Domain-Specific Methods and Tools for the Design of Advanced Interactive Techniques

Guillaume Gauffre, Emmanuel Dubois, and Remi Bastide

IRIT – University of Toulouse
118, route de Narbonne
31062 Toulouse Cedex 9, France
{gauffre,emmanuel.dubois,bastide}@irit.fr

Abstract. Novel interactive systems such as Augmented Reality are promising tools considering the possibilities they offer, but no real development methods exist at the moment to help designers in their work. We present in this paper a design method for tightly coupling early interaction design choices and software design solutions. Based on an existing model used for abstract UI design, our work introduces a second model dedicated to the software UI specification and the model-based process used to derive one from the other. To achieve this, we present here a framework based on domain-specific models and transformations to link them and thus support the development process.

Keywords: Mixed Interactive Systems, Model-Driven Engineering, Domain-Specific Languages, Metamodeling, Model Transformations, Design Process.

1 Introduction

In the past 10 years, a new HCI trend has emerged: traditional "Window, Icon, Menu, Pointing device" interfaces tend to be replaced by new forms of interaction that involve physical artifacts, easily manipulated by users. Augmented Reality systems, for example, are interactive systems in which the realization of a physical task is enriched by the presence of digital data and/or services. Tangible User Interfaces and ubiquitous systems are other forms of interactive systems which merge physical and digital worlds. Because they deal with similar concepts and techniques, we group these approaches under the single term: Mixed Interactive Systems (MIS). To support the development of such systems, MIS frameworks have been developed and adopt bottom-up or top-down approaches. Each of them brings consequent advances at different levels of abstraction of the design [7], but interlacing them remains difficult to accomplish, thus limiting the coverage of the development process.

As the use of MIS increases, enhancing the robustness, efficiency and quality of these systems is required. In this sense, elaborating a convenient development process becomes necessary. To cover the different steps of such a process, our approach promotes the results gathered in the early design phases and bridges the gap between the abstraction levels of these results and the implementation. To do so, we articulate

H. Giese (Ed.): MoDELS 2007 Workshops, LNCS 5002, pp. 65–76, 2008.

models to progress along the development process and adopt an MDE approach, thus introducing a Domain-Specific Language [1] for MIS.

2 MIS Engineering Framework

Common processes for HCI development include four phases: requirements gathering, design, implementation and evaluation. Figure 1 presents how our tools cover the first three phases.

Following interviews and observations, task modeling is one of the major tools used to support the requirements gathering. Task models are used to characterize the sequence of sub-tasks with their type (i.e., user's activities, system's activities or interactive activities), the domain objects involved and the events triggered, and to structure these sub-tasks in a hierarchical form corresponding to the global system task.

The design phase can be decomposed into two separate steps: UI design and the related software specification. The former step is concerned with user interaction aspects. It may be linked to requirements gathering by combining users' observation, brainstorming or focus-groups to collect user needs, and an interaction model to organize them according to the specificities of MIS [4]: domain objects description, user abilities, physical and digital artifacts, forms of interaction. During the latter step, design aspects related to the software architecture are considered, using a model dedicated to the description of MIS architectures.

The next step is the implementation of the system by using component-based platforms improving flexibility and adaptability.

Finally an evaluation can be carried out in different ways such as user experiments or ergonomic inspection.

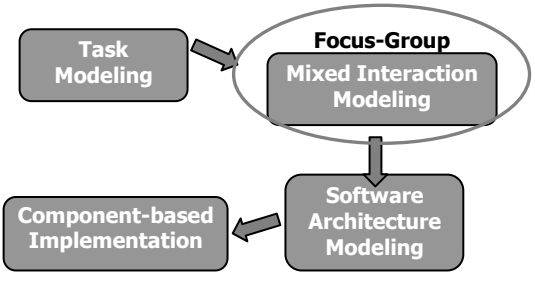

Fig. 1. MIS domain-specific process

At each step, a set of existing models, notations and tools exists: task model in the requirements gathering, dialog and interaction models in the UI design, software architecture and system objects models in the software specification. In this context, rather than modifying the different models involved in order to articulate their usage, we describe a DSL to support this process. Indeed, the current state of the design approach is consistent with two major aspects that are well addressed by an MDE approach:

- Multiple models are required in each phase of the development process and one role of MDE is to "promote models to primary artifacts that drive the whole development process" [1]. MDE will facilitate their articulation and permit the elicitation of coherence rules.
- The MIS domain, with regards to its applications in our every day life, produces emergent systems. Elaborating methods for developing them requires to evaluate the adequacy of models and to support their evolution when required. The MIS domain is in a phase of empiricism and begins to develop theories; MDE will be a powerful support of this evolution.

