Model-Driven Development of Interactive and Integrated
2D and 3D User Interfaces using MML

Andreas Pleuss∗
Lero, University of Limerick, Ireland

Denis Gračanin†
Virginia Tech

Xiaoyu Zhang‡
Virginia Tech

Abstract
While there is a lot of research done in the area of 2D or 3D user interfaces (UIs) construction, comparatively little is known about systematic approaches to designing and developing integrated 2D/3D UIs and applications. The previously developed Multimedia Modeling Language (MML) provides a top down approach for a model driven development of 2D/3D UIs and applications. The MML Structure Model and Media Components provide support for including X3D based content and automatic generation of application skeletons. We use a work instruction manual for a woodchipper as an example to illustrate how to apply MML. We discuss the ramifications of this approach and opportunities for some improvements.

CR Categories: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities; I.3.6 [Computer Graphics]: Methodology and Techniques—Languages

Keywords: Model-based development, 2D and 3D user interfaces, application development

1 Introduction and Motivation
There has been a noticeable increase in the level of perceptual information on the Web through leveraging the ability to connect various media types, including 2D and 3D content. Presented together, abstract and perceptual data complement each other by clarifying a concept or problem. The mixture of the two does speed up the learning process and promotes retention. Composing a complex mix of 2D, 3D and other media content presents a significant challenge. The previous work has focused on constructing either mostly 2D user interfaces (UIs) or mostly 3D UIs [Dachselt et al. 2002; Vitzthum and Pleuß 2005; Zhang and Gračanin 2007].

An example of a mix of abstract and perceptual information in a Web page is shown in Figure 1. It is a 3D manual that uses X3D to display a 3D mechanical model while the 2D section shows textual data associated with different parts of the model. The instruction steps for the device assembly are animated in the 3D model and described in text. Hovering the mouse pointer over a part of the 3D model causes the application to highlight the corresponding textual data. Clicking the part will replace the textual content with more detailed information. Selecting an instruction step in the 2D interface causes the 3D animation to jump to this step. This mixed media integrated 2D/3D view of a mechanical object allows the user to navigate around the object, getting a spatial understanding of it, while also getting additional textual information. Integrating 2D UI elements in a Virtual Environment (VE) can provide helpful information [de Haan et al. 2006]. However, some usability challenges must be addressed to provide better user experiences.

The UIs for VEs have different characteristics compared to the traditional UIs. Traditional UIs follow an existing design standard, WIMP (Windows, Icon, Menu, Pointer), while VE UIs lack such standard. VE UIs have higher degree of interactivity and exhibit continuous input/output “exchange” which occurs concurrently. Large number of different tools and presentation formats make the situation even more challenging. New abstractions and approaches for describing and implementing these UIs are needed.

Most of the tasks performed within a VE are application-specific and difficult to model. However, there are some basic VE interactions tasks. The majority of VE interaction tasks fall into these task categories: navigation, selection / manipulation, symbol input and system control. The space in a VE is a 3D content constructed using a number of virtual objects. The state of the VE is determined by the state of each object. Besides the 3D content description, 3D file formats typically have an inherent event model.

X3D is an Extensible Markup Language (XML) based standard that is emerging as the de-facto open standard for cross-platform, inter-application 3D content delivery (http://www.web3d.org). The X3D interactive profile provides basic interaction with a VE through various sensor nodes for user navigation and interaction, enhanced timing, and additional lighting. X3D can be used to construct interactive 3D applications, either stand-alone or web-based. A component based approach, used in the CONTIGRA architecture [Dachselt et al. 2002] is based on declarative XML documents describing the component implementation, interface, configuration and composition of 3D UIs and VEs. In [Dachselt and Rukzio

∗e-mail: andreas.pleuss@lero.ie
†e-mail: gracanin@vt.edu
‡e-mail: zhangxy@vt.edu

Figure 1: Taken from Cortona3D RapidWorkInstruction Woodchipper demo (http://www.cortona3d.com/)
2 Proposed Approach

The Multimedia Modeling Language (MML) [Pleuss and Hussmann 2011] supports a model-driven development for interactive applications with 3D and multimedia content. MML provides five kinds of models. The Task Model specifies the user’s tasks from the viewpoint of human-computer interaction. It is the most abstract model that is not used directly for code generation. Instead, it is used as the starting point for other models.

The Structure Model specifies the Domain Classes (UML Class Diagrams) and Media Components for various media types. A Media Component encapsulates the media content together with the basic animation and rendering functionality. Possible Media Component types include Audio, Video, 2D Animation, 3D Animation, etc.

The Scene Model specifies the navigation between Scenes in a way similar to UML State Charts. The MML Scene is an abstraction of the UI area on a screen. It can be parameterized and dynamic.

The Presentation Model specifies the UI structure in terms of abstract UI elements (e.g., input, output, and selection). The UI elements can be realized by instances of the Media Components from the Structure Model.

The Interaction Model is denoted as an extended UML Activity Diagram and specifies the interplay between the UI elements (including media instances) and domain classes, e.g., which event causes which operation call.

