Enriching UsiXML language to support Awareness requirements

Jose Figueroa-Martinez\textsuperscript{a,*}, Víctor López-Jaquero\textsuperscript{b}, Francisco Luis Gutiérrez Vela\textsuperscript{a}, Pascual González\textsuperscript{b}

\textsuperscript{a}Universidad de Granada, C/ Cuesta del Hospicio s/n, 18071, Granada, Spain  
\textsuperscript{b}Universidad de Castilla-La Mancha, CP 02071, Albacete, Spain

Abstract

Awareness support in model-driven architecture approaches is almost nonexistent. Although it is a required feature for the development of collaborative applications, the consideration of awareness in development methodologies and tools has been hindered by the lack of model-driven oriented conceptual models. This work presents an extension to the user interface description language UsiXML for describing generic awareness support in software development, with special focus on requirements level, and keeping a traceable path throughout the development stages. UsiXML describes multimodal and multicontextual user interfaces by following a model-driven approach to software development, and one of its purposes is to enable the development of highly interactive user interfaces, where awareness plays an important role. Furthermore, a template to gather awareness information requirements is provided to help designers explore and describe them in the early design stages.

Keywords: Awareness, model-driven user interface development, requirements specification

*Principal Corresponding author

Email addresses: jfigueroa@ugr.es (Jose Figueroa-Martinez), victor@dsi.uclm.es (Víctor López-Jaquero), fgutierrez@ugr.es (Francisco Luis Gutiérrez Vela), Pascual.Gonzalez@uclm.es (Pascual González)

URL: buhoz.net/jose (Jose Figueroa-Martinez)
1. Introduction

Generally speaking, awareness is “the knowledge of what is going on” [1], although the Groupware community is more familiar with the concept of Workspace Awareness as “the up-to-moment understanding of another person’s interaction with a shared workspace” [2].

Awareness has been identified as a required feature of Groupware [2], and as essential for enabling effective interaction in dynamic environments [1]. This is not only true for human behavior, but also for computer systems, as awareness improves interaction in both real and virtual environments and supports collaborative work.

The development of complex interactive systems requires special attention to the user interface (UI), where interactive processes take place. The model-driven approach to UI development [3] has been used to better integrate requirements, design and implementation. The sophistication of this approach has been recognized through the creation of a W3C group [4] to promote the standardization of model-based UI design.

Despite the importance of awareness and its increasing use in software methodologies and development tools, none of the UI markup languages available support awareness [5].

Model-driven development aims to build software using a set of models to which a transformation process is applied. Instead of manually coding domain, tasks, or UI, as is done in traditional software development, they are modeled and then transformed in order to generate the final software.

The use of the Model-driven approach in UI design is supported by an underlying UI description language (UIDL) that provides a way to represent the models that are required for specifying a UI. The most recent review on the UIDLs available can be found in [6]. Undoubtedly, one of the most active UIDLs is UsiXML [7], based on the availability of tools and its community.

UsiXML is an XML based markup language that supports the description of multi-context and multi-platform UIs. It supports the specification of the UIs at different levels of abstraction, thus fostering the separation of concerns in UI design. UsiXML development tools can generate specialized code for different target platforms using the UsiXML specification.

Like other UIDLs, UsiXML enables the context of use for the designed UI to be described. However, it does not support either the specification of awareness requirements or awareness itself, even though these awareness features are essential to the specification of any collaborative system [2].
In this work we present an extension to UsiXML language for awareness support, which integrates a set of models for the representation of any kind of Awareness information and requirements. This extension represents an improvement to the UsiXML’s methodology for developing interactive, dynamic and collaborative systems following the model-driven approach.

2. Related Work

In order to integrate a new type of requirement into a software methodology or description language, the main components and their relationships must be well identified and defined. To integrate awareness as a special information requirement we must know what awareness is and how to describe it. The Situation Awareness theory [1] is probably the most suitable one for describing awareness in general and the challenges inherent in awareness support, such as privacy, information overload and stress, among others.

Some seminal works on awareness can be found in the Groupware community, such as the theory of Workspace Awareness [2], which defines a way to support awareness in shared workspaces, and the works on Activity Awareness, such as [8], which describes different awareness facets as part of team collaboration. Furthermore, some other facets of awareness have been identified [9], such as social-awareness or task-specific awareness. These works help developers identify some of the most important aspects of Groupware development, such as awareness information requirements.