3 Two Domain-Specific Models

The core of the MIS Domain-Specific Language is based on two models:

- ASUR [4], a model which describes the user's interaction with a Mixed Interactive System. It can be used by itself or as mentioned before, in combination with a focus-group.
- ASUR-IL [4], a complementary model that have been introduced to cover the description of the software decomposition and structure. Its aim is to prepare the implementation step by producing a coherent architecture, promoting the forms of interaction chosen in a technological perspective.

After an overview of the ASUR metamodel in the next section, we present the ASUR-IL metamodel to enable the collaboration of our two domain-specific models.

3.1 ASUR Overview

For a given task, the role of ASUR is to support the description of the physical and digital entities that make up a mixed interactive system and the boundaries among them. ASUR components are *adapters* (A_{In}, A_{Out}) that bridge the gap between both digital and physical worlds, digital tools (S_{Tool}) or concepts (S_{Info}, S_{Object}), the *user* (U) and physical artifacts that are used as tools (R_{Tool}) or objects of the task (R_{Object}).

Components can be interconnected by several kinds of relationships. The most important one, *data exchange*, is used to describe the kind of data that is transmitted. In the physical part, the relationships represent the information channels between components, and in the digital part the way the system treats them. The *representation* link is used to express the fact that two components are two representations (one digital and one physical) of the same concept: this link is characterized in terms of behavior and rendering. Finally, *real associations* express the physical proximity of two physical components and *triggers* represent an action of one component on another. On the basis of previous works in the domain, design-significant aspects have been identified and added to the model as objects attributes: ASUR characteristics improve the specification of components (*perception/action sense*, *location*, etc.) and relationships (*type of language*, *point of view, dimension, etc.*). By analyzing the characteristics of each element, the model supports the predictive analysis of two properties: continuity and compatibility of interactions.

To illustrate ASUR (Figure 2), let us consider a system for 3D object modeling. This system offers, among other features, a dedicated physical artifact for translating, scaling and rotating the 3D object during its edition. This tool embeds a marker for video-based detection of its position and a pressure sensor for switching between each mode (translation, scale and rotation). The physical tool is modeled in ASUR as an R_{Tool}, manipulated by the user. The 3D object is the main digital concept of the task and is modeled as an S_{Object}. The second digital concept is the interaction mode and is typed as an S_{Tool}. Two *adapters* for input (A_{In}) collect data (marker detection and pressure sensor) to control each digital concept. These are in turn connected to one A_{Out} for visual output: the mode is rendered as textual data and the 3D object in a 3D scene.

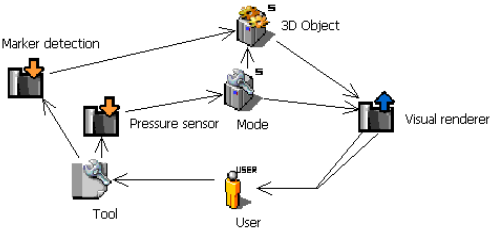

Fig. 2. ASUR model of the 3D object modeler example

3.2 ASUR-Implementation Layer: Towards the Implementation Phase

For each ASUR model, i.e. a given mixed interactive task, an ASUR-IL model is associated. The main contribution of this model is to identify the software components and relationships required to implement this specific task. Only the components involved in the interaction part of the system are described. The description of functional parts of the system is out of ASUR-IL scope. This model is also the frontier between Platform Independent Model (PIM) and Platform-Specific Model (PSM): it describes the software components involved in the task and their communications, the next step being the transfer to a PSM where each ASUR-IL component will be associated to existing software component or new ones.