2.1 MML Modeling Process

The overall MML modeling process conducted during the design phase (after the requirement analysis) of an application is shown in Figure 2. The Task Model captures the required user tasks (derived from the requirements) from the UI design point of view.

In the Structure Model, the Domain Classes are derived using conventional object-oriented design process. The Media Components are also derived from the requirements and they relate to the Domain classes through Media Representation relationships. These relationships may, in turn, require additional Domain Classes.

The Scene Model includes one or more Scenes and describes the navigation between the Scenes. The Scenes can be identified based on the Task Model. Each Scene can provide 2D or 3D content. Thus, the Scene Model supports integrated 2D/3D content.

The Presentation Model is defined by the UI designer for each Scene. It contains the Abstract Interaction Objects (AIOs) derived from the Task Model or provided by the designer. The AIOs are associated with the corresponding Domain Objects, the instances of the Domain Classes in the Structure Model. The Presentation Model is then complemented with Media Instances and Sensors. The Media Instances correspond to the Media Components in the Structural Model. This may require creating additional Media Components and Domain Classes.

The Interaction Models specifies how UI events and Sensor events trigger operation calls on the Domain Objects. Additional model elements (e.g., Scene objects) can be identified and added.

2.2 Code Generation

MML is implemented with the Eclipse Modeling Framework (EMF) (http://www.eclipse.org/emf/) and supported by a visual modeling tool based on Magic Draw (http://www.magicdraw.com/). Using the EMF-based format it is possible to generate code skeletons for different target platforms using model transformations specified in the Atlas Transformation Language (ATL) (http://www.eclipse.org/m2m/atl/). Currently supported target platforms include Flash, Java, SGV/JavaScript, and Flash Lite for embedded devices.

The most mature implementation is the transformation for the target platform Flash. The generated application (skeleton) consists of Flash documents and code in object-oriented language ActionScript. The Domain Classes are mapped to ActionScript code where the bodies of operations have to be completed manually (i.e., programming of the application logic using ActionScript). The Media Components are mapped to skeletons which have to be filled out with the concrete media content. Each Scene from the MML Scene Model is transformed to a Flash document, which provides the UI (i.e., a view in terms of the Model-View-Controller or MVC pattern) and an ActionScript class acting as controller (in terms of MVC). The view is generated according to the Presentation Model including instances of the Media Components. The controller class realizes the navigation between the Scenes according to the Scene Model and the event handling according to the Interaction Model.

The generated application skeleton can be directly executed, e.g., to test the navigation between the Scenes, but does not contain application logic or media content yet. The skeletons can be directly loaded into the authoring tool to fill out the placeholders for media content, implement the domain class operations, and to finalize the visual appearance of the UI. In this way, strengths of both, models and authoring tools, are combined. However, while MML supports 3D content as well, the work presented in this paper is the first extensive effort in using MML to integrate 2D and 3D UIs and support 3D standards such as X3D.
3 An Illustrative Example

Figure 1 shows an example of a class of applications that have mixed 3D and 2D content. This is a demo of digital interactive work instructions created by Cortona3D RapidWorkInstruction toolkit (http://www.cortona3d.com/). The demo provides work instructions for a woodchipper, more specifically for a radiator mount. The available parts include the radiator, the radiator mount, bolts, etc. We use this class of application to demonstrate the application of MML for mixed 3D and 2D content.

We start modeling the MML Structure Model shown in Figure 3 by specifying the Domain Classes. In the example, a Device consists of multiple Parts and an Instruction of multiple InstructionSteps. An InstructionStep refers to several parts required for this step. We chose here rather generic Domain Classes to support multiple devices and instructions (of course, the choice is up to the modeler).

A Domain Class can be associated with one or more Media Components to indicate that it is represented by them. In the example, the classes Instruction, InstructionStep and Part are represented by a textual description (Media Components PrerequisiteDescription, StepDescription, and PartDescription of media type text). In addition, the Part and the Device are represented by 3D animations Part3D and Device3D. The Media Components are modeled in a generic way at the class level (not the instance level) which allows us to dynamically change the concrete media content at runtime, e.g., by loading them from a repository. A Media Component already encapsulates some basic standard functionality to manipulate and to animate/render the content. Therefore, some standard operations are defined in MML (like play() or goToAndPlay() for temporal media and setX(), setY() for spatial media, see [Pleuss and Hussmann 2011]) which can be used later in the Interaction Model.

MML supports the definition of the inner structure of Media Components. Therefore, MML provides different kinds of Media Parts whose type depends on the media type. For 3D, these are Object3D, Transformation, Viewpoint, Light, and Camera. These abstract types can be mapped onto specific 3D formats, such as X3D. They can be nested in a tree hierarchy in the same way as in 3D scene graphs or in SSIML [Vitzthum and Pleuß 2005]. However, as MML is not intended to generate the complete media content, the model abstracts to those elements which are relevant for the integration with other application parts (e.g., elements which should be modified from outside). An inner element of a Media Component is specified as an Inner Property which in turn is defined by a name and its type of Media Part. Like UML class properties, the Inner Properties reside on “role level”, i.e. provide a placeholder for concrete instances and can have multiplicities. Optionally, an Inner Property can be “typed” with another Media Component (like part:Part3D) to indicate that it is an instance of that Media Component. If a multiplicity is specified, the single instances can be accessed (in the generated code and the MML Interaction Model) using an array-like notation, e.g., “part[1]”.