In [10] the awareness information structure is specified as a component of awareness support. This work provided the foundations for using awareness in model-driven development (MDD).

Tesoriero [11] presented a UsiXML extension to support user-aware UIs to improve model transformation processes and support more dynamic context scenarios. However, this was the only proposal to add some kind of awareness support to UsiXML until the extension proposed in [12], which presented the first version of a UsiXML extension to add generic awareness support capabilities to this language.

Modeling awareness as a requirement for collaborative systems has also been discussed in [13]. However, although it is aimed at modeling requirements for collaborative systems, it does not support either design stages or the MDD.
3. UsiXML: a method and a language for UI design

UsiXML is both a UIDL and a methodology. It is based on the Cameleon Reference Framework [14], which defines UI development steps for multi-context interactive applications. Four steps are identified: Task & Concepts (T&C), which represents the tasks to be performed by the users and the domain concepts required by the tasks to be carried out; Abstract UI (AUI), which defines interaction spaces that are independent from both modality and platform by combining several tasks according to certain criteria (this is done by means of Abstract Interaction Objects); Concrete UI (CUI), which represents an AUI at a lower level of abstraction, for a specific context of use, and by means of Concrete Interaction Objects; and lastly, final UI (FUI), which represents the operational UI that users actually interact with.

UsiXML provides several models to deal with multi-modal and multi-platform UIs, as well as an evolving transformation system to create intermediate models, mappings [15] (links between models) and code. The UsiXML’s metamodel is composed of several models (see Figure 1) that represent the main aspects of multi-modal and multi-context UIs, and they aim at supporting the Cameleon framework.

Figure 1: UsiXML’s models organization (including the Awareness Model added).

Although different development paths are possible in UsiXML methodology [7], the most widely-used one starts with the specification of the domain and task models. The DM is the link between the application core and the UI. That is, the DM is the interface the UI uses to access the functionality. In UsiXML, the DM is based on UML class diagrams. Therefore, this DM is actually a representation of a UML class and object diagram. On the other hand, the foundations of the task model in UsiXML is ConcurTaskTrees [16].
Thus, there is a hierarchical decomposition of tasks. These tasks are linked by means of temporal relationships that represent the temporal constraints among them. For instance, for a register form task, you could represent that you should first do your enter login task and then your enter password task. Then, an AUI is generated out of both the task and DM. AUI is composed of two basic abstract interaction object types: abstract containers and abstract individual components. Abstract containers represent the groupings of abstract individual components. Next, different CUI can be generated for different modalities (graphical, vocal, textual, etc). The CUI metamodel in UsiXML represents in an abstract manner those widgets usually found in the different modalities supported. For instance, for the graphical modality it includes containers, such as dialog boxes and windows, and individual components, such as buttons and check boxes. And lastly, the code (FUI) is generated for each CUI. Notice that the transformation process is made according to the target context of use specification described in the Context Model. The traceability between the different models is kept by means of the mapping model. For instance, these mappings represent that a task from the task model triggers a specific method in the DM, or that some abstract individual objects from the AUI are used to represent a specific task from the task model.

As aforementioned, awareness support has been found paramount to achieve real collaboration among the users of an applications. Nevertheless, in UsiXML no support is included to support the specification of all the artifacts required to bring the benefits of awareness to those applications designed by means of UsiXML.

To address this shortcoming identified in UsiXML, an Awareness Model has been included to represent awareness related aspects, but also some entities have been added to other metamodels, such as the Mapping Model, the Context Model and the Domain Model (DM) to link the Awareness Model to all the other metamodels (see Figure 1). Notice that throughout this paper all the entities of UsiXML will appear with gray background in the figures, while the entities of our extension will appear with white background.

4. Awareness support extension for UsiXML

The elicitation of requirements for awareness support should therefore be tackled at T&C level, together with the domain and tasks specification.
To describe our UsiXML’s extension, first awareness as an aspect of human interaction is described to explain its role in computer systems. Then, the Awareness model that enables the representation of any kind of awareness is described. We then describe how to specify Awareness Requirements and the process for manipulating the mappings created during that process.

4.1. Awareness

Awareness is a knowledge state. Endsley defines it as “the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future” [1].

Being aware is to know the current state of the environment and understand its behavior. The observer obtains information from his or her own perceptions, then applies intelligent processes to infer new knowledge about the observed elements: environment, entity, object, behavior, and so forth.

