On stories, models and notations: Storyboard creation as an entry point for model-based interface development with UsiXML

Kris Luyten, Mieke Haesen,
Dawid Ostrowski, Karin Coninx
Hasselt University - tUL -IBBT
Expertise Centre for Digital Media
{kris.luyten,mieke.haesen,dawid.ostrowski,
karin.coninx}@uhasselt.be

Sylvain Degrandsart[1,2]**, Serge Demeyer**[1]
[1] Dept. Mathematics and Computer Science,
Universiteit Antwerpen
[2] University of Mons - UMONS
{Sylvain.Degrandsart,
Serge.Demeyer}@ua.ac.be

ABSTRACT
Storyboards are excellent tools to create a high level specification of an interactive system. Because of the emphasis on graphical depiction they are both an accessible means for communicating the requirements and properties of an interactive system and allow the specification of complex context-aware systems while avoiding the need for technical details. We present a storyboard meta-model that captures the high level information from a storyboard and allows relating this information with other models that are common for engineering interactive systems. We show that a storyboard can be used as an entry point for using UsiXML models. Finally, this approach is accompanied by a tool set to make the connection between the storyboard model, UsiXML models and the program code required for maintaining these connections throughout the engineering process.

Author Keywords
UsiXML, Storyboards, Model Transformations

INTRODUCTION

For over a decade, model-based interface development has been subject of ongoing research to improve the methods and tools on the one hand, and to bridge the gap with traditional software engineering on the other hand. Model-driven engineering seems to be one of the central methods that can connect model-based interface development with software engineering because of the concepts they share: models that describe different aspects of an interactive system. Despite the availability of various tools [6,9,10,11], most models require domain expertise or technical knowledge. As such, they fail to serve as a means of communication for the whole team including all stakeholders and their usage is limited to particular stages in the engineering process. Specifically for engineering complex systems such as context-sensitive or distributed multi-user systems, maintaining involvement of the whole team and providing consistency checks with initial requirements is hard.

We believe a simple and graphical tool can resolve many issues, given it can be easily connected with the typical models that are used throughout the engineering process.

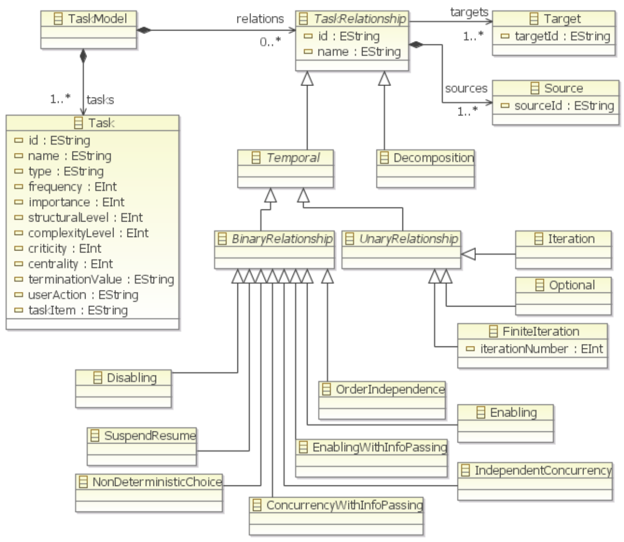
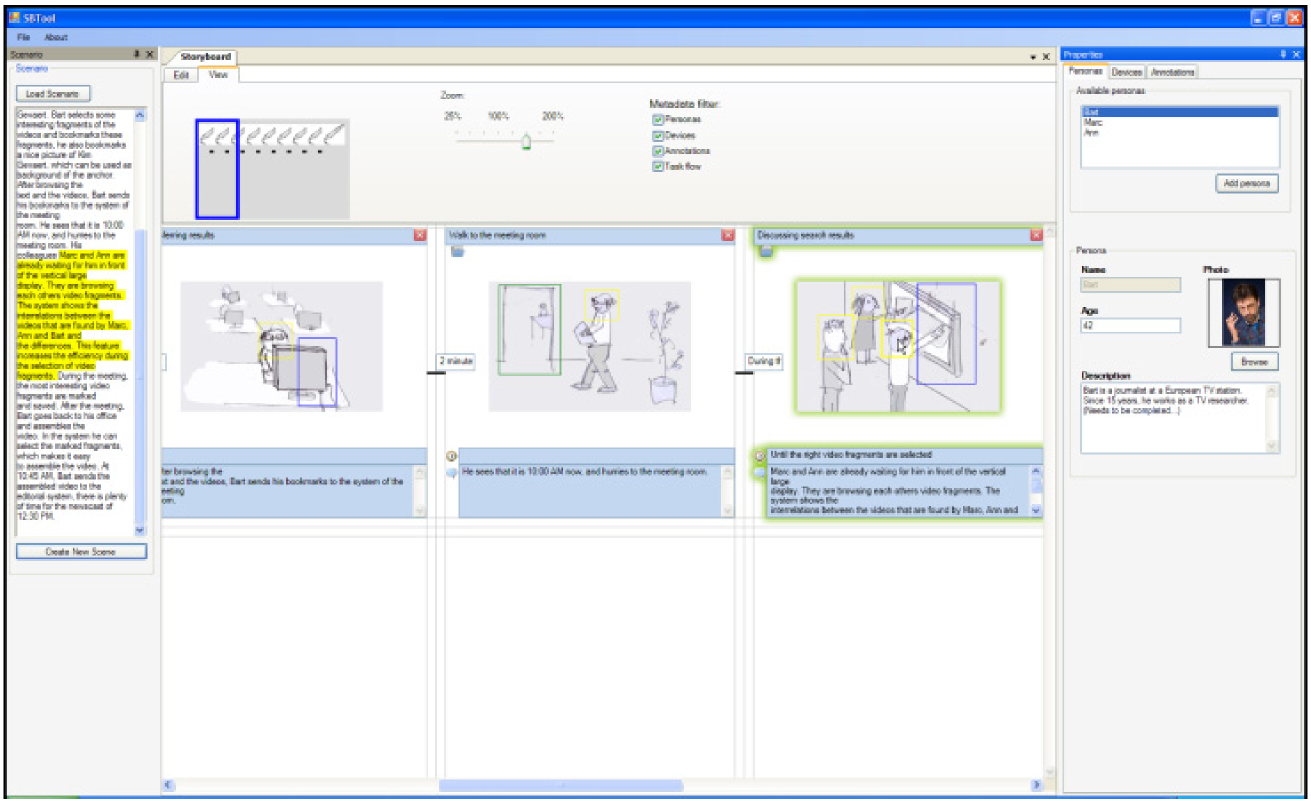


