Extending UsiXML to Support User-Aware Interfaces

Ricardo Tesoriero, Jean Vanderdonckt

Extending UsiXML to support User-aware Interfaces

Ricardo Tesoriero[1,2] and Jean Vanderdonckt[1]

[1] Université catholique de Louvain
Place des Doyens, 1 (B-1348)
Louvain-la-Neuve, Belgium
[2] University of Castilla-La Mancha
Av. España S/N. Campus Universitario de Albacete (02071)
Albacete, Spain
ricardo.tesoriero@uclm.es, jean.vanderdonckt@uclouvain.be

Abstract. Mobile and portable devices require the definition of new user interfaces (UI) capable of reducing the level of attention required by users to operate the applications they run to improve the calmness of them. To carry out this task, the next generation of UIs should be able to capture information from the context and act accordingly. This work defines an extension to the UsiXML methodology that specifies how the information on the user is modeled and used to customize the UI. The extension is defined vertically through the methodology, affecting all layers of the methodology. In the Tasks & Concepts layer, we define the user environment of the application, where roles and individuals are characterized to represent different user situations. In the Abstract UI layer, we relate groups of these individuals to abstract interaction objects. Thus, user situations are linked to the abstract model of the UI. In the Concrete UI layer, we specify how the information on the user is acquired and how it is related to the concrete components of the UI. This work also presents how to apply the proposed extensions to a case of study. Finally, it discusses the advantages of using this approach to model user-aware applications.

1 Introduction

In 1994, Mark Weiser introduced the notion of Calm Technology in [16]. The aim of Calm Technology is to reduce the "excitement" of information overload by letting the user select what information should be placed at the center of their attention and what information should be peripheral. A way to support Calm Technology is the use of the context information to reduce users' work.

According to [11], context-aware applications [4] are characterized by a hierarchical feature space model. At the top level, there is a distinction between human factors, in the widest sense, and the physical environment. Both, the human factors and the physical environment, define three categories of features each. While, the human factors are defined in terms of features related to (a) the

information on the user (knowledge of habits, emotional state, bio-physiological conditions, ...); (b) the user's social environment (co-location of others, social interaction, group dynamics, ...); and (c) the user's tasks (spontaneous activity, engaged tasks, general goals, ...); the physical environment are defined in terms of features related to (a) the location (absolute position, relative position, co-location,...), (b) the infrastructure (surrounding resources for computation, communication, task performance, ...) and (c) physical conditions (noise, light, pressure, ...).

This article explores the development of multi-modal UIs that are affected by human factors. Concretely, it focuses on those features related to the information on the user (i.e. emotional state, bio-physiological conditions, skills, experience, ...) and the user's tasks (i.e. , defined by the role played in the society).

The proposal is based on an extension to the UsiXML [14] methodology based on the social model of CAUCE methodology defined in [13]. While UsiXML provides a model-based approach to design multi-modal UIs based on the Cameleon reference framework [2], the social model provide designers with the ability express how user features affect the application UI.

The paper is organized as follows. In Section 2, we present the most relevant related works on the development of multi-modal UIs. Then, the UsiXML extension to support user awareness is exposed in Section 3. Afterwards, the extension is applied to a case of study in Section 4. Finally, in Section 5 we expose conclusions and future works.

2 Related Work

Teallach tool and method [1] exploit three models: a task model, a domain model as a class diagram, and a presentation model both at logical and physical levels. Teallach enables designers to start building a UI from any model and maps concepts from different models one to each other. However, the tool does not support the development of context-aware UIs. Moreover, from the user modeling point of view, Teallach does not support any type of user profile nor awareness.

The approach exposed in [3] describes a toolkit of interactors, which are designed to develop UIs that handle both input and output using multiple mechanisms. The toolkit supports adaptation for a change in the resources available to the widgets, or a change in the context the platform is situated in. However, the approach does not support any model to capture user information from external sources that are not directly related to the platform or the widget contents defined in the interface.

XIML [5] is a more general UIDL than UIML as it can specify any type of model, any model element, and relationships between. The predefined models and relationships can be expanded to fit a particular context of use. The term context is interpreted as the platform of the application is running and not the application is adapter to such platform, instead of the information that affects,

or is relevant, to the application. No other issue related to the context awareness is taken into account by the model.

SeescoaXML [8] supports the production of UIs for multiple platforms and the run-time migration of the full UI. However, the development does not take into account the definition of the user-aware behavior of the UI.

The CTTE (ConcurTaskTrees Environment)[9] is a development environment to support the development and analysis of task models for interactive systems. Based on these models, the TERESA (Transformation Environment for inteRactivE Systems representAtions) [10] produce different UIs for multiple computing platforms by refining a general task model. Thus, various presentation and dialog techniques are used to map the refinements into XHTML code adapted for each platform such as the Web, the PocketPC, and mobile phones.