The system captures the required information from the environment, processes it and makes it available to the actors, who need it to create awareness and improve interaction.

To support awareness, the previously mentioned awareness facets should all be modeled. To this end, two main models and their relationships have been identified: (1) Awareness characterization model and (2) Awareness requirements model.

Awareness characterization aims at defining the awareness information structure in terms of the domain (student, activity, professor, etc.) and system (pointer, session, task, etc.) entities. Awareness Requirements answer the following questions about the awareness information requirements and their context:

- **Why?**
  - Informal requirements, which may be described in natural language, answer the *why* question of the requirements. Each AR is useful as both documentation and guidance during the development process and describes the justification for the awareness mechanisms included.

- **Who?**
  - The system users or actors that require awareness. These can be identified as a static or dynamic group, as a role, or individually.

- **What?**
The awareness information that is provided to satisfy the AR. Awareness information is extracted from the domain and system entities. The Awareness characterization model (see section 4.2) is used to specify this kind of information during the system development.

The access restrictions for the required awareness information. These are used for privacy management and to reduce disruption; e.g., normally, a user should not be aware of every user in the system, but only of those collaborating with him or her.

- **Where?**
  - The task (and sub tasks) which require awareness.

- **When?**
  - The system, domain or context state that limits the applicability of the requirement; e.g., apply the requirement when the system time is between 9 and 11 pm.

- **How?**
  - The interaction techniques chosen to provide the users with awareness information. They depend on the client’s technology, platform and context, as well as any special needs the actors may have.

The relationships between awareness and development models define a closely linked structure that answers the *how?* question of how awareness requirements are supported.

These questions are important as a mean to provide a guide to designers during awareness requirements elicitation. These questions describe the different facets that any awareness requirement should considered. Thus, designers are asked to answer all these questions in order to proceed with awareness requirements modeling stage. Designers should use these questions as a checklist to make sure they actually have the required information to properly model the awareness requirements.

Following the UsiXML approach, several awareness-specific metamodels and mappings have been developed that are connected to the UsiXML ones, thus building a structure of mappings to provide awareness support at all development stages.

### 4.2. Awareness characterization

An Awareness of “something” (the existence of a resource, the location of an entity, the duration of a process, etc.) is defined by a set of awareness
elements. An awareness element refers to something that can be received and understood by an observer; e.g., location, height, weight, and so on.

To define any kind of awareness type, we provide the Awareness model (Figure 1). The literature about awareness presents several awareness types, depending on what users may need during their system tasks. These include Group awareness [2], Activity Awareness [8] and Situation Awareness [1], among others [9]. The entity $awConcreteType$ in our meta-model, shown in Figure 2a, represents an Awareness Concrete Type (ACT) composed of awareness elements. These ACT support the specification of the different awareness types available in the literature. Latter, these ACTs can be reused from one project to another, since they represent generic knowledge. The awareness requirements will use these ACTs as the main building block.

![Figure 2: Awareness characterization metamodel to: a) define. b) characterize.](image)

There are three types of awareness elements (see Figure 2):

- **Concrete element** (CE). It is a feature of an observed entity, e.g., height, size, weight, color, location, identity, capacity, among others. These represent entity attributes that are obtained or stored by the system.

- **Composite element**, which represents the understanding of certain elements (concrete or composite) as a group. It is a derived element created from the values that originate from other elements, either concrete or composite, such as Speed (computed from Distance and Time) and Knowledge (computed or inferred using complex algorithms). The $awCompositionFn$ entity supports the calculation processes. Composite elements can be seen as derived attributes, whose values are computed from the values of other awareness elements.

- **Awareness Projection** (AP), which represents the values of one or more elements (either concrete or composite) over time. An AP returns the
value of the awareness elements for a given period of time, such as the weather forecast for the next week. The \textit{awProjectionFn} element supports these kinds of computation processes by calculating the values for a given range of time.

The \textit{Composite element} is defined as a composition of \textit{Concrete elements} and/or other \textit{Composite elements}, joined by a composition function. e.g., the Composite element \textit{Speed} is composed of two CEs called \textit{distance_traveled} and \textit{elapsed_time}, using the composition function \( f(d, t) = d/t \).

