Modeling the User Interface of Multimedia Applications

Andreas Pleuß

Institut für Informatik, Ludwig-Maximilians-Universität München
Munich, Germany
andreas.pleuss@ifi.lmu.de
http://www.medien.ifi.lmu.de

Abstract. Multimedia applications are a branch of software development with growing importance. Typical application areas are training applications and simulations, infotainment systems - e.g. in cars - or computer games. However, there is still a lack of tailored concepts for a structured development of this kind of application. The current paper proposes a modeling approach for the user interface of multimedia applications with the goal of a model-driven development. We identify the special properties of multimedia application development and the resulting aspects to be covered by the user interface model. Existing conventional user interface modeling approaches are not sufficient, as they do not cover the media-specific aspects of the application. However, a multimedia application usually includes conventional user interface elements as well. Thus, we first propose a solution for the media-specific part. Second, we elaborate an integration of our approach with existing conventional approaches. Finally, we discuss the overall model-driven development approach and outline its benefits.

1 Introduction

Multimedia applications are an application domain with still growing importance. In typical application areas, like training and simulation software or computer games, the intensive usage of multimedia is already established since many years. Additionally, in the last years the production of sophisticated user interfaces became more and more common even in other applications areas. Often cited examples are information systems with an emphasis on a pleasing and entertaining user interface, so called infotainment systems. Modern cars contain such applications to provide integrated access on the car's entertainment, comfort and navigation functionality. For a classification of multimedia applications see e.g. [1], [2].

The extract of the most common definitions of the term *multimedia application* postulates an interactive application integrating at least a temporal and a discrete media type. *Discrete media* refers to media which does not change over time, like a still image, while temporal media is time dependent like audio or video. As today a high amount of software is compliant to this definition we

restrict this for the purpose of this paper: First, the usage of media objects is a core feature of the application, including also complex media types like video and animation. (Animation refers here to graphics which changes over the time in any way). Second, the application provides sophisticated interaction associated with application logic. Purely document-oriented software, i.e. a (static) hypertext document, is not in the main focus of this paper.

The development of multimedia applications is characterized by the integration of knowledge, tools, and experts from two different areas: software engineering and media design. While requirements analysis is performed analogously to conventional software and the implementation phase is supported by powerful multimedia authoring tools (e.g. *Macromedia Flash* [3]), there is still a lack of concepts to bridge the gap between analysis and implementation. Current multimedia development methods, like described in [4], use mainly informal methods with emphasis on media production and design. Concepts for the structured integration of the application logic and the consideration of software engineering principles are still missing, although they are heavily claimed by various research contributions like [5], [6], [7], [8]. This results in badly structured applications, where maintenance and extension requires exceeding effort, although changes of requirements are also common for this type of application [9]. Common software engineering methods, like UML-based design methods, are not sufficient, as they do not cover media integration and user interface design [2], [10], [11].

Early approaches to address this problem focus on single specific aspects of a multimedia application, in particular synchronization (see [12], [13]), or very specific application domains like [14]. The first comprehensive approach is *OMMMA* (*Object-oriented Modeling of Multimedia Applications*, [10], [15]) which provides a design model for multimedia applications based on UML. Based on OMMMA, we propose in [16] further refinements and enhancements enabling a model-driven development of multimedia applications.

The current paper continues the research in [16], where we described an overall frame for model-driven development of multimedia applications and the relationships between the different views of a multimedia application model. These views are the static structure (i.e. the domain model), the user interface, the interaction, and the overall temporal structure (i.e. coarse-grained program flow). The current paper takes one of these views – the user interface, which is probably the most important and extensive view – and goes into the details.

We first discuss in detail the required aspects to be covered by a user interface model for multimedia applications. While existing work focuses only on the media-specific user interface elements, we take into account, that usually also conventional user tasks are part of a multimedia application. Thus, we consider in addition the existing task-based user interface modeling approaches for conventional widget based applications. On that base we first propose a detailed and platform-independent modeling approach for the media-specific modeling part. Afterwards, we elaborate the integration with the required parts from conventional task-based user interface models. Finally, we discuss the resulting overall model-driven development approach for multimedia applications.