Although CTT (the language used to describe the task model of the applications developed using TERESA) supports the role definition and the definition of different task models for each role, the task and role concepts are so coupled that the definition of similar interfaces derive in different models (one for each role). Besides, this approach does not take into account the attributes of role definitions, although CTT allows designers to assign values to standard attributes. Thus, the definition of custom attributes is not supported directly.

RIML [12] consists of an XML-based language that combines features of several existing markup languages (e.g., XForms, SMIL) in a XHTML language profile. This language is used to transform any RIML-compliant document into multiple target languages suitable for visual or vocal browsers on mobile devices. Although RIML provides the ability to specify multi-modal UI, RIML is focused on the view of the application independently of the context it is being executed. Therefore, no context or user awareness is taken into account.

3 The user-aware UsiXML extension

UsiXML defines a development process based on the Cameleon Reference Framework [2] to build multi-device interactive applications. The development process is divided into four layers.

The Task & Concepts (T&C) layer describes users' tasks to be carried out, and the domain-oriented concepts required to perform these tasks.

The Abstract User Interface (AUI) defines abstract containers and individual components [7] (two forms of Abstract Interaction Objects [15]) 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 modality. Thus, an AUI is considered as an abstraction of a CUI with respect to interaction modality.

The Concrete User Interface (CUI) concretizes an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) [15]. It also defines the widget layouts and the interface navigation. It abstracts a FUI into a UI definition that is independent of any computing platform. Therefore, a CUI can

also be considered as a reification of an AUI at the upper level, and an abstraction of the FUI with respect to the platform.

Finally, the Final User Interface (FUI) is the operational 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 interactive development environment).

These layers are defined by the UIModel depicted in Figure 1. Each model that is part of the UIModel represents an aspect of the UI to be developed. Thus, the proposal defines the User Model to represent how the features of the user that affect the UI.

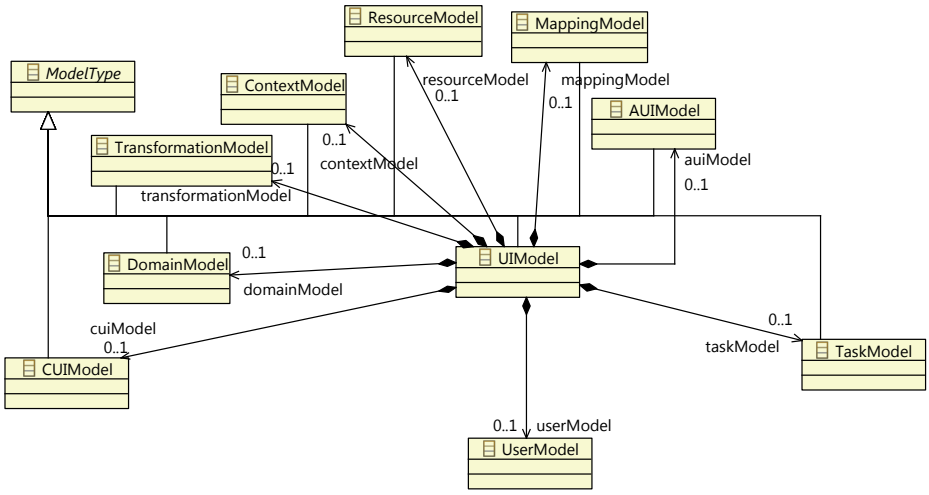

Fig. 1. UIModel models defining the UI

3.1 The Task & Concepts layer extension

The User Model is the core of the user-awareness modeling. It is defined in the T&C layer of the UsiXML methodology. The goal of this model is the representation of the user features that affect the UI.

To carry out this task, the model represents this information in two levels of abstraction: the user feature level and the user profile level.

- The user feature level defines the features of the user that affect the UI. Thus, designers are able to represent the user features that are relevant to the application domain providing flexibility when describing user profiles.
- The user profile level is based on the information defined at the user feature level. It characterizes the features according to runtime situations. Thus, designers are able to identity different groups of individuals that share the same characteristics.

Both, the user feature level and the user profile level are rooted in the UserModel metaclass, as depicted in Figure 2. It represents the user characteristics that affect the UI at both levels of abstraction.

