontologies define concepts related to each model, specified in XML, according to the USIXML formalism [12]. For instance, name, type, and frequency are concepts for the task model. The specification of these models is made using the tool called *Protégé* [7], which is an ontology editor and a knowledge-base editor. Protégé is also an open-source Java tool that provides an extensible architecture for the creation of customized applications and interacts with XML. Protégé will allow experts to create the ontologies for the models and designers to instantiate the models using forms.

The rules that represent users' preferences and constraints are defined using the *Rules Plug-in*, which allows the expert to define rules using concepts from the models and to compile such rules into Java classes using Java Embedded Object Production System (JEOPS) [5]. The JEOPS adds forward chaining, first-order production rules to Java through a set of classes designed to provide this language with some kind of declarative programming. With that, the development of intelligent applications, such as software agents or expert systems is facilitated [5].

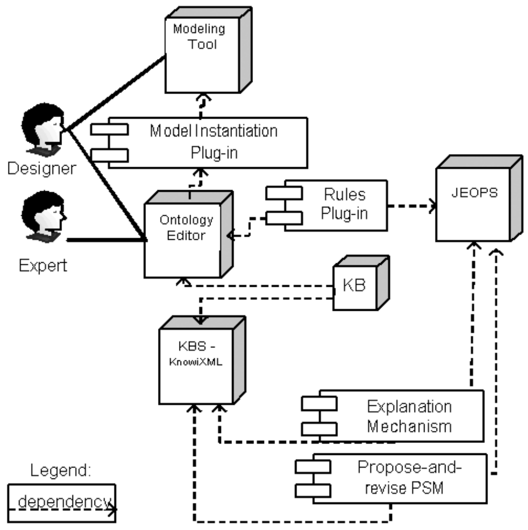

Figure 5. IKnowU Components.

Component 2: the Modeling Tool
Designers can use *Protégé* to instantiate the models or they can use tools that they are accustomed to (such as IBM Rational Rose for the use case model or CTTE [13] for the task model), and then use such tools to translate these models into XML, which will be instantiated in a predefined ontology in *Protégé* [7]. As a result, we expect to provide tools to enable the execution of an integrated SDP that considers artifacts and activities from SE and HCI.

Component 3: the Knowledge-based System (KBS)
We consider that the task to generate AUIs based on conceptual specifications involves problems related to the configuration of multiple AUIs, such as which abstract object should be on a UI to achieve a good level of usability. Therefore, the proposed KBS implements a Problem-Solving Method (PSM) used to generate AUIs based on configuration propositions, and revisions of such propositions when they violate specific constraints. This method is called propose-and-revise.

This configuration method initially processes the preferences, which are elicited from users' usability requirements. These preferences allocate abstract objects on the AUI. After that, AUI constraints are also processed in order to guarantee that all the allocated abstract objects are in conformance to users' preferences and constraints.

IKnowU contains a Knowledge Base (KB) and an inference engine. Therefore, IKnowU is responsible for the multiple AUI generation, through the analysis of the instantiated models in Protégé, and through the execution of rules by the JEOPS inference engine.

This declarative definition for problem-solving facilitates the knowledge acquisition process and allows the exploration of such knowledge through, for instance, explanation about the system reasoning process to solve problems [19]. That is, the resulting AUI presented to the designer can be negotiated using an explanation mechanism that provides information concerning the steps taken to achieve the solution and other possible ways to reach different results. An explanation mechanism was developed and implemented in Java to provide adaptive messages according to the expert knowledge level about KBS decisions executing a design PSM [19].

AN EXAMPLE

In order to explain the details of generating AUIs, we present a scenario for an interactive system and the steps taken by the domain expert (HCI expert), the KBS user (the designer) and the interactive system users (professors and students). In our selected scenario, a student, who is taking a distance learning course in order to accommodate a busy working schedule, wants to access the course materials in different situations: (i) from the *desktop computer* at home during the night and weekends; (ii) from the *notebook* at the office during breaks and lunch hours or at a hotel during business trips; (iii) from the *Palm Top* during business trips while waiting for a flight in the airport.