We define APs as the computation process used to obtain the value of some elements (concrete or composite) in the future. An AP answers the question “what value will the element X have at Y (future) time?”; e.g., at 11pm tomorrow, gas tank GT01 will contain 15% of its capacity if its usage does not vary much. To support APs, the system should store the value changes over time for the elements belonging to the projection. This technical requirement is inherent to the nature of projection, since any complex analytic process uses the historical element values.

An ACTs represent the structure of awareness information elements. Their CEs are linked to domain class attributes to create sources of awareness data. The mapping [15] \textit{observes} in Figure 2b is used for this purpose.

ACTs represent generic awareness types. Therefore, they can be reused from one domain to another. Thus, the knowledge about the awareness types does not depend on the application domain. Nevertheless, the domain model for different applications will be different, and the same concept might be available in a different class and even under a different name. Since there is a mapping between the CEs of the ACT and the USIXML domain classes of the application, these mappings cannot be reused from one project to another. Therefore, these mappings must be specified each time the ACT is used in a different domain.

4.3. Awareness Requirement

An awareness Requirement (AR) represents the needs that a group of system actors has for up-to-date information in a given situation in order to create the knowledge state they need for decision making during a task.

The definition of system and user requirements takes place during the first stages of software development, but it can be refined after user testing, prototyping or system evolution. ARs should be treated the same way, as they are also special information requirements.
The definition of an AR consists of several phases and it is separated in some specialized entities related with the awareness support aspects explained in Section 4.1.

To define the required generic awareness we use the ACT. Then, for the specific requirement we select the elements that will be provided using the awSet entity (Figure 3b), which groups awareness elements from the same ACT.

![Figure 3: a) UsiXML’s Context model extension to enable runtime constraints for Awareness users or applicability situations. b,c,d) awRequirement models.](image)

The required information is always delimited in some way. We have added the entity selectionConstraint (Figure 3c) to the Domain Model with the purpose of defining constraints that delimit the instances of the domain classes related with the ACT. A selectionConstraint combined with an awSet defines an awDataSource (Figure 3c), which represents a specific grouping of awareness information.

The entity awRequirement presents the awareness information required in some situation and it is composed of several awDataSources (Figure 3d). At this point, the situation where this awareness is required is not yet defined. The runtimeCondition (Figure 3a) was added to the Context Model of UsiXML to specify those situations where this awareness is required.

The runtimeCondition and the selectionConstraint represent predicates to be applied to the context situation or to the awConcreteType data sources (domain classes). These predicates can be expressed by using different languages, such as OCL and SQL, among others.

The awRequirement is linked to the task from the task model that requires the AR via the mapping requires, as shown in Figure 4a. This relationship means that during the task, within a specific context and satisfying the related RC, some specific actors (selected via runtimeCondition) will receive
Figure 4: Main awareness support relationships in UsiXML used to propagate awareness from a) Tasks to b) User Interface.

Information from the awareness data sources (\textit{awDataSource}) of the requirement, restricted by the \textit{selectionConstraint}.

Awareness support propagation starts with the mappings used to link the tasks and the \textit{awRequirements}. Since UsiXML approach is task-centered, the generation of the UI will begin taking into account both the task and the ARs they have. The mapping \textit{observes}, shown in Figure 4b, will be used to link the awareness elements to those AUI elements (AIOs) used to represent them. UsiXML’s mappings complete the awareness support structure.

Finally, the model to provide awareness support is complete when all these relationships are established.

Using this approach, the system provides the users with the awareness they need and only when they need it, reducing the cognitive overload, distraction and privacy issues related with the awareness support.

5. Case study

To illustrate the specification of ARs a case study for the management of fire trucks in a Fire Station is used. The system can be used by either an expert or an auxiliary firefighter. Each one has different skills, but they must do the same thing: send the required fire trucks to cover current emergencies. Due to space constraints the example is focused on two ARs. UML object diagrams are used to better describe the instances of the entities of our metamodel used to model this case study. In the figures, plain lines represent that an object is composed of other objects.

The example begins with the identification of the domain classes and the tasks the user will have to achieve through the UI. The \textit{Firetruck} domain class used in this case study has the following attributes: \textit{id}, \textit{name}, \textit{gas_capacity}, \textit{water_capacity}, \textit{location}, \textit{being_repaired}.