Figure 1. The meta-model for the UsiXML Task Model.

ABOUT THE MODELS AND NOTATIONS

Model-based interface design is hard to get started with: most models use an abstract notation and require at least some domain expertise. The task model is the one model that is often used as a starting point for model-based interface development processes. A task modeling notation such as the ConcurTaskTrees (CTT) notation [11] uses a graphical notation for identifying the different task types and has a specialized notation for specifying how tasks behave in time with relation to each other in time. UsiXML does not provide a separate graphical notation such as CTT, but does include the semantics for similar elements in its specification. This leaves the choice for a graphical notation up to the tool developer. Figure 1 shows the meta-model for UsiXML compliant task models.

In Faure, D., Vanderdonckt, J., (Eds.), *Proc. of 1st Int. Workshop on User Interface Extensible Markup Language UsiXML'2010* (Berlin, 20 June 2010), Thales Research and Technology France, Paris, 2010. ISBN 978-2-9536757-0-2



Geesert. Bart selects some interesting fragments of the videos and bookmarks these fragments, he also bookmarks a nice picture of Kim Geesert, which can be used as background of the anchor. After browsing the text and the videos, Bart sends his bookmarks to the system of the meeting room. He sees that it is 10:00 AM now, and hurries to the meeting room. His colleagues Marc and Ann are already waiting for him in front of the vertical large display. They are browsing each others video fragments. The system shows the interrelations between the videos that are found by Marc, Ann and Bart and the differences. This feature increases the efficiency during the selection of video fragments. During the meeting, the most interesting video fragments are marked and saved. After the meeting, Bart goes back to his office and assembles the video. In the system he can select the marked fragments, which makes it easy to assemble the video. At 10:45 AM, Bart sends the assembled video to the editorial system, there is plenty of time for the newscast of 12:30 PM.

ter browsing the
xt and the videos, Bart sends his bookmarks to the system of the
eeting
om.

He sees that it is 10:00 AM now, and hurries to the meeting room.

Until the right video fragments are selected

Marc and Ann are already waiting for him in front of the vertical large
display. They are browsing each others video fragments. The system shows the
interrelations between the videos that are found by Marc, Ann and

Bart
Marc
Ann

Name: Bart
Age: 42

Bart is a journalist at a European TV station.
Since 15 years, he works as a TV researcher.
(Needs to be completed...)

Figure 2. Screenshot of the COMuICSer tool. The centre panel shows the storyboard, while the left and right panels are reserved for the scenario and annotations.

Just as is the case with CTT, there are tools that allow specifying the task model by using the UsiXML notation, e.g. IdealXML [10] or KnowiXML [6]. We see that the CTT notation serves as an important inspiration for most of these tools and similar graphical notations are being used. UsiXML does provide well-defined relationships with other typical models such as the dialog, context and presentation model. For this purpose, it is based on the well-known Model-Driven Engineering (MDE) [13] approach and uses (semi-automatic) transformations between models to progress from abstract toward concrete models.