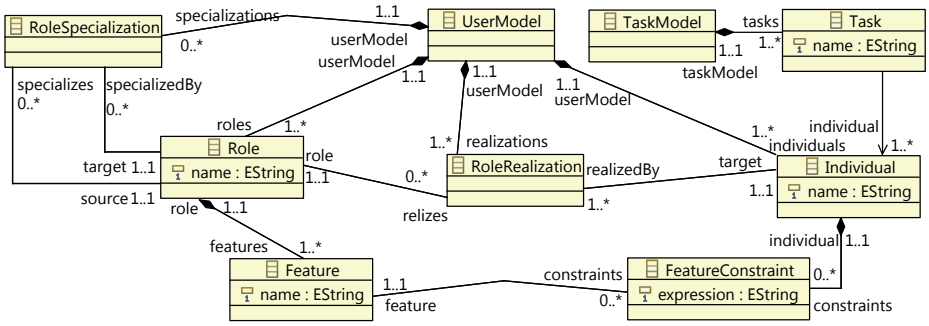

Fig. 2. The UserModel metamodel

The user feature level defines the user characteristics in terms of *Role*s and *Feature*s. We have defined *Role* as a set of user *Feature*s where a *Feature* represents an attribute related to the user playing this role in the system. For instance, a Patient *Role* may define the age, cardiac pulse, temperature and glucose level *Feature*s.

In order to represent *Role*s that have common *Feature*s, we have defined the *RoleSpecialization* relationship between *Role*s. It is a directional relationship defined by the *source* and the *target* attributes. The semantic meaning of the *RoleSpecialization* relationship is defined as follows:

Let A and B be instances of the *Role* metaclass; Let $FwoS(R)$ be the function that takes the *Role* R as parameter and returns the set of *Feature*s defined by R *Role* **without taking into account the *RoleSpecialization* relationship**. Let $F(R)$ be the function that takes the *Role* R as parameter and returns the set of *Feature*s defined by R *Role* **taking into account the *RoleSpecialization* relationship**. Finally, let $S(A, B)$ be the *RoleSpecialization* relationship that defines A as the *source* and B the *target* of the relationship.

Then,

$$F(B) = FwoS(B)$$
$$F(A) = FwoS(B) \cup FwoS(A)$$

The user profile level defines user characteristics in terms of *Individual*s and *FeatureConstraint*s

Users are characterized according to the *Feature*s defined by the *Role*s they play in the system. The characterization at this level is defined by the *FeatureConstraint*s metaclass that is related to an *Individual* playing a *Role*. Thus, the *RoleRealization* metaclass defines the relationship between the *Role* and the *Individual*. Then, the *Individual* defines a set of *FeatureConstraint*s that are related to the *Feature*s defined by the *Role* it plays in the system.

For instance, following the example exposed on the user feature level, the Patient *Role* may be realized by the *aPatientWithFever Individual* that defines a *FeatureConstraint* where the *temperature Feature* is higher than 38 Celsius Degrees.

Thus, the same user may be a *aPatientWithFever* or not, according to the body *temperature* of the user.

Finally, the *UserModel* is related to the *TaskModel* by the means of reflecting how the characteristics defined by the *UserModel* are propagated to the UI in the T&C layer of the UsiXML methodology. Each *Task* defined in the *TaskModel* is affected by an *Individual* that describes the user situation in which the *Task* is performed. Therefore, in order to perform a *Task*, all the *FeatureConstraint*s defined by the *Individual* that is related to the task must be satisfied.

3.2 The AUI layer extension

The AUIModel is part of the AUI layer of the UsiXML methodology. Although the user awareness extension does not affect the AUIModel definition directly, it introduces some modifications in the AUI layer by the means of the definition of new inter-model relationships in the MappingModel.

These relationships are established between Abstract Interaction Objects (*AIO*s) and *Individual*s. However, they do not affect Abstract Individual Components *AIC*s in the same way they affect Abstract Containers *AC*s.

On the one hand, *AC*s are directly affected by *Individual*s. On the other hand, *AIC*s are affected through *Facet*s that are affected by *Individual*s.

The Figure 3 shows the extensions of the MappingModel and how it affects the AUIModel.