An ASUR-IL model is an assembly of components which contains two kinds of sub-assemblies: *adapters* and *entities*. Each of them is related to ASUR components (ASUR *adapters* → ASUR-IL *adapters*, ASUR *system* components → ASUR-IL *entities*). Each sub-assembly regroups several components with specific roles in the architecture (*devices*, *APIs*, *models*, *controllers*, *and views*). ASUR-IL *adapters* for input or output, corresponds to the *adapters* in the ASUR model and group *devices* and software libraries (*APIs*), used to connect physical and digital worlds. *Devices* are used to capture/render data from/to the physical world. They can translate physical phenomenon into digital data and vice versa. *APIs* permit to combine several computing facilities to obtain required data: for example, ARToolKit is a specific toolkit for Augmented Reality, which grabs a video frame and produces 3D coordinates of the recognized markers. Therefore *adapters* compose the system part which is likely to be reused: a software implementation of an *adapter* can either exist and satisfy the

ASUR modeling, or be developed on the basis of a combination of existing *devices* and *APIs*.

ASUR-IL *entities* are the other kind of sub-assemblies that make up an ASUR-IL model. They correspond to the digital concepts that are involved during interaction and which are identified in ASUR as S_{Tool}, S_{Object} or S_{Info}. They are triplets of three ASUR-IL components called *models*, *views* and *controllers*, inspired by the MVC decomposition [8]. *Controllers* interpret the physical phenomena and translate data from *adapters* into commands on *model* parts. *Models* are the entry point to the functional core. They are an abstraction of it, enabling the dialog with the application core. Finally, *views* are in charge of the computation required to reflect the state of each digital concept on each output *adapter* connected.

Finally, the relationships named *data flows* connect each component by using the interfaces *port*. The correctness of the *data flow* between two components is ensured by the value given to the attribute *data type* of each *port*.

The ASUR-IL model (Figure 3) that describes the 3D object modeler cited in the previous section is composed of 13 components. A first *adapter* collects the pressure level on the tool using only one *device* component. A second one produces a 4x4 matrix for position and orientation of a marker, captured by a camcorder *device* and computed by the ARToolKit *API*. The last adapter is in charge of rendering the digital concepts, using a screen *device* connected to a window *API*. To render each concept, two *API* components are added: a text field and a 3D canvas.

The two ASUR-IL *entities* follow the MVC decomposition. For example, the 3D object is composed of one *model* which contains the object's characteristics (position, size, etc.). One *controller* transforms a 4x4 matrix into a scale/rotation/translation factor. Finally, one *view* is in charge of inserting the object into the 3D scene by using 3D primitives. The second *entity*, the interaction mode, follows the same decomposition: one *model* containing the three states, one *controller* to convert one level of pressure into one of these three values and a *view* to express the current mode as a string of characters.

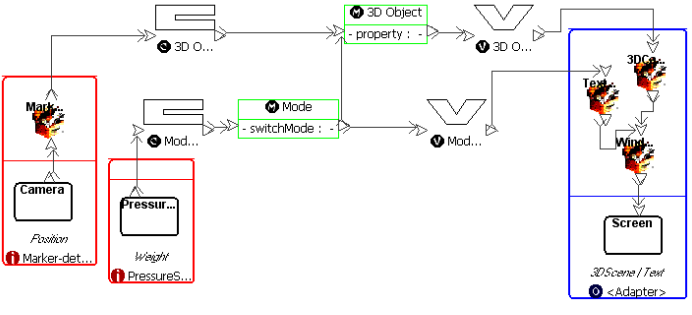

Fig. 3. ASUR-IL model of the 3D object modeler example

3.3 MIS Design Support

ASUR has its own editor: GuideMe [6]. It is a graphical editor which can export diagrams as XML files. After its metamodel was defined [3], a second version of the editor

was developed using EMF to separate graphical editing from model manipulation. As mentioned above, ASUR and ASUR-IL are two models required at different steps of a MIS design process. Other models could also be required such as task models for requirements gathering or system models for functional core specification. To support the integration of our two models and further evolution, we adopt an MDE approach and choose to instrument it with tools from the Eclipse Modeling Project (EMP [5]). This enables the creation of dedicated tools for each model with EMF, GMF, and others. Therefore each model can be edited using the corresponding plug-ins in Eclipse (cf. Figure 4).

Using these tools, the designer can manipulate the two models easily. The main challenge is now to link them by model transformations to rapidly observe the consequences of modifying the description of the interactive situation modeled with ASUR on the software architecture described with ASUR-IL. The next section presents the transformation between ASUR and ASUR-IL and finally introduces the transformation between ASUR-IL and a software component model: WComp [2].