UsiXML supports an MDE approach by defining a set of meta-models that contain all elements and relations included in the different models [16]. It tries to cover all models that are required for user interface analysis and design, and encompasses models that are commonly used in model-based interface development. We propose the addition of another model that can serve as an entry point for the whole process to engineer an interactive system: the storyboard model. In the next section of this paper we define a storyboard model and we show how it can be connected to other models contained in the UsiXML specification.

STORYBOARDS: GRAPHICAL NARRATIVE MODELS

Scenarios [2] are suitable to be used by all team members in user interface design to define the first concepts of a future system. Unfortunately, technical people encounter difficulties when translating these scenarios into technical specifications and models [5,7]. Furthermore, when using scenarios, it is necessary that all team members have read the narrative. Nevertheless, the scenario can be interpreted ambiguously [15]. To bridge the gap between scenarios and models, and to increase involvement of the entire team, storyboards can be created. We define a storyboard as: a sequence of pictures of real life situations, depicting *users* carrying out several *activities* by using *devices* in a certain *context*, presented in a *narrative* format. Each scene in the storyboard can depict a fragment of the scenario, and hence provides the connection between the scenario and the storyboard. This specific definition immediately provides us with a clear overview of the four primary pieces of information that can be found in a storyboard: *users*, *activities*, *devices* and *context*. COMuICSer is our approach for storyboarding that embraces these four elements. The COMuICSer tool supports a graphical representation of a scenario and allows a smooth integration with structured engineering models such as the UsiXML models. The COMuICSer tool is shown in Figure 2. In this figure, the tool is being used to visualize a scenario of how a multimedia search interface can be used on several platforms by an editorial board of a TV channel.

In contrast to a scenario, this graphical representation contains more structural information, while the entire team can still easily understand the information conveyed by the storyboard.