Step 1 – Knowledge Base Creation

Some examples of guidelines are: facilitate undo tasks, provide progress indication, facilitate object selection, etc.

Step 2 – Requirements Elicitation

In our scenario, one user preference is:

1. To have quick access.

As the user does not require quick access to all system tasks, only to the most frequent ones, which must be accessible at any time while interacting with the system, two user constraints associated to the preference above are:

2. Only to have quick access to tasks with high frequency of execution.
3. Tasks with high frequency of execution can not be interrupted.

The fix that can solve the differences among these preferences and constraints is:

4. Offer quick access only to tasks with high frequency of execution throughout the system.

Step 3 – Models Instantiation

The designer is responsible for instantiating the tasks, domain and context model using *Protégé* (Tables 1-4). In the task model, we have three subtasks: (i) the user selects one theme from the course to view a list of course materials; (ii) the user can view data of any selected material; and finally, (iii) the user can actually view the course material (table 1). In table 2, the first task can be repeated many times and enables the execution of the second task, and the other two tasks are optional, while the second task enables the execution of the third one. Tables 1-4 involve model attributes as they are typically featured in USIXML models. In the domain model, we define the attributes and methods for the class "Material", which has 4 attributes and 3 methods.

task			
id	01	02	03
name	select theme	view course material detail	view course material
type	interactive	interactive	interactive
frequency	5 (very frequent)	4 (very frequent)	4 (very frequent)
importance	5 (very important)	5 (very important)	5 (very frequent)
structurationLevel	5 (very structured)	5 (very structured)	5 (very structured)
complexityLevel	2 (easy)	2 (easy)	2 (easy)
userAction	go	view	view
taskItem	operation	operation	operation

Table 1. The Task Model.

taskRelationship			
unaryRelationship			
id	01	02	03
name	finiteIteration	optional	optional
binaryRelationship			
targetId	01	02	
sourceId	02	03	
name	enabling	enabling	

Table 2. The Relationships among Tasks.

domainClass				
id	01			
name	material			
Attribute				
name	title	sentBy	situation	file
attributeDataType	string	string	string	string
attributeCardMin	1 (mandatory)	1 (mandatory)	1 (mandatory)	0 (not mandatory)
attributeCardMax	1 (mandatory)	1 (mandatory)	1 (mandatory)	0 (not mandatory)
attributeDomainCharacterization	-	-	-	-
Method				
name	include()	update()	delete()	

Table 3. The Domain Model.

Environment				
type	-	-	-	-
id	01	02	03	04
name	office	home	hotel	airport
isNoisy	true	false	false	true
lightingLevel	high	high	high	high
isStressing	false	false	false	true
userStereotype				
id	01	02		
stereotypeName	expert student	novice student		
taskExperience	5 (high)	5 (high)		
systemExperience	5 (high)	3 (low)		
device Experience	5 (high)	2 (low)		
taskMotivation	5 (high)	4 (medium)		
browserUA				
id	01			
name	Internet Explorer			
browserName	IE			
browserVersion	6.0			
isFramesCapable	true			
isJavaAppletEnabled	true			
isJavaScriptEnabled	true			

Table 4. The Context Model.

In the context model, we specify four different environments in which the user might interact with the system, two different user profiles, and characteristics of the browser the user might be interacting with.

Step 4 – Constraints creation

Some examples of task rules are:

1. If the unary relationship of a task is 'finite iteration' and the unary relationship of a related task is 'optional', then use a splittable abstract container.
2. If the type of the task is 'interactive' and the domain model associated to this task has attributes that are

mandatory, then use the 'output' abstract individual component.
3. If the binary relationship of a task is 'enabling', then use the 'navigation' abstract individual component.
4. If the type of the task is 'interactive' and the domain model associated to this task has methods, then use the 'control' abstract individual component.