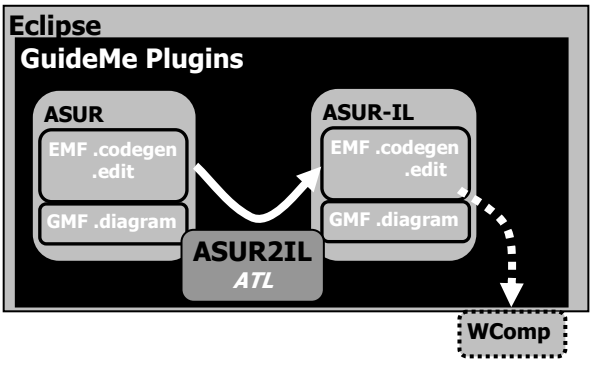

Fig. 4. Tools integration

4 Domain Transformations

In order to implement domain transformations, the Atlas Transformation Language (ATL) was chosen. One of the main reasons is that ATL is now fully integrated with the Eclipse Modeling Project [5] and so ensures complete coherence between the different tools. ATL also provides some precious characteristics for the manipulation of our models: transformation rule inheritance (as class inheritance in object-oriented language) and three ways to define a rule: using a declarative paradigm, an imperative or a mixed one. A model-2-text engine (JET) is also used to produce the PSM for the WComp platform, from the PIM ASUR-IL. The metamodel of the software component model WComp is currently only expressed as code in the platform itself. Thus at the moment, only platform-specific code generation is supported in the framework.

4.1 ASUR2ASUR-IL: Software Modeling Initialization

The goal of this transformation is to prepare the construction of a component-based architecture. ASUR identifies several digital concepts and describes their roles in the interaction: this is the left-hand side of the transformation. On the right-hand side, ASUR-IL is in charge of describing the different kinds of software components involved in the interactive part of the system, with adequate *ports* and *data flows* between them. Practically, the principles of the correspondence between these two parts are well-known, but verbally or textually expressed and not formalized. With ATL, these rules are expressed using a transformation specification language and thanks to the transformation engine, are applied on the models.

Each ATL rule follows roughly the same behavior: the type of each ASUR component plus the characteristics of the relationships between them are identified, and the satisfying rules are applied. It consists, for example, in creating for each ASUR *adapter*, an ASUR-IL *adapter* (Figure 5-1) that contains one default *device* and some *APIs* that account for the kind of interaction modalities described in the ASUR model. The rules include imperative code to interconnect components (Figure 5-3) and to factorize common processes. When ASUR digital components are transposed into ASUR-IL (Figure 5-2), they potentially trigger the creation of multiple *views* and *controllers* after *models* have been created: one *controller* per modality used to interact on the digital component, one *view* per modality used to reflect its state.

This transformation is the starting point of the software architecture design. From the characterization of a mixed interactive situation with ASUR, it produces the basis of the software architecture. It offers to rapidly design the software components structure of a concrete system before starting its implementation. This combination supports the designers during design phases, by linking abstract UI design and software UI specification. Following the transformation, designers can extend the specification

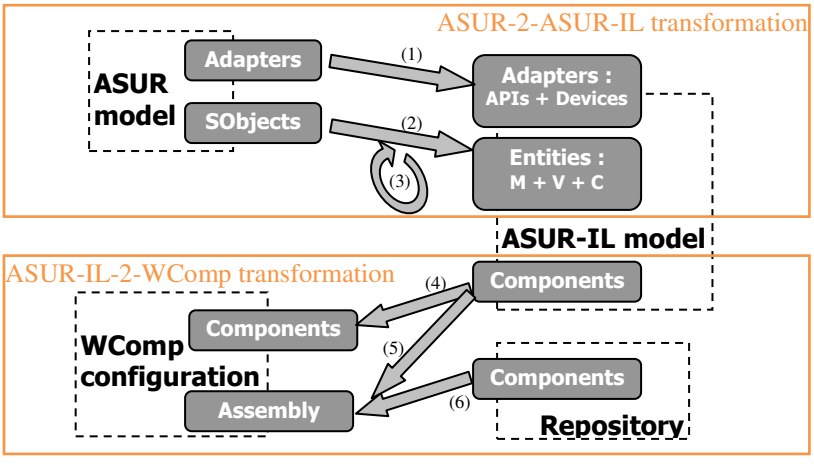

Fig. 5. Specific transformations of MIS process

by additional design decisions, such as the instantiation of other *APIs* or *devices* considering some technical limitations.