Then, ARs are identified and documented. We aim to provide the expert firefighter with all the up-to-date information about the fire trucks, and the
<table>
<thead>
<tr>
<th>Awareness Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Id</strong></td>
</tr>
<tr>
<td><strong>Actors (Who)</strong></td>
</tr>
<tr>
<td><strong>In task (Where)</strong></td>
</tr>
<tr>
<td><strong>Required information (What1)</strong></td>
</tr>
<tr>
<td><strong>Access restrictions (What2)</strong></td>
</tr>
<tr>
<td><strong>Applicability situation (When)</strong></td>
</tr>
<tr>
<td><strong>Justification (Why)</strong></td>
</tr>
<tr>
<td><strong>Other</strong></td>
</tr>
</tbody>
</table>

Table 1: Example of an Awareness Requirement.

auxiliary firefighter with “simpler” (but not incomplete) information, and a projection of the availability and the location of the fire trucks. Thus, two ACTs have been identified to do so: Fire truck Awareness (myFiretruckAw) and Fire truck Location Awareness (myFtLocationAw). One of the AR used in this case study is documented in Table 1. This table shows a template to gather ARs, which can be described in natural, or any other, language. The template is similar to the one presented in [17]; however, we have separated and identified the components of ARs more precisely to provide designers with further guidance during the description of these requirements. Once the ACTs are identified, the elements that provide the information required are modeled (as shown in Figure 2a):

myFiretruckAw has the following elements (see Figure 5): the Concrete Elements (`egas`, `ewater`, `eonmaintenance`, and `eid`), one Composite Element called `cavailability` (derived from the values of `egas`, `ewater`, and `eonmaintenance`) and one Projection called `pavailability` to compute the value of `cavailability` in future time.

Figure 5: Definition of the example’s Awareness Types: **myFiretruckAw** and **myFtLocationAw**.
The Composite element cavailability represents that a firetruck is available. The projection pavailability represents that a firetruck will be available at some moment of time. Their values are computed by the system.

Using the arrow icon to represent a observes relationship, we subsequently show the mappings between awareness elements and those domain class attributes from the DM of UsiXML they observe (see Figure 6). For instance, the id attribute from the domain class Firetruck will be observed by the concrete element eid. Thus, we are completing the characterization process (as shown in Figure 2b).

---

**Figure 6:** The result of the Awareness Type's characterization process through the observes mapping.

Next, awareness sets and datasources are defined (see Figure 7) according to the awareness required in each case (see Figure 3b):

On one hand, the expert user requires all the awareness elements from both myFiretruckAw and myFtLocationAw concrete awareness types. Therefore, the set awftExpertSet is defined as a composition of the following concrete elements (see Figure 7): tsegas, ewater, eonmaintenance, eid and the awftLocationSet set is composed of the concrete elements: efiretruck_name and elocation.

On the other hand, the auxiliary user requires higher-level knowledge, such as the availability element. We thus create the awftAuxSet set (see Figure 7) composed of: id, cavailability, and pavailability.

The definition of the awDataSource for the previous awSets is shown in Figure 8.

An awDataSource is composed of an awSet which gives the structure and one or many selectionConstraints (which delimit the data to be provided to the users). The selectionConstraints s1ExpertUser, s1AuxUser and s1AllUsers are used to specify constraints for the each corresponding awDataSource. e.g., the expert user may receive information about all the Fire
Figure 7: Awareness sets defined to group the Awareness elements the Expert and Auxiliary users require.

trucks, including those being repaired. That constraint would be specified in $s1ExpertUser$.

Figure 8: Awareness datasources defined to specify the Awareness information the Expert and Auxiliary users will be provided with.

After ARs are defined along with the applicability situations for each requirement, they are linked to the task where this AR is needed.

An $awRequirement$ is composed of as many $awDataSource$ as required (as shown in Figure 3d). The expert user requirement $awftExpertReq$ is composed of $awftExpertSrc$ and $awflLocationSrc$. This means that the expert user will receive the awareness information from both awareness data sources.

Figure 9: Awareness requirements defined to group the Awareness information the Expert and Auxiliary users will be provided with.

There are two ARs for the task $ftManagementTask$ and they are restricted by the runtime conditions $rcExpert$ for expert users, and $rcAux$ for auxiliary
users (see Figure 10). Therefore, it means that \texttt{ftManagementTask} task has the \texttt{awftExpertReq} awareness requirement when the runtime condition \texttt{rcExpert} is met. This task also has the \texttt{ftManagementTask} awareness requirement when the runtime condition \texttt{rcAuxUser} is met.