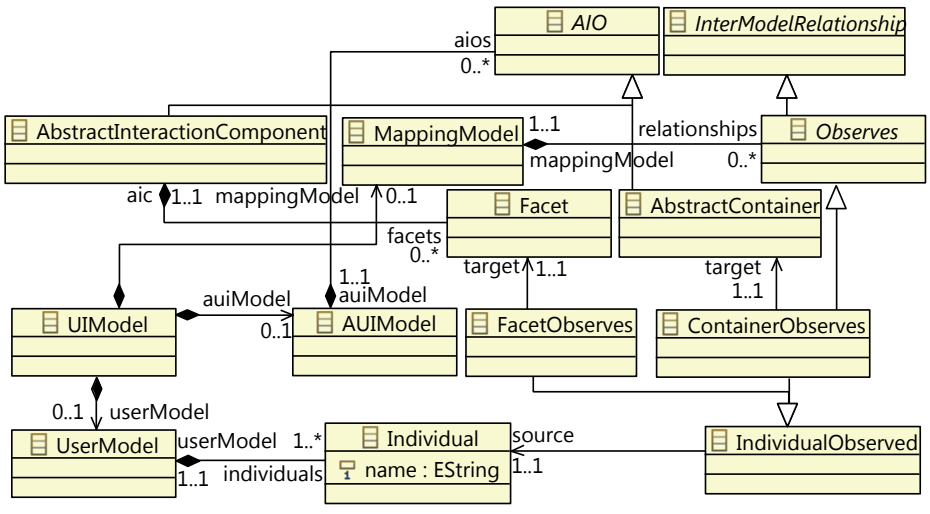

Fig. 3. MappingModel extensions in the AUI layer

Let suppose that an *Individual* is related to an *AC*. If all the *FeatureConstraint*s of the *Individual* are satisfied, then the *AC* is "active", "enabled" or "available". Otherwise, the *AC* is "inactive", "disabled" or "unavailable".

*AIC*s define *Facets*. These *Facet*s are manipulated by the *FeatureConstraint*s defined by the *Individual* they are attached to.

Let suppose that an *Individual* is related to a *Facet*. If all the *FeatureConstraint*s of the *Individual* are satisfied, then the *Facet* is "active", "enabled" or "available". Otherwise, the *Facet* is "inactive", "disabled" or "unavailable".

Therefore, the behavior of the UI is affected by profiles defined in the Tasks & Concepts layer of the UsiXML methodology. This relationship is defined by the *FacetObserves* and *ContainerObserves* submetaclasses of the *Observes* metaclass, which belongs to the MappingModel.

Thus, as an *AC* or a *Facet* can "observe" many *Indivuduals*, a conflict among *FeatureConstraint*s may arise. Therefore, we enable the *Facet* or *AC* when any of the *Individuals* match the user state.

Depending on the development path, these relationships can be derived from the TaskModel and the *IsExecutedIn* inter-model relationships defined in the MappingModel using transformation rules defined in the TransformationModel.

3.3 The CUI layer extension

The extension to support the user awareness in the CUI layer is depicted in Figure 4.

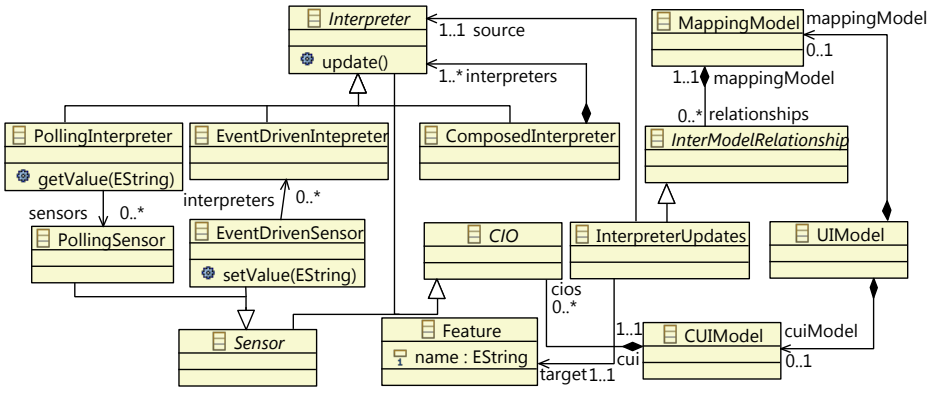

Fig. 4. Relating Sensors and Interpreters to Features

The metamodel is divided into two parts:

- The CUI model extension
- The mapping model extension

The CUI extension goal is the description of the system is aware of the information on the user. To carry out this task, we have introduced the *Sensor* and the *Interpreter* entities that are in charge of capturing, interpreting and providing information to be processed by the system.

The information can be perceived in different ways according to the sensor it is used to capture it. Therefore, we have defined two types of *Sensor*s: *EventDrivenSensor*s and *PollingSensor*s. The information that is perceived by sensors should be interpreted accordingly. To perform this task, we have defined *Interpreter*s. *Interpreter*s are in charge of transforming information that comes from *Sensor*s into information that is compatible with the user environment definition.

We have defined three types of *Interpreter*s: the *PollingInterpreter* that deals with information coming from a *PollingSensor*, the *EventDrivenInterpreter* that is in charge of interpreting information from an *EventDrivenSensor*, and the *ComposedInterpreter* that is responsible for the interpretation of information coming from several sensors. The *Interpreter* hierarchy is an implementation of the Composite design pattern [6].

The information processed by the CUI extension is propagated through the rest of the model by the mapping model extension. Therefore, the connection between the *CIO*s and the rest of the models is performed through the *interpreterUpdates* submetaclass of the *intermodelRelationship* defined by the MappingModel. This relationship is used to define the relationship between *Feature*s defined in the user environment model and the *Interpreter*s.

Thus, *Individual*s are notified of the changes of the information on the user through *FeatureConstraint*s.