In our example, Device3D consists of multiple instances of Part3D. Those are modeled as a group partGroup as for each part there is also a viewpoint (partViewpoint) and a transformation to animate the part (partPositioning). In addition, there are multiple Cue Points to control the animation. A Cue Point in MML represents a point on the timeline of a temporal media component allowing the application, e.g., to jump to a Cue Point or to throw an event if a Cue Point is reached. Here, there is one Cue Point for each instruction step to synchronize the textual description of a step with its animation.

The Media Components as specified in MML do not provide statements about the concrete media content. For instance, Part3D can be realized by different 3D artifacts, e.g., for the radiator, the radiator mount, the bolts, etc. Those are called Media Artifacts in MML and can be optionally specified (if already known at development time) by either annotating the Media Components or in an additional table (not shown here). It is important to notice that the Media Artifacts and instances of Media Components are two different concepts, as each artifact can still be instantiated multiple times; for instance, multiple bolts in the woodchipper example.

The next step in the MML modeling process is defining the Scene Model. In our example there is only a single Scene WorkInstructionsMain representing the application’s main window, so we do not further discuss the Scene Model here. The Presentation Model specifies the UI for each Scene. Figure 4 shows the Presentation Model for WorkInstructionsMain. The example application (Figure 1) provides four 2D views and another view for the 3D content (called assembly in the model). Each view is modeled by a UI Container (abstraction of, e.g., a panel or a tab) containing AIOs. The AIOs in the example are all either output elements which display data to the user (like prerequisites or currentStep in Figure 4) or selection elements which display a set of data from which the user can select (like instructionSteps or parts).

A Scene also contains Domain Objects (the Instruction, the Device, and currentStep) which are instances of the Domain Classes. Each AIO represents a Domain Object or one of its properties or operations. For instance, the output element currentStep represents the Domain Object with the same name while requiredParts represents the property requiredParts of currentStep (denoted as annotation of the relationship). Finally, there are media instances which partially realize the AIOs (indicated by a dotted arrow). For the code generation this means that the AIO is replaced by this media instance. The AIOs not realized by a specific media instance are realized by standard widgets. In the example, all AIOs are realized by media instances, e.g., the AIO parts in the bOM view by the partDescription and the AIO parts in assembly (i.e. the 3D view) by part:Part3D.
To enable interaction with the media components, the instances can also be equipped with Sensors which represent the same concept as sensors in a 3D domain (in MML this concept is also available for other media types like 2D animations). MML supports the Sensor types collision, touch, proximity, visibility, and time. In the example, part:Part3D is associated with a Touch Sensor to allow selection of parts in the 3D view.

During code generation, each Scene is mapped to a class and the contained instances to class properties. Their initial values are either taken from parameters passed to the Scene or can be defined in the model using an additional instance model. Defining an additional instance model can be particularly interesting when specifying complex 3D content at the concrete level. Figure 5 shows the instance Model of device3D for the woodchipper example having the multiplicities resolved and artifact names specified. Several artifacts are used multiple times as the woodchipper consists of, e.g., multiple Bolt elements. An alternative notation for the Instance Model (which can be more appropriate for large models) is a textual table listing the names of instances and artifacts.

Finally, the MML Interaction Model specifies (in terms of an adapted UML Activity diagram) the interplay between the defined elements to realize the UI behavior. Figures 6 shows an example of the interaction when the user selects an instruction step using the selection element instructionSteps in the view document. If a selected event occurs the selected object is set as currentObject in the Scene and the AIOs representing them (currentStep and requiredParts in instructions, see Figure 4) are updated accordingly.

In addition, the 3D animation should show the selected animation step. Therefore, the Cue Point associated with the instruction step has to be retrieved (using the generated operation getStepAnimation() and the 3D animation has to be set to that Cue Point (using the MML standard operation gotoAndPlay(cuePoint)).

Figure 7 shows event handling for the 3D view: if a part is selected in the 3D view (Touch Sensor SelectPart), the corresponding Domain Object is achieved and all associated AIOs are highlighted.

4 Conclusion

In this paper we presented an approach to automatic generation of integrated 2D/3D UIs and related application code skeletons. The approach is based on the MML and includes X3D for the target 3D content format. An arbitrary number of 2D and 3D UI elements can be integrated into a single, dynamic and parametrized MML Scene. The current implementation supports automatic generation of the skeleton code using Adobe Flash or Java as a target.

Acknowledgements

This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero — The Irish Software Engineering Research Centre (www.lero.ie).

References