Example of AUI constraints generated by KnowiXML through the analysis of the rules above and of the instantiated models (in tables 1, 2, and 3) are:

1. The AUI must have 2 splittable abstract container (task id 1 and 2, 1 and 3).
2. The AUI must have 3 'output' abstract individual component (task id 1, 2, and 3).
3. The AUI must have 2 'navigation' abstract individual component (task id 1 and 2).
4. The AUI must have 3 'control' abstract individual component (task id 1, 2, and 3).

If there is any unconformity of abstract objects in relation to users' references and constraints, certain actions (e.g. include new abstract IOs, substitute allocated abstract IOs to new ones, etc.) are performed to correct unconformities until all the allocated abstract IOs are in conformance to usability preferences and constraints.

Step 5 – Abstract User Interface Generation

As a result of analyzing the user preferences and the constraints generated from the task rules, the PSM processing results in the abstract UI, depicted in Table 5.

The PSM guarantees that the abstract objects are in accordance to users' usability requirements and to the task model. The accordance to the task model is achieved with the use of task rules that result in the allocation of abstract objects on the AUI in order to facilitate the users to perform their tasks. In the AUI for the scenario in which the user is interacting with a web browser on a desktop, there are two containers, one showing the list of themes (task 1) and the other one showing the course material detail data (task 2); there are nine individual components, 4 output, 2 navigation, and 3 control objects.

Step 6 – Progressive generation of CUI and FUI

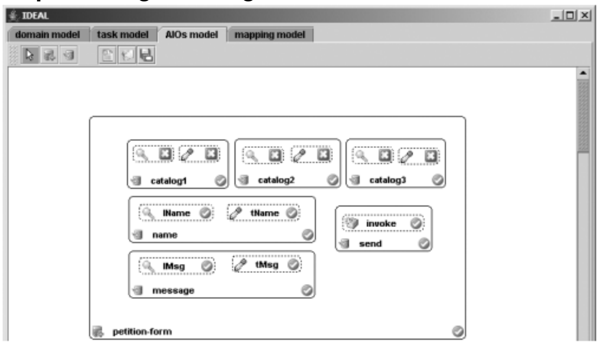

Figure 6. Import of the AUI in IDEALXML.

Once the AUI has been generated, its corresponding USIXML specifications can be imported into IDEALXML (Fig. 6), which is the integrated development environment from which the transformation engine is called (TransformiXML) to turn the AUI into a CUI, into a FUI.

CONCLUSION

Our main goal is to save design and development time by automating the generation of UI models and assure consistency among different platforms with the application of such models. By noticing the similarities among different approaches, we can assure that our approach and framework has a wide range of possibilities to be applied and validated by many research groups. With the use of a KB, we intend to process rules that concern device characteristics, user preferences, contextual issues, among other aspects in order to provide designers and developers a framework that dynamically organizes and personalizes the UI and also that learns with experience.

We hope to develop interactive systems that are easy to learn and use, therefore, helping users in performing their daily tasks in an efficient manner. This work focused on the knowledge acquisition process, which is performed semi-automatically with the models that compose the knowledge base, thus, guiding interviews with the HCI expert. This process, however, has demonstrated to be difficult and time-consuming. A perspective that we are investigating is to define how to create tools that automate part of this process. This is possible by defining filters that translate conceptual specifications in PSM knowledge rules (e.g., constraints, preferences, and solutions). This way, interviews with designers can be useful to validate the acquired knowledge, which would considerably decrease the knowledge acquisition time, besides, maintaining coherence with what was specified on the functional, usability requirements, and on user profiles.

Since we are currently working on generating the ontologies in Protégé and screens for IKnowU, we intend to perform usability, applicability, and reliability evaluations to be presented it in a future paper.