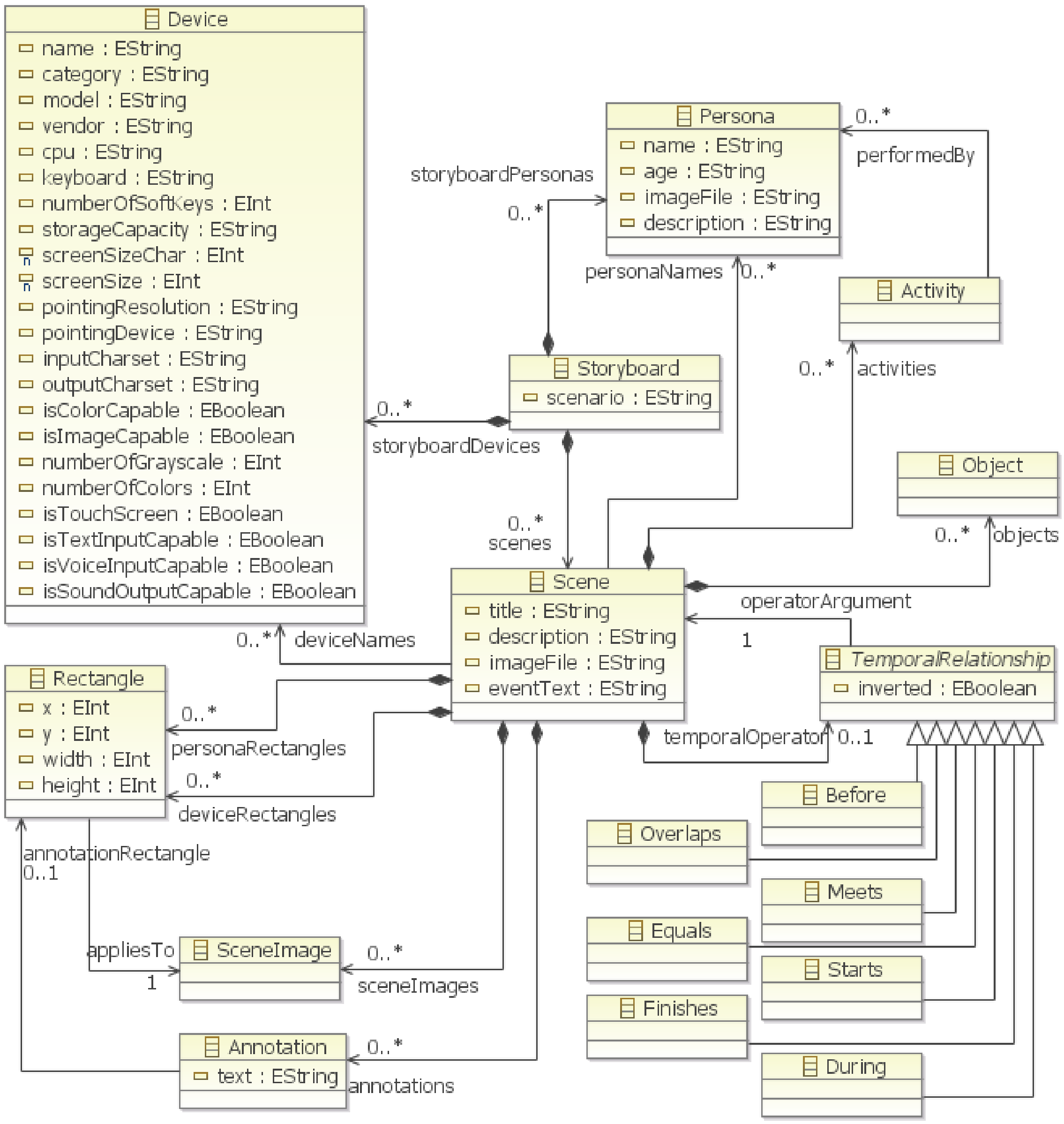


Figure 3. Our initial COMuICSer storyboard meta-model. It contains the graphical depiction with the objects of interest (context), personas, devices and activi ties. Scenes are related using the Allen interval algebra operators.

By annotating the COMuICSer storyboard, it is possible to provide a connection to structured engineering models and UI designs. In the COMuICSer tool, rectangles can be drawn on top of scenes, to specify particular annotations. These rectangles can refer to personas [12], device specifications and free annotations. Furthermore, the connections between scenes of a storyboard can be labeled, while each scene can contain timing information (e.g. about the duration of a situation in a scene). One can tag each storyboard scene with keywords like a picture can be tagged in most photo management software today. Among the tags, the verbs provide us with a list of activities and tasks that are being executed in a scene. The nouns specify objects of importance in such a scene. We want to show that ”low-fidelity” models such as storyboards can be easily connected with UsiXML models and eventually generate parts of these models based on the storyboards. This would avoid a completely manual transformation of high-level requirements that are contained in a storyboard but at the same time does not exclude the creative input that is often part of the storyboarding process.

A STORYBOARD META-MODEL

To enable integration of COMuICSer storyboards with other models, we need a meta-model for these storyboards. Given the freedom to use arbitrary drawings and pictures as scenes of the storyboard, the meta-model will provide a scheme for the metadata that can be found in the respective scenes of the storyboard. Our initial storyboard meta-model is shown in Figure 3. It is by no means our intention to formalize or restrict storyboarding activities with this meta-model. We merely want to connect this model with other artifacts that are used to design an interactive system. Our approach still provides all degrees of freedom typical for storyboard creation and mainly comes into play when the transformation of a storyboard into other models is required. In this section we provide an overview of the different elements that make up the storyboard meta-model.

There is one central element in the meta-model: the *Scene*. A set of scenes that are related using *TemporalRelationShip* elements is a *Storyboard.* The *TemporalRelationShip* element is based on Allen´s interval algebra [1]. Though, similar to comics [8], the most common relationships used in storyboarding are the "*before*" and "*meets*" relationships, we think parallel activities should be supported since they are common in collaborative and multi-user activities. "*Before*" indicates one scene happened before another, and there is undefined time progress in between scenes. "*Meets*" indicates one scene is immediately followed by another scene, and the time progress between two scenes is virtually none. It also allows us to define more precise temporal relationships between scenes that can be exploited later on, e.g. by mapping them on the temporal relationships that are used in the task model.

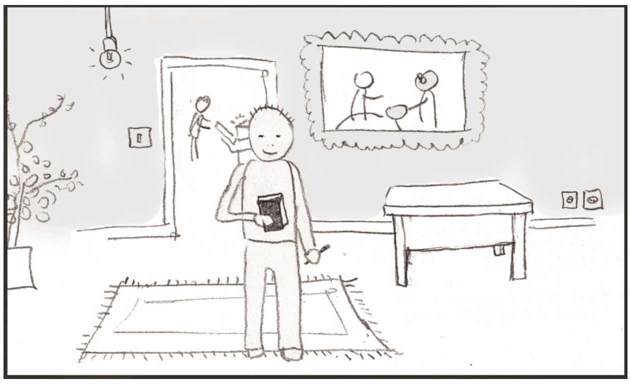


Figure 4. Scene from a storyboard that can be juxtaposed with labels that show the associated meta-data. The storyboard meta-model provides a framework to capture and serialize this meta-data for further usage.