Based on this software specification, the next step is defining a platform-specific model of the system. We present in the next section another transformation process to support this final transition.

4.2 ASUR-IL2WComp: Platform-Specific Model Definition

Assuming, that during ASUR-IL editing the designers carefully identified each component of the system, they now must be transposed to the platform model. The currently chosen platform is WComp [2] which is dedicated to rapid prototyping of wearable and ubiquitous interactive systems. Considering these purposes, this platform allows the creation of assemblies of components with a small granularity and the runtime adaptation to the platform context (i.e., low battery level, devices disconnected, etc.). Its flexibility and its simplicity are the major reasons that motivated its use.

The definition of this transformation is on-going work using the model-2-text engine of EMF: JET. It will build the bridge from the PIM (ASUR-IL) to the PSM (an assembly of WComp components), with two goals:

- Creation of software components. It consists in:
  - describing the data manipulated and the associated interfaces (Figure 5-4),
  - identifying an existing software component in a repository (Figure 5-6) that contains previously developed components or standard *APIs* and *devices*,
- Management of the assembly of components (Figure 5-5) to establish the connections between each component in accordance with the ASUR-IL model.

The code required for implementing new components that will be generated by the transformation includes the definition of constructors, interfaces and the common files to generate ready-to-use libraries for the platform. Finally, the assembly corresponding to the system will be expressed as an XML file, in accordance with an XML schema specific to the WComp platform. The generated XML file contains the kind of components to instantiate and the communication channels between each interface.

Once the definition of this set of transformations is complete, our work will provide MIS designers with a range of tools from interaction design to implementation. It will help to rapidly experiment with designed interactive situations from the ASUR results to the WComp assembly of components dedicated to MIS. To illustrate the kind of process it will create, we next describe our tools on a case study.

5 TUI for Museum Exhibitions

The goal of this case study is to design innovative interactive situations in the context of museum exhibitions. Our task is to design solutions promoting knowledge transmission and entertainment in a science museum for particular themes: in this case the evolution of species. By using this approach, we can rapidly experiment with advanced interaction and adapt them to other themes by reusing components.

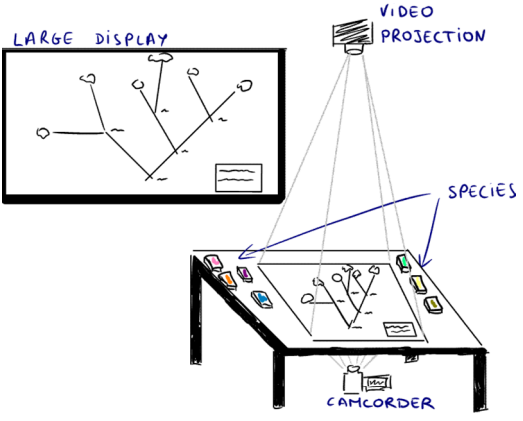

Fig. 6. Schema of the mixed interactive system

The current project aims at proposing visitors to discover species evolution by elaborating an evolution tree based on phylogenetic criteria. Adopting MIS in that context offers the opportunity to keep the visitors away from technologies as much as possible, by letting them manipulate physical objects: visitors thus remain focused on the content and are not impressed or afraid of the use of technologies such as mouse, keyboard, complex 3D devices, etc. Using MIS also increases the visitor's experience by adding digital rendering (video, 3D, sound, etc.). To elaborate the evolution tree, the user manipulates physical representations of species (a frog, a crocodile, etc.) to add them to the tree which is rendered by video on the interactive space with related phylogenetic criteria (Figure 6).