3.4 The transformation process and the FUI

In this section, we point out some issues related to the transformation process that takes place between the abstract user interface *AUI* and the concrete user interface *CUI*. The way the information captured by the models is translated to source code depends on the *AIO* we are dealing with.

If we are dealing with *abstractContainer*s, the translation of an *Individual* that matches the state of the user usually results in the modification of a property in the *CIO* that represents it. For instance, the *visible*, *enabled* or *opaque* property of the *CIO* is set to `true`.

However, the mechanism used by *AIC*s is not the same because *Individual*s are related to *Facet*s. Therefore, some aspects of the *AIC*s may match some *Individual*s and some of them may not.

Suppose that an *AIC* defines the following *Facet*s: *Input*, *Output* and *Navigation*. In addition, each *Facet* is related to different *Individual*s: $I_1 \leftrightarrow input$, $I_2 \leftrightarrow output$ and $I_3 \leftrightarrow navigation$.

Thus, if I_1 is the unique *Individual* that matches the user environment, then the *AIC* may be represented by a `TextField`. However, if I_2 is the unique *Individual* that matches the user environment, then the *AIC* may be represented by a

`Label`. Finally, if I_3 is the unique *Individual* that matches the user environment, then the *AIC* may be represented by a `Button` or a `Link`.

Therefore, the same *AIC* can be turned into different *CIO*s, at runtime, according to the user state. The Figure 5 represents the possible results of the transformation, according to the interface specification taking into account the user environment.

The situation is solved using different mappings for the same *AIO*. Thus, the user state defines which component is available according to the user environment.

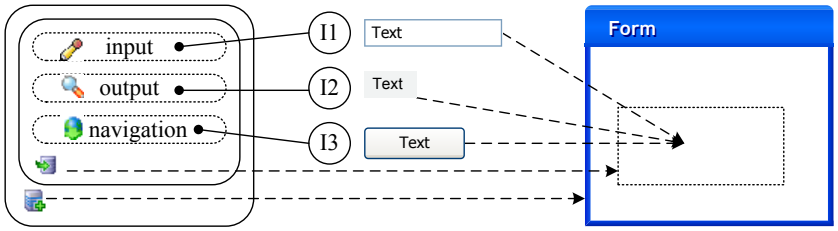

Fig. 5. Possible results on the Final User Interface

4 Case of study

The application that will serve as case of study is the "Healthy Menu". The goal of the application is the presentation of dishes according to the role and the biophysical state of the user. Thus, views and controls are customized to provide the user with Calm UIs.

The explanation is focused on issues related to the definition of the user environment and the relationships between this environment and the UI. Other issues are left behind for the sake of clarity.

4.1 The user model

The application defines five roles: Users, Patients, Doctors, Nurses and Visitors.

The Users of the application are identified by the *idNumber*, *roleName* and *userName* features.

As all other roles are specializations of the User role, these roles inherit User features. Besides these features, Patients are characterized by the *age*, *temperature* and *glucose* features; Doctors by the *specialty*; Nurses by the *experience* (in years); and Visitors by the patient they are visiting (the *patientId* feature).

As we will focus on the Patient role, we have defined five *Individual*s for this role (aPatient, NormalPatient, PatientWithHyperGlycemia, PatientWithHipoGlycemia and PatientWithFever); and only one for each remaining role (anUser, aPatient, aNurse, aDoctor and aVisitor).

Each *Individual* is defined by a set of *FeatureConstraint*s. For instance, we say that a Patient is normal, if the body temperature is between 36.5 and 38 Celsius degrees and the Sugar level in blood is between 70 and 110 millimoles/liter.

The Figure 6 shows the user model of the application. Rounded rectangles represent Roles, the dashed rectangles defined inside them represent features, the specialization of roles is represented by a triangle-ended arrow pointing to the role that is being specialized, circles represent instances and the rectangles on drawn on the right of these circles represent feature constraints.

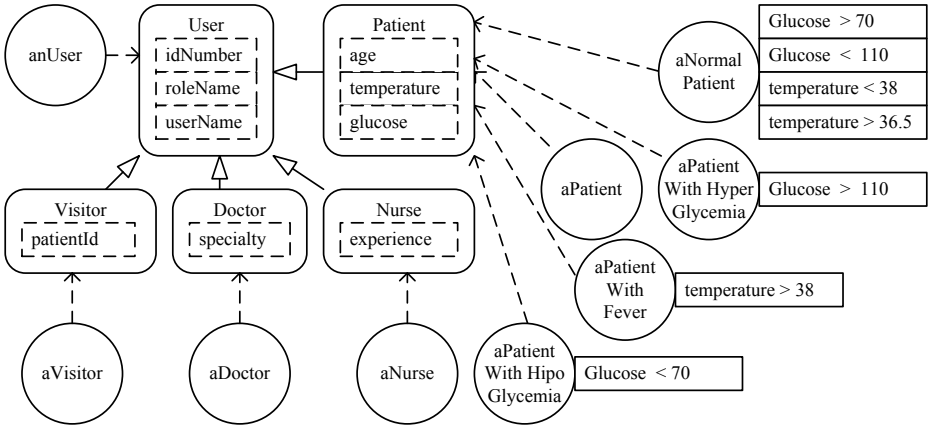

Fig. 6. Healthy screen user model

4.2 The task model

The simplified version of the task model for the Healthy screen application is defined in Figure 7.