A *Scene* is annotated with different types of information: *SceneImage* is a graphical representation of the scene, *Personas* specify archetypical users involved in the scene, *Devices* present what type of computing devices and systems are used within the scene. Because of the graphical depiction used in the *SceneImage*, it describes the context (situation) of the activities of the users in a comprehensive way. In contrast to traditional approaches in which there is some predefined structure for defining the context of use, images can be used to infer the context of use. When constructing a storyboard, the drawings or photographs used, will often contain a lot of contextual information. Dow *et al.* show storyboarding, especially contextual storytelling, is useful for context-aware application design (in their case ubicomp applications) but lacks a good way of formalizing the context data [5].

By providing tagging of scenes, we support a rudimentary way of translating the context inferred from the *SceneImage* into a readable format. *Objects* are physical objects in a scene that are also sufficiently important to label. Take the scene depicted in Figure 4 for example: two objects that might be of interest are the light bulb (needed for watching the book) and the table in the room (needed to hold the books to read). This implies these objects could be taken into account later in the engineering cycle and thus need a representation in the storyboard meta-model. *Personas* on its turn have a set of related activities (or tasks as you might wish) and are both related to the whole *Storyboard* as well as the separate *Scenes* they participate in.

Though tagging is used to allow for a more comprehensible description of the context-of-use in a scene, several other elements of the storyboard meta-model already provide additional contextual information. Consider the UsiXML context meta-model depicted in Figure 5. This graphical depiction is created based on the UsiXML context meta-model scheme and was clearly inspired by browser profiles that were popular at the time this model was created. One can identify other one-to-one relationships between the UsiXML context meta-model and the storyboard meta-model. The *Persona* and *Device* classes can be used to generate *userStereoType* and *hardwarePlatform* classes respectively. One valuable extension is to integrate the custom tags with Wordnet (http://wordnet.princeton.edu/) to extract richer semantics of these tags. This could lead to a more complete transformation than is currently supported.

We showed a graphical depiction (*SceneImage*) could have high value to obtain a usable model of the context of use in previous work [17]. Figure 4 shows a scene from a storyboard that was tagged with labels about the information included by the storyboard meta-model. Several information items contribute to the context model as defined by UsiXML in Figure 5.

We do not consider our current meta-model to be complete. It does include a set of minimal elements for the storyboard to be connected with other models in a MDE process. The COMuICSer tool can generate model instances for this meta-model, which in turn can undergo transformations to enable integration with other models. We currently use this approach for two goals: first, to define mappings and consequently generate partial models (e.g. generate a set of tasks and initial temporal relationships based on the scene orderings and included activities); second, to check for consistency (e.g. the sequence of an activity diagram might contradict the temporal relationships from a storyboard).