The first solution (Figure 7) uses marker-based detection to capture tangible objects (species) and visual rendering to report the data. These two facets of the interaction are described by three *adapters* on the ASUR model. The first one, marker-detection, is able to determine the position and orientation of each physical component representing species. The second one is capable of visually rendering the state of each digital component of the system. Another *adapter* for output is used for displaying the evolution tree to the rest of the audience using a large display. When describing with ASUR the task of inserting a species in the tree, an ASUR *system* component is identified to depict the digital object that includes the characteristics of the selected species. A second *system* component is required to depict the digital information related to the hierarchical classification of the species: this is a second digital concept manipulated in this task. These two digital concepts are thus connected to the three ASUR *adapters*: these relationships express the fact that information captured by the *adapter* for input (the camcorder) will affect the two digital concepts and that these two digital concepts are also affecting the *adapters* for output (namely the video-projection and the large-display).

Figure 8 shows the ASUR-IL model resulting from the asur2asur-il transformation. Each *adapter* has been translated into an ASUR-IL *adapter*, combining a default *device* connected to one *API* component which will be used to adapt the data emitted or needed by each device. In this case, the localization of each physical object

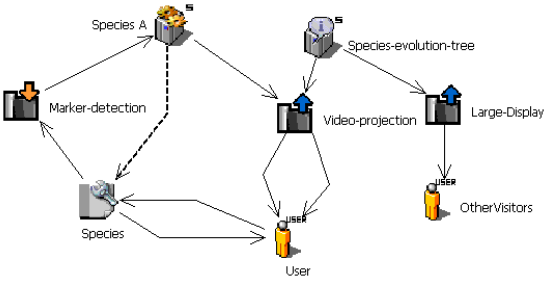

Fig. 7. ASUR model for evolution-tree construction

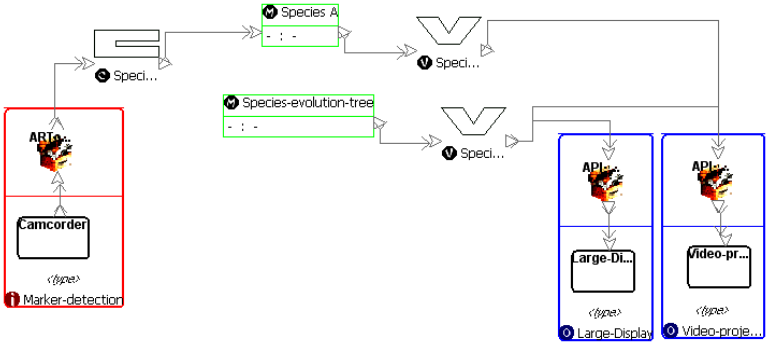

Fig. 8. Asur2asur-IL transformation result

representing one species will be made using a camcorder, producing a picture used by the *API* ARToolkit to obtain the 3-dimensional coordinates.

For each ASUR digital component, an ASUR-IL entity is created with the correct amount of *controllers* and *views* depending on the number of modalities used during the interaction. In this case, only one *controller* and one *view* are necessary for the interaction with the species, and only one *view* is used to render the evolution tree (same modality on each *adapter*: video-projection and large-display). The core behavior of each digital concept will be implemented in the *model* components, and the interaction with them will be coded into *controllers* for input and *views* for output.

To illustrate the dependencies between the two models, we focus on the case where the system also provides vocal feedback when selecting a species. This way, the user gets a description of the selected species. It results (Figure 9) in the addition of an A_{Out} in the ASUR model, an *adapter* for output corresponding to Voice synthesis, and its translation to the ASUR-IL model. The transformation will produce another *view* component for the species because of the two modalities used.

Once the architecture is designed, the next step is to use the ASUR-IL model for the implementation of the system on the WComp platform, a .NET platform using C# code. This transformation will generate component skeletons, such as interfaces, constructors and parameters, to be loadable into the platform. This is the behavior for