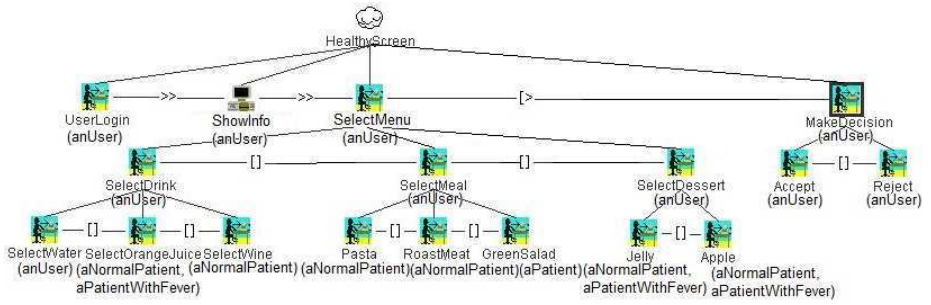

Fig. 7. Healthy screen task model

The *UserLogin* task allows the system to identify the user that is using the application. The information that is retrieved from the system is exposed to the user by the means of the *ShowInfo* system task. Then, the user is able to choose the menu (*SelectMenu* task). To improve task model readability, the menu is divided into three types of selection defined under the *SelectDrink*, *SelectMeal*, *SelectDesset* task. Each task represents a possible menu option. For instance, *SelectWater* (to select water as a drink) is available to all users. However, *SelectWine* (to select wine as a drink) is available for normal patients only. It is also possible to assign more than one individual for each task. Therefore, it is available if any of the involved individuals match the "user state".

4.3 The AUI model

Once the user model was defined, we define the *AUI* model. The AUI model defines the UI without taking into account its modality.

Although the AUI layer introduced two new types of mappings to describe the relationship between *AIO*s and *Individual*s, the extension does not introduce new elements to AUIModel. Therefore, AUIs are described in the same way they are described traditionally.

The Figure 8 depicts a partial view of the *AUI* model focused on the Patient *Role*. On the left, we show an overview of the containers that are related to the roles defined in the user environment. On the right, we show a detailed view of the Patient role AUI.

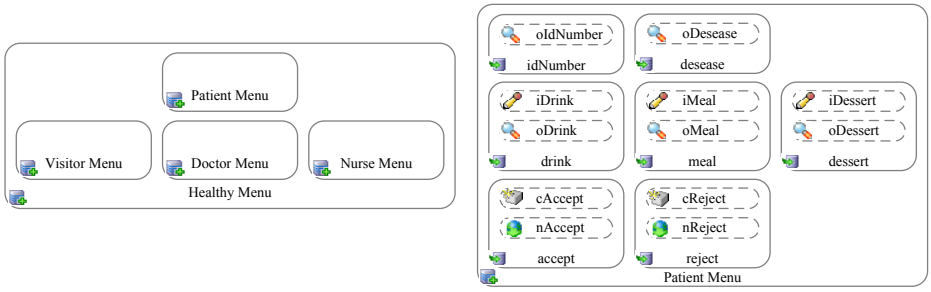

Fig. 8. Partial Healthy Menu AUI model

Then, these *AUI*s are linked to *Individual*s to reflect the modifications in the "user state" accordingly.

The Table 1 shows the *Observes* relationships that relate *AC*s and *Facet*s to *Individual*s . These relationships are defined in the mapping model and described in terms of the $O_c(I, AC)$ and the $O_f(I, F)$ functions. While the O_c function represents the *ContainerObserves* relationship, which relates an *Individual* I to an *AbstarctContainer* AC, the O_f function represents the *FacetObserves* relationship, which relates an *Individual* I to a *Facet* F.