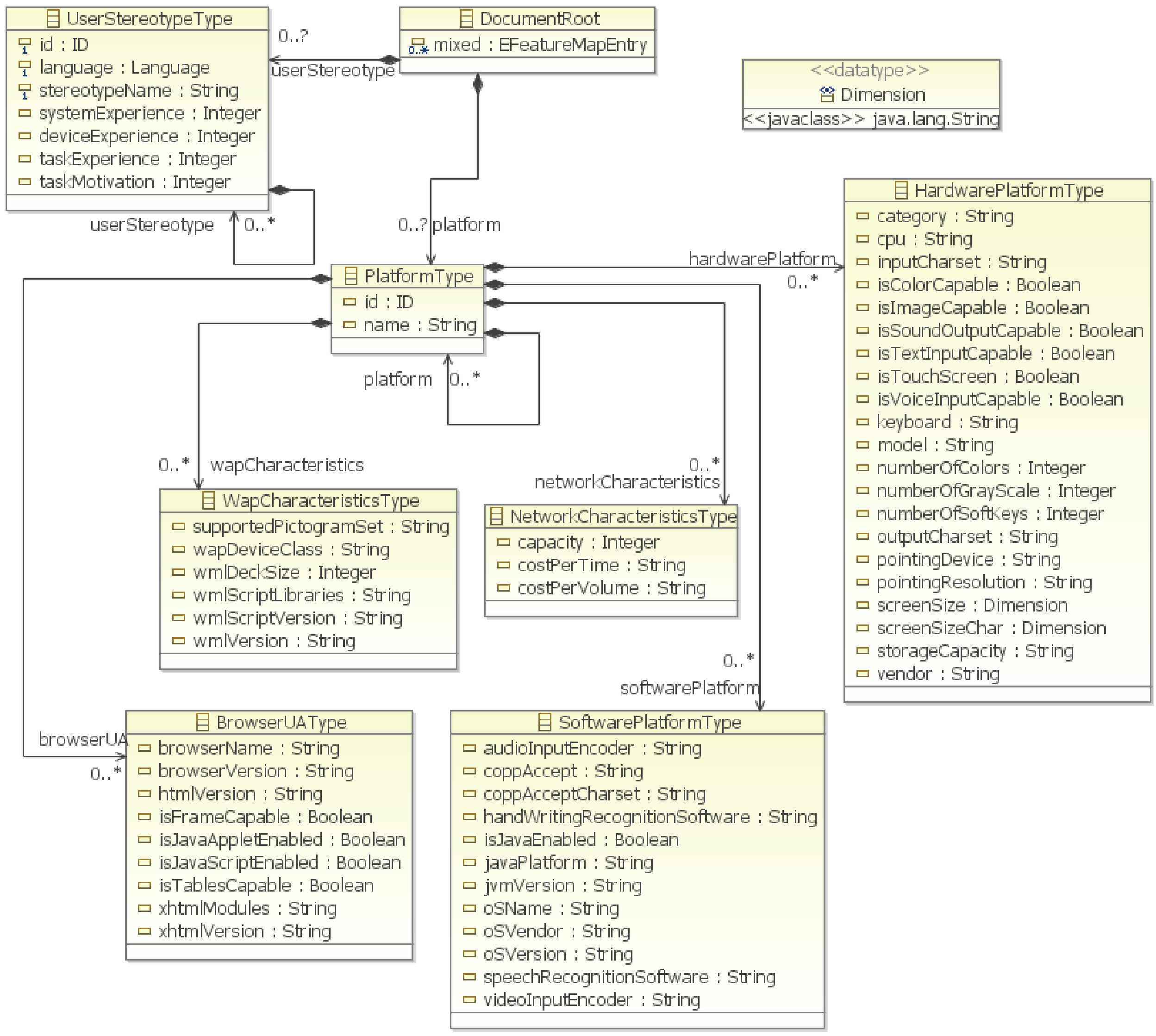


Figure 5. The meta-model for the UsiXML context model.

MAPPING BETWEEN STORYBOARDS AND USIXML

Now that we have the meta-model for the storyboard in place, we can connect it with other (UsiXML) models. Since the storyboard is often used as the initial model at the start of the engineering cycle, it can be used to generate parts of other models or to check for consistency when later models are created. The usage of a MOF-compliant meta-model ensures later integration with other MOF-compliant models is also possible. In this paper we mainly focus on integration with UsiXML models, but we envision integration more closely with traditional software engineering models in the future (e.g. activity diagrams or state charts). To show the possibilities of a storyboard meta-model, two example transformations are presented. First, by defining a mapping rule between the storyboard model and the UsiXML task model (depicted in Figure 1), we can partially generate a UsiXML task model or check its consistency. Second, by exploiting the metadata from the storyboard we can add detail to the UsiXML context model (Figure 5).

Partial Task Model Generation and Consistency Check

Each scene in a COMuICSer storyboard depicts one or more activities. With the mapping editor (see next section), a developer can specify a mapping rule stating each activity from the scene should become a leaf task in a task tree. To enforce these tasks to be leaf tasks, we need to define a constraint that imposes this task cannot be decomposed any further.

All activities within the same scene are also valid during the same period of time. Thus, we can define that all activities of a scene are mapped on leaf tasks from the same enabled task set. An enabled task set is a definition from the ConcurTaskTrees language that specifies a set of tasks is valid during the same period of time. Since scenes can be related using the Allen interval Algebra [1], for some leaf tasks it can be derived whether they need to be executed in parallel with other tasks or should use an enabling operator with its siblings. The transformation into enabled task sets could be defined using a constraint. There are clearly extensive possibilities exploiting the temporal relations and activities that occur in the storyboard to feed the task model or check the task model for consistency. It is mainly a matter of defining the rules one wants to apply and write these down as transformations that take the storyboard model as an input. Unfortunately, the Allen interval operators do not offer one to one mapping relationships with the typical CTT temporal operators. For example, two scenes s_1 and s_2 containing each two activities, activities(s_1)=$\{a,b\}$ and activities(s_2) =$\{c,d\}$, can be translated into either *(a[]b) >> (c[]d)* or *((a>>c)[]d)[]b*.

The personas, containing a description of a hypothetical archetype of an actual user, can add the distinction of different roles to a model. The different personas contained by the scene of a COMuICSer storyboard, can be connected to one or more tasks of a task model. When several personas are available in a storyboard, the personas can be the foundation for a cooperative task model.