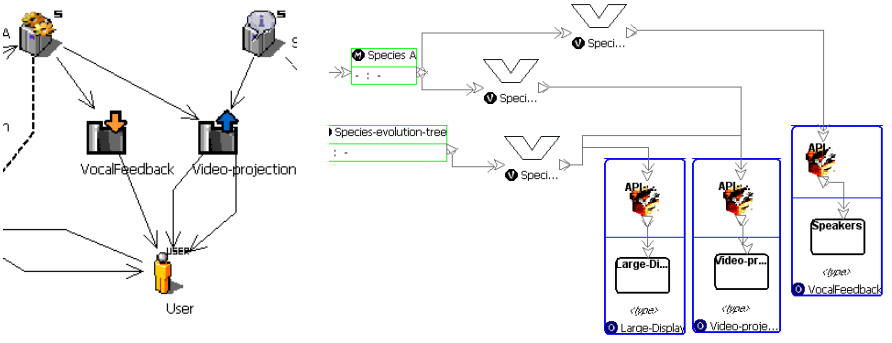

Fig. 9. Model evolution

novel components. The other choice is to specify a component that has already been developed and is described in a repository. Following this step of choosing or generating components, an XML file will be generated containing the assembly description of the system, used by WComp to run it. In the example, the components for ARToolkit *API* and camera *device* but also the frame component based on the *device* for video-projection, have been yet developed. Finally only *entities* and PiccoloCanvas components must have to be developed on the platform WComp.

6 Conclusion and Future Work

This work is a step toward the definition and instrumentation of a design process for Mixed Interactive Systems. This process will permit us to increment on the designed solution until obtaining a convenient degree of usability. The advances presented here, ASUR-IL model and related transformations, offer rapid navigation between the abstract design of innovative interaction techniques, expressed with ASUR, their concrete specification, expressed in ASUR-IL, and the final realization corresponding to their implementation by a WComp assembly. The Domain-Specific Language developed is an efficient tool for promoting the characteristics issued from the user-centered design into the crucial phase of implementation. As this approach uses models as primary artifacts, thanks to the MDE tools, each level of abstraction defined in the development process embeds properties standing for the usability of the interactive system.

The ASUR model defines some properties related to the quality of the interaction between a user and a mixed environment. Our goal is to integrate them throughout the entire process, to finally evaluate their evolution during each cycle of the process. Further work will aim at identifying additional properties, relevant at the software design level (ASUR-IL) such as computing time or hardware constraints, and structuring their impacts on the remaining design steps of our process. It will increase the ability to evaluate the quality of each interactive situation.

Another perspective is to study the feasibility of reverse transformations between each step and their impact on the higher levels of abstraction. A modification of a

WComp assembly (choosing one device instead of another) could be evaluated at the ASUR level to determine the consequences of such choices.

Finally, we focus here on specific models for MIS. To make possible the development of concrete systems, other aspects could be included: collaboration with business models for the connection with the functional core, interactive modalities ontology to support the choice of specific devices and APIs, and also description of the behavior of the components using dialog models for example (State charts, Petri nets, etc.).

As already mentioned, the MDE approach is very helpful to articulate and transform models. However, it appears that designing MIS may rely on a lot of models and maintaining the coherence among all of them may be difficult. The management of this combination of models and transformations needs to be investigated to better assess the usability of the MDE approach for a MIS development process.

References

[1] Bézivin, J., Jouault, F., Kurtev, I., Valduriez, P.: Model-based DSL frameworks. In: 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, Portland, USA, pp. 602–616 (2006)

[2] Cheung, D.F.W., Tigli, J.Y., Lavirotte, S., Riveill, M.: WComp: a Multi-Design Approach for Prototyping Applications using Heterogeneous Resources. In: Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping, Chania, Crete, pp. 119–125 (2006)

[3] Dupuy-Chessa, S., Dubois, E.: Requirements and Impacts of Model Driven Engineering on Mixed Systems Design. In: Gérard, S., Favre, J.-M., et Xavier Blanc, P.-A.M. (eds.) Proceedings of the conference IDM 2005, Paris, France, pp. 43–54 (2005)

[4] Dubois, E., Gauffre, G., Bach, C., Salembier, P.: Participatory Design Meets Mixed Reality Design Models. In: Conference Proceedings of Computer Assisted Design of User Interface (CADUI 2006), Bucarest, Romania. Information Systems Series, pp. 71–84. Springer, Heidelberg (2006)

[5] Eclipse modeling Project, http://www.eclipse.org/modeling/

[6] GuideMe, http://liihs.irit.fr/guideme

[7] Hampshire, A., Seichter, H., Grasset, R., Bilinghurst, M.: Augmented Reality Authoring: Generic Context from Programmer to Designer. In: Proceedings of the 20th conference CHISIG of Australia, OZCHI 2006, pp. 409–412. ACM Press, Sydney, Australia (2006)

[8] Krasner, G.E., Pope, T.: A cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk-80. The Journal of Object Oriented Programming, 26–49 (1988)