AUI mappings	
Containers	
$O_c(aPatient, PatientMenu)$	
$O_c(aNurse, NurseMenu)$	
$O_c(aDoctor, DoctorMenu)$	
$O_c(aVisitor, VisitorMenu)$	
Facets	
$O_f(anUser, nAccept)$	$O_f(anUser, oIdNumber)$
$O_f(anUser, cAccept)$	$O_f(anUser, oDesease)$
$O_f(anUser, nReject)$	
$O_f(anUser, cReject)$	
$O_f(aPatientWithHipoGlycemia, oDrink)$	$O_f(aPatientWithFever, iDrink)$
$O_f(aPatientWithHipoGlycemia, oMeal)$	$O_f(aPatientWithFever, oMeal)$
$O_f(aPatientWithHipoGlycemia, oDessert)$	$O_f(aPatientWithFever, iDessert)$
$O_f(aPatientWithHyperGlycemia, oDrink)$	$O_f(aNormalPatient, iDrink)$
$O_f(aPatientWithHyperGlycemia, oMeal)$	$O_f(aNormalPatient, iMeal)$
$O_f(aPatientWithHyperGlycemia, oDessert)$	$O_f(aNormalPatient, iDessert)$
$O_f(aPatientWithFever, oDrink)$	$O_f(aNormalPatient, oDrink)$
$O_f(aPatientWithFever, oMeal)$	$O_f(aNormalPatient, oMeal)$
$O_f(aNormalPatient, oDessert)$	

Table 1. AUI mappings

To conclude this section, we analyze the *drink AIC* defined in Figure 8 to show an example of how to define the AUI and how to link it to the *Individuals* defined in the user model.

From the AUI perspective, the *drink AIC* represents the component that is in charge of providing users with the information about drinks. This information may be input and output information (if the user is able to select the drink, i.e. *aNormalPatient*), or may be output information only (if the user is not able to select the drink, i.e. *aPatientWhyHyperGlycemia*). Therefore, two Facets (*oDrink* and *iDrink*) were defined for this *AIC*.

From the Mapping perspective, lets analyze the individual *aPatientWithHipoGlycemia* and its relationship with the *drink AIC*. The *O_f(aPatient With Hipo Glycemia, oDrink)* is the only relationship between individuals and facets of the *drink AIC*. Therefore, the *drink AIC* is an output control. However, if we analyze the *aNormalPatient* individual, we see that the *O_f(aNormal Patient, iDrink)* and the *O_f(aNormal Patient, oDrink)* relationships define an input/output relationship.

4.4 The CUI model

The *CUI* definition is based on two sensors: *PollingGlucoseSensor* and *Polling TemperatureSensor*. Both of them are instances of the *PollingSensor* metaclass. It also defines two instances of the *PollingInterpreter* metaclass, the *GlucoseInterpreter* and *TemperatureInterpreter*.

Table 2 shows the mapping between the Patient role features and sensors. The function $R_p(S_p, I_p)$ represents the relationship between a *PollingSensor* (S_p) and a *PollingInterpreter*(I_p). The function $U(I, F)$ represents an instance of the *InterpreterUpdates* submetaclass of *Updates*, which relates an *Interpreter* to a *Feature*.

CUI mappings
Sensors
$R_p(PollingGlucoseSensor, GlucoseInterpreter)$
$R_p(PollingTemperatureSensor, TemperatureInterpreter)$
Features
$U(GlucoseInterpreter, Patient.glucose)$
$U(TemperatureInterpreter, Patient.temperature)$

Table 2. CUI mappings

To illustrate the use of the elements defined in the CUIModel, we will expose how the *temperature* feature is related to the environment.

The temperature is captured from the environment through a *Polling Temperature Sensor*. To poll the sensor, we have to link it to a *Polling Temperature Interpreter* in charge of requesting the sensor status and propagating it to the rest of the system.

Finally, the *Polling Temperature Interpreter* is linked to the *temperature* feature of the Patient role to propagate the changes from the environment.

4.5 The HealthyScreen FUI

The result of the *FUI* of a Patient is depicted in Figure 9. The first capture shows the UI for a Patient whose vital parameters are considered normal, the second one shows the UI for a Patient whose body temperature is above normal, and the third capture shows the UI for patients affected with HipoGlycemia or HyperGlycemia.

Finally, we show how the elements defined by the user awareness extension work together in this case of study. To see the effect of the user awareness, we set the initial state of the individual as *aNormal Patient*. Then, we modify the *temperature* feature of the individual, and we set it to 39 Celsius degrees.

This change in the temperature is captured by the *Polling Temperature Interpreter* that is constantly polling the *Polling Temperature Sensor* to be aware of the changes in the environment (in this case the user). Once the interpreter captures the information, it is propagated to the *temperature* feature defined by the Patient role. All features are linked to the features constraints that reference them. Thus, the chance is captured by the individual that is defined by these constraints. If all the feature constraints that are defined by the individual are satisfied then the individual is able to propagate this information through the

Fig. 9. Healthy Menu GUIs

IndividualObserves mappings to the AUI. Consequently, the UI is aware of the changes produced by the change on the temperature level.