The following code listing shows an ATL transformation for generating tasks from activities within scenes:

```
module activity2task;
create OUT : MUsiXmlTask from IN : Mstoryboard2;
rule activity2task{
   from
      a : Mstoryboard2!Activity
   to
      t : MUsiXmlTask!Task (name <- a.title)
}
```

Partial Context Model Generation

Another UsiXML model that has a clear relationship with the storyboard model is the UsiXML context model. We depicted the context model in Figure 5 for the reader´s convenience. From literature it is clear that context is a vague term that encompasses many elements. The UsiXML context model makes a trade-off of what is useful in the UsiXML model-driven engineering process, but we think the context model will be subject to change in the future. In the storyboard, we think the combination of the graphical depiction, showing the situation of the users, the predefined elements in the meta-model and the arbitrary tags that are allowed, provide a powerful means to describe the context-of-use as specific as possible. For "hidden" context properties, such as the platform specification of a device, the COMuICSer tool provides dialogs to fill in this data. There are separate dialogs for adding the device profile, the persona descriptions and describe activities. Notice the UsiXML mapping model does not provide explicit support for a changing context over time, so we need to generate multiple UsiXML context model instances. The UsiXML *event* definition links to a context in which the event is executed, so this could be used to describe changes in context too. Using the storyboard meta-model we can now construct a mapping to generate parts of the context model:

For each scene instance in the storyboard model:

- Each *Persona* element will be mapped on a new *userStereoType* element, as it is the closest to a persona UsiXML currently offers.
- Each *Device* element will be mapped on new *hardwarePlatform* and *softwarePlatform* elements (including properties such as *screenSizeAspect*, *osName*,…).
- Each *tag* that describes the *surroundings of the user and matches:*
    - *Noise* generates the *isNoisy* property and set it to *true.*
    - *Light* generates the *lightningLevel* property.
- If the scene description indicates the user is in a stressful environment generate the *isStressing* property and set it to *true*.

Now that we have a set of context models, each of them can be related by a set of tasks that are grouped per scene, of which the generation was explained in the previous subsection. To make this set of models useful and usable, there still needs to be tool support that allows a designer to specify user interfaces according to the context of use. One example of such a tool is Gummy, presented in [9], but has only rudimentary support for UsiXML. Another candidate tool could be SketchiXML [3].

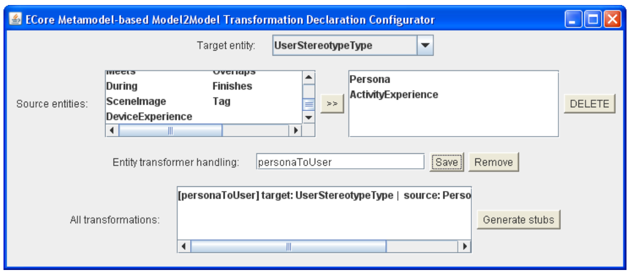
This approach makes it more convenient to specify the context, since most information is already contained in the storyboard. We believe this approach could also lead to a more complete definition of context, created from experience rather than using a pre-defined description. The combination of images and tags makes there are little to no limitations in defining context of use.

TRANSFORMATIONAL DEVELOPMENT TOOLS

We created tool support for progressing from the storyboard models toward UsiXML models. In spirit of openness of our approach, the Eclipse setup in which the tools are embedded is provided as free software and can be downloaded from the project website that can be found at the following URL: http://research.edm.uhasselt.be/kris/research/projects/StoryBoardML.

Our tool allows defining Model-to-Model transformation rules by creating mappings between entities contained in the source and target meta-models. Rule definitions are essentially data-driven. Each definition specifies a set of element types from the source model as input and an element type from the target model as output. The rule can be applied to instances of the source model and create the target elements according to this input. A screenshot of the tool is shown in Figure 6. It shows a mapping editor that allows defining a mapping rule. We consider this a proof of concept for showing the capabilities of our approach.

According to [4] our solution can be classified as an approach that uses a target-oriented organizational structure of unidirectional rules, and follows a strategy of deterministic rule application. Our solution also supports reuse of rules by means of inheritance or composition. The tool is able to generate method stubs in Java, which can be used later on to program all the details of data passing between entities. I.e. the transformation mechanism can be further refined in the method stub and techniques such as property value transfer, type lookup, type casting, data trimming and decoration of elements are all available. The mapping tool is mainly a starting point for further refinement and definition of complex transformations. Our aim is not to provide a full set of transformations, rather to enable a developer to define transformations on a concrete level when facing a set of high-level transformation rules that need to be enforced.


ECore Metamodel-based Model2Model Transformation Declaration Configurator

Target entity: UserStereotypeType

Source entities: Meets, Overlaps, During, Finishes, SceneImage, Tag, DeviceExperience → Persona, ActivityExperience

Entity transformer handling: personaToUser

All transformations: [personaToUser] target: UserStereotypeType | source: Perso

Figure 6. Mapping editor to define a target-oriented unidirectional mapping rule for generation partial UsiXML models from a storyboard model.

Our tool support is built on top of the Eclipse environment and makes extensive use of the Eclipse Modeling Framework Core (EMF core or ecore) [14]. Thanks to EMF, developers who create new rules can also use automatically generated code that supports manipulation and persistence of the instances of any meta-model. To provide smooth integration of COMuICSer and parts of UsiXML, we created an ecore version of storyboard meta-model. The UsiXML schemes were also converted in pure ecore models (model descriptions in XMI) so they can be easily used within the Eclipse environment. This has two major advantages: first, one all the MDE tools that are available in Eclipse can be used (querying, transformation, validation) and second, editing tools can be automatically generated for ecore compliant models. The Eclipse Modeling Framework can be used through a powerful API, which has allowed us to rapidly generate code templates based on our ecore conversions of the UsiXML models.