REFERENCES

1. Ali, M.F., Pérez-Quiñones, M.A., and Abrams, M. Building Multi-Platform User Interfaces with UIML. In A. Seffah & H. Javahery (eds.), *Multiple User Interfaces*. John Wiley & Sons, New York, 2004, 95–117.
2. 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 15*, 3 (2003) 289–308.
3. Carroll, J. Introduction to the Special issue on "scenario-Based Systems Development". *Interacting with Computers 13*, 1 (2000) 41–42.
4. Constantine, L. and Lockwood, L. *Software for Use: A Practical Guide to Models and Methods of Usage-Centered Design*. Addison-Wesley, Reading, 1999.
5. Figueira, C. and Ramalho, G. JEOPS – The Java

Embedded Object Production System. In M. Monard, J. Sichman (eds.). *Proc. of 7th Ibero-American Conference on AI* (Atibaia, November 19-22, 2000). Lecture Notes in Artificial Intelligence, Vol. 1952. Springer-Verlag, Berlin, 2000, 53–62.

6. Forbrig, P., Dittmar, A., and Müller, A. Adaptive Task Modelling: From Formal Models to XML Representations. In A. Seffah & H. Javahery (eds.). *Multiple User Interfaces*. John Wiley, New York, 2004, 171–192.
7. Gennari, J.H., Musen, M.A., Fergerson, R.W., Grosso, W.E., Crubézy, M., Eriksson, H., Noy, N.F., and Tu, S.W. *The Evolution of Protégé: An Environment for Knowledge-Based Systems Development*. SMI Report Number: SMI-2002-0943, Stanford, 2002. Accessible at http://smi.stanford.edu/pubs/SMI_Abstracts/SMI2002-0943.html
8. Grundy, J. and Zou, W. AUIT: Adaptable User Interface Technology, with Extended Java Server Pages. In A. Seffah & H. Javahery (eds.). *Multiple User Interfaces*. John Wiley & Sons, New York, 2004, 149–167.
9. Jacobson, I., Christersson, M., Jonsson, P., Overgaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Reading, 1992.
10. Kruchten, P. Ahlqvist, S., and Bylund, S. User Interface Design in the Rational Unified Process. *Object Modeling and User Interface Design*. Addison-Wesley, 2001.
11. Lauesen, S. Task & Support - Task Descriptions as Functional Requirements. *Proc. of AWRE'2001*. Centre for Advanced Software Engineering Research, Univ. of New South Wales, Sydney, 2001, pp. 83-91. Accessible at http://www.itu.dk/people/slauesen/Papers/TasksSupportAWRE.pdf
12. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M. and Trevisan, D. USIXML: A User Interface Description Language for Context-Sensitive User Interfaces. *Proc. of the AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" UIXML'04* (Gallipoli, 25 May 2004). EDM-Luc (2004), 55–62.
13. Paternò, F. *Model-based Design and Evaluation of Interactive Applications*. Springer-Verlag, Berlin, 1999.
14. Phillips, C. and Kemp, E. In Support of User Interface Design in the Rational Unified Process. *Proc. of the Third Australasian User Interface Conf*. 2002, 21–27.
15. Preece, J., Rogers, Y., and Sharp, H. *Interaction Design-Beyond Human-Computer Interaction*. John Wiley & Sons, New York, 2002.
16. Puerta, A., Eisenstein, J. XIML: A Multiple User Interface Representation Framework for Industry. In A. Seffah & H. Javahery (eds.). *Multiple User Interfaces*. John Wiley & Sons, New York, 2004, 119–148.
17. Schlee, M. and Vanderdonckt, J., Generative Programming of Graphical User Interfaces. In M.F. Costabile (ed.). *Proc. of AVI'2004* (Gallipoli, May 25-28, 2004). ACM Press, New York, 2004, 403–406.
18. Sousa, K. and Furtado, E. An Approach to Integrate HCI and SE in Requirements Engineering. In M.B. Harning & J. Vanderdonckt (eds.). *Proc. of Interact'2003 Workshop on Closing the Gaps: Software Engineering and Human-Computer Interaction* (Zürich, 1 September 2003). 81–88.
19. Vasconcelos, E., Pinheiro, V., and Furtado, V. Mining Data and Providing Explanation to improve Learning in Geosimulation. *Proc. of Int. Conf. on Intelligent Tutoring Systems ITS'2004* (Maceió, 2004), to appear.