5 Conclusions and Future Work

This work exposes a model-based approach to develop user-aware multi-platform and multi-modal UIs based on the UsiXML methodology. It encourages the separation of the user modeling from the application domain to improve the model reuse during the development of UIs.

The approach embraces all steps of the application. It means that covers from conceptual modeling of the user environment to the specification of the sensing infrastructure to support the different user profiles dynamically.

In the "Tasks & Concepts" step of the methodology, we introduced two levels to define custom characterizations of the users. While the first level allows designers to specify the user features that are taken into account by the application, the second one allows designers to quantify these characteristics in order to characterize a group of individuals that have common characteristics.

As consequence, designers are able to specify customized user characteristics instead of standard characteristics that are difficult to interpret because of their general nature. Besides, it provides designers the ability to characterize different groups of individuals that define the same characteristics, and so the user characterization can be easily reused.

This separation also allows the definition of static and dynamic characteristics at the same time in the same space of definition.

Finally, another advantage of using the UsiXML methodology is the separation of the definition of concepts and tasks from the definition of UIs. Thus, the

characterization of users can be modified without having to modify the abstract user interface model, and vice versa.

As future work, we are actually working in the definition of an extension of the user awareness in order to model the social awareness of the user interfaces. The social awareness allows the UI to be aware not only of the user is operating it; it makes the UI be aware of other users that are part of the system. Thus, we cover the description of the social environment of context aware applications.

Another issue we have considered as part of future works is the inclusion of the location awareness as part of the UI specification to cover other context aware characteristics of the UI, such as the infrastructure environment, user position, etc.

Finally, we are also working on the definition of a common feature-based framework allowing designers to express characteristics that are related to the combination of the social and location features of context-aware UIs, such as the co-location.

References

1. Barclay, P.J., Griffiths, T., McKirdy, J., Kennedy, J.B., Cooper, R., Paton, N.W., Gray, P.D.: Teallach - a flexible user-interface development environment for object database applications. J. Vis. Lang. Comput 14(1), 47–77 (2003)
2. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting with Computers 15(3), 289–308 (2003)
3. Crease, M., Gray, P.D., Brewster, S.A.: A toolkit of mechanism and context independent widgets. In: Proceedings of DSV-IS. pp. 121–133 (2000)
4. Dey, A.K.: Understanding and using context. Personal and Ubiquitous Computing 5, 4–7 (2001)
5. Eisenstein, J., Vanderdonckt, J., Puerta, A.R.: Applying model-based techniques to the development of uis for mobile computers. In: IUI. pp. 69–76 (2001)
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading/MA (1995)
7. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: USIXML: A language supporting multi-path development of user interfaces. In: Bastide, R., Palanque, P.A., Roth, J. (eds.) EHCI/DS-VIS. LNCS, vol. 3425, pp. 200–220. Springer (2004)
8. Luyten, K., Laerhoven, T.V., Coninx, K., Reeth, F.V.: Runtime transformations for modal independent user interface migration. Interacting with Computers 15(3) (2003)
9. Mori, G., Paternò, F., Santoro, C.: CTTE: Support for developing and analyzing task models for interactive system design. IEEE Trans. on Soft. Eng. 28, 797–813 (2002)
10. Paternò, F., Santoro, C., Mäntyjärvi, J., Mori, G., Sansone, S.: Authoring pervasive multimodal user interfaces. Int. J. Web Eng. Technology 4(2) (2008)
11. Schmidt, A., Beigl, M., Gellersen, H.W.: There is more to context than location. Computers & Graphics 23(6), 893–901 (1999)

12. Spriestersbach, A., Ziegert, T., Grassel, G., Wasmund, M., and, G.D.: A single source authoring language to enhance the access from mobile devices to web enterprise applications. In: WWW2003 Developers Day Mobile Web Track. p. (not printed). Springer Verlag, Heidelberg etc., Germany (2003)
13. Tesoriero, R.: CAUCE: Model-driven Development of context-aware applications for ubiquitous computing environments. Ph.D. thesis, University of Castilla-La Mancha (December 2009)
14. Vanderdonckt, J.: A MDA-compliant environment for developing user interfaces of information systems. In: Proc. of 17 th Conf. on Advanced Information Systems Engineering CAiSE'05. pp. 13–17. Springer-Verlag (2005)
15. Vanderdonckt, J.M., Bodart, F.: Encapsulating knowledge for intelligent automatic interaction objects selection. In: Ashlund, S., Mullet, K., Henderson, A., Hollnagel, E., White, T. (eds.) Proc. of the Conf. on Human Factors in computing systems. pp. 424–429. ACM Press, New York (April 1993)
16. Weiser, M., Brown, J.S.: The coming age of calm technology. In: Denning, P.J., Metcalfe, R.M. (eds.) Beyond Calculation: The Next Fifty Years of Computing, pp. 75–85. Copernicus (1997)