CONCLUSION

This paper presented a transformational approach to integrate storyboarding with various UsiXML models. For this purpose we created a storyboarding meta-model that serves as the starting point for applying the transformations. In contrast to the storyboard meta-model definition, the transformations themselves are not hard-wired in our approach. By using the features of the Eclipse Modeling Framework a developer can edit custom mapping rules using a simple interface. A custom set of UsiXML models using pure XMI was generated to enable smooth integration in the Eclipse environment. These mappings can be applied to the storyboard model to generate partial UsiXML models or check consistency with the storyboard when editing the UsiXML models. The work presented here is still work in progress. The tools and models described in this paper are fully functional, though they are not integrated to the point they allow full consistency checks between UsiXML and other models.

ACKNOWLEDGMENTS

This work is supported by the FWO project *Transforming human interface designs via model driven engineering* (G. 0296.08) and IWT project *AMASS++* (SBO-060051).

REFERENCES

1. Allen, J. F. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26,11 (1983), pp. 832–843.
2. Carroll, J.M. *Making use: scenario-based design of human-computerinteractions*. MIT Press, Cambridge (2000).
3. Coyette, A., Kieffer, S., and Vanderdonckt, J. 2007. Multi-Fidelity Prototyping of User Interfaces. In *Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2007* (Rio de Janeiro, September 10-14, 2007). Lecture Notes in Computer Science, Vol. 4662. Springer-Verlag, Berlin (2007), pp. 149–162.
4. Czarnecki, K. and Helsen, S. Classification of Model Transformation Approaches. In *Proc. of OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven* Architecture (Anaheim, 2003).
5. Dow, S., Saponas, T. S., Li, Y., and Landay, J. A. External representations in ubiquitous computing design and the implications for design tools. In *Proc. of the 6th Conf. on Designing Interactive systems DIS'2006*. ACM Press, New York (2006), pp. 241–250.
6. Furtado, E., Furtado, V., Sousa, K., Vanderdonckt, J., and Limbourg, Q. KnowiXML: A Knowledge-Based System Generating Multiple Abstract User Interfaces in UsiXML. In *Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004* (Prague, November 15-16, 2004). Ph. Palanque, P. Slavik, M. Winckler (Eds.). ACM Press, New York (2004), pp. 121–128.
7. Johansson, M. and Arvola, M. A case study of how user interface sketches, scenarios and computer prototypes structure stakeholder meetings. In *Proc. of the 21st British CHI Group Annual Conf. on Human-Computer In-*

teraction BCS-HCI '2007 (Swinton, 2007). British Computer Society, Cambridge (2007), pp. 177–184.

8. McCloud, S. *Understanding Comics: The Invisible Art*. 1994.

9. Meskens, J., Vermeulen, J., Luyten, K., and Coninx, K. Gummy for multi-platform user interface designs: shape me, multiply me, fix me, use me. In *Proc. of the ACM Working Conf. on Advanced Visual Interfaces AVI'2008*. ACM Press, New York (2008), pp. 233–240.

10. Montero, F. and López-Jaquero, V. IdealXML: An Interaction Design Tool-A Task-Based Approach to User Interfaces Design. In *Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI' 2006* (Bucharest, 6-8 June 2006). Chapter 20, Springer-Verlag, Berlin (2006), pp. 245–252.

11. Mori, G., Paternò, F., and Santoro, C. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. *IEEE Trans. on Software Engineering* 28, 8 (2002), pp. 797–813.

12. Pruitt, J. and Adlin, T. *The Persona Lifecycle: Keeping People in Mind throughout Product Design*. Morgan Kaufmann (2006).

13. Schmidt, D.C. Model-Driven Engineering. *IEEE Computer* 39 (2006).

14. Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. EMF: Eclipse Modeling Framework, 2nd Edition. Addison-Wesley Professional, Eclipse Series (2008).

15. Truong, K. N., Hayes, G. R., and Abowd, G.D. Storyboarding: an empirical determination of best practices and effective guidelines. In *Proc. of the 6th Conf. on Designing Interactive systems DIS'2006*. ACM Press, New York (2006), pp. 12–21.

16. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucarest (2008), pp. 1–10.

17. Vanderhulst, G., Luyten, K., Coninx, K. Photo-based User Interfaces: Picture it, Tag it, Use it. In *Proc. of the 4th Int. Workshop on Ontology Content (OnToContent'09* (Vilamoura, November 1-6, 2009).