The transformation process can use the mappings between models to propagate awareness in the same way the domain and task aspects are propagated into the AUI and the CUI by using the mappings. This propagation process [18] is a cornerstone of UsiXML’s methodology.

6. Evaluation Design

As part of the future work, we are designing an evaluation for the extension of UsiXML described in this work.

The Awareness models presented above extend the specification capabilities of UsiXML in order to model (1) Awareness information requirements and (2) Awareness support as part of the Task & Concepts layer of the UsiXML’s methodology. Since the original UsiXML language has no support for the specification of awareness, it is clear that our extensions improves UsiXML when designing collaborative systems. Nevertheless, many different metamodels could be used to represent all awareness aspects required to provide awareness support. In this sense, an evaluation of the metamodels and the constructs used to provide awareness support would help in identifying possible issues.

An informal evaluation was already conducted in collaboration with collaborative systems developers from SymbiaIT corporation to detect potential flaws in the metamodels provided. This corporation is one of our partners in UsiXML project, where they aim at creating collaborative applications by using UsiXML framework.

This informal evaluation was two-folded. On the one hand, four developers were asked to check some awareness requirements models specified by using our metamodels. They were asked to verify that the questions de-
scribed in Section 4.1 were answered in the awareness requirements models provided, and ensure they could understand what the answer to those questions was. Some issues were then detected regarding the way the instances of the domain attributes were being filtered by using the specification of constraints.

The second part of the informal evaluation took the opposite path, that is, we asked the same four developers to specify some awareness requirements by using our metamodels. Then we check the metamodels to find out whether the subjects of the evaluation had been able to specify all the answers to the questions described in Section 4.1.

After the analysis and the interviews with the developers, we found out that we should improve the reuse of parts of the specification. Therefore, awareness set concept was introduced into the specification. Although these specifications were actually really useful, there are more aspects of the metamodels that should be more thoroughly assessed. Therefore, more evaluations are planned.

One of the most interesting aspects of any modeling language is its “understandability”. Understandability has been recognized as a quality factor for both usability and software products.

Our extension is close to information requirements modeling. In this area, [19] has conducted an evaluation of a requirements language extension, namely CSRML [13], which enables the specification of requirements related to CSCW systems and the Workspace Awareness [2]. In a similar way, the evaluation designed consists in a case study that will be provided to the subjects of the experiment, already modeled by using our extension. Then a questionnaire will be answered by the subjects where they will be asked to answer questions regarding the different aspects of awareness identified in section 4.1 to test how understandable is the modeling of the different facets of awareness.

Using the evaluation described in this section the awareness extension of UsiXML can be assessed, beyond the experts evaluation already conducted.

7. Conclusions and Future Work

MDD techniques tackle important problems in the software development process, such as the lack of systematic practices or traceability. They rigorously manage requirements and ensure their justification and fulfillment.
They have been similarly used for UIs, allowing requirements to be incorporated into the final UIs and assisting developers and users of complex interactive systems.

Our main contribution is a model-driven approach for developing awareness supported multi-platform and multi-modal UIs based on the UsiXML methodology. The integrated awareness models enables the representation of any kind of awareness and extends the base notation to support awareness at requirements level.

A new aspect of this approach is that awareness is treated as an information requirement throughout all development stages, making it possible to re-use models, trace requirements support, and verify their accuracy. Other advantages also arise from being able to process models using software.

Awareness models are useful because they can be used during the development process to analyze the information that users are provided with: its origin and its potential impact on issues such as privacy violation, distraction, information overload or lack of valuable data for decision making.

One implicit contribution of this extension is to integrate awareness as another important component of software development. Awareness is not an optional add-on, but an essential component in any methodology or tool for developing CSCW.

As with most model-based or driven approaches, a bigger effort is required for the first awareness models created. Nevertheless, this initial effort is drastically reduced because of model reuse in subsequent projects. Specially, the reuse of ACTs.

As future work, we are currently working on improving the transformation process that manages the awareness models throughout the different steps of the UsiXML software development process, specially by providing a repository of transformations for awareness specific models. We also plan to conduct the evaluation proposed to our models in order to improve them with more designer’s feedback.

Acknowledgments

This research has been funded by the MICINN of Spain (TIN2008-06596), the CONACYT of Mexico and the UsiXML Project (ITEA2-08026), funded by the Spanish government (AVANZA TSI-020400-2011-20).
References