The paper is structured as follows: section 2 introduces conventional approaches for user interface modeling. In section 3 we discuss in detail the requirements for multimedia applications and how they affect the concepts from conventional user interface modeling. On that base, we discuss in section 4 the required model elements for the media-specific part and propose a notation. In section 5 the media-specific part of section 4 is integrated with the required elements from conventional task-based modeling. We discuss the overall model-driven approach in section 6. Section 7 provides the conclusions and the outlook.

2 Conventional User Interface Modeling

There is already a good understanding of the basic principles for modeling user interfaces. This section briefly describes the basic concepts based on the detailed overview provided in [17], followed by the introduction of a concrete approach, *UMLi* [18], which we use as base for the further work in this paper.

First work on user interface modeling started already in the 1980's, e.g. [19]. The main problem of early approaches was that they emerged either from the engineering domain or from the UI designer domain (see [11]) – a similar problem as addressed in this paper for multimedia applications. Advanced approaches integrate the both views. They usually base on a conventional domain model, like UML class diagrams, as well as on a task model, like *ConcurTaskTrees* [20]. The task model has its origins in the human-computer-interaction community. It represents the user's tasks and decomposes them hierarchically into subtasks, down to primitive actions on the user interface. Task and domain model are usually modeled during analysis.

The *abstract UI model* describes the user interface in an abstract and platform-independent way. It consists of three different kinds of elements: *Abstract interaction objects* allow primitive user actions like invoking an action or selecting an element from a list. *Information elements* present information to the user, which can be either from the domain model or additional information like a label text. Interaction objects and information elements are assigned to *presentation units*, which represent an abstraction of windows on the screen. The elements are derived from the task model and are related to the domain model either by invoking actions or by presenting or manipulating information.

Finally, a *concrete UI model* is derived from the abstract UI model. It contains concrete user interface elements, usually widgets, and their concrete layout. Often the concrete UI model is realized by specialized implementation tools like user interface builders. The transformation from abstract to concrete UI model can be done semi-automatically, e.g. rule-based like in [21].

In the following we briefly sketch a concrete approach, *UMLi* [18]. It applies the mentioned principles of UI modeling and realizes them as an extension of UML. As today UML is a de-facto standard and widely understood, we use this approach for the further work in this paper.

A presentation unit in the abstract UI model in UMLi contains the following elements: *inputters*, which receive information from the user, *displayers*, which

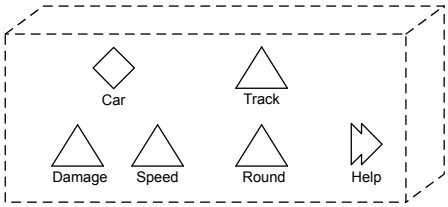

Fig. 1. UMLi diagram for the racing game example

provide information to the user, *editors*, which are simultaneously inputters and displayers, and *action invokers*, which receive events from the user (e.g. like a button).

Figure 1 shows an UMLi diagram for the race screen of a racing game application. We use this example, as on the one hand it demonstrates all characteristics of a multimedia application and on the other hand its requirements are easy to understand without specific domain knowledge. During the race the user has to steer the car over the track. In addition to the car and the track the application displays information about the current status, like the car's speed and damage, and the number of completed rounds. Moreover, the user can leave the race to view the application's help. The UMLi diagram consists of a presentation unit representing the race window (dashed lines). It contains the required user interface elements. To display information about the car status and the current round, we use displayers (represented by upward triangles). For the invocation of the help window we use an action invoker (represented by the semi-overlapped triangles). As the user constantly modifies the car's position and orientation, we decide to represent it by an editor (notated as a diamond) and the track by a displayer. (Another decision is also possible, as actually the car stays always in the center of the screen while the track moves, but probably such decisions belong rather to the concrete layout than to the task-based abstract layout modeled with UMLi).

The behavior of the user interface elements, i.e. the dialogue, is modeled in UMLi using UML activity diagrams. To describe the relationships between actions and user interface elements they use Object Flows. The difference between plain UML and UMLi activity diagrams are stereotypes which mainly aim for a more compact notation of constructs frequently occurring for user interfaces.

3 Required Aspects to Model

The foregoing section describes the established concepts for modeling a user interface. The focus of those approaches lies only in user interfaces for "conventional" applications, like database applications. Their user interface objects are restricted to standard objects, usually widgets, as explicitly stated e.g. in [17], [18]. As this focus is not sufficient for multimedia applications we discuss in the following section the required aspects for this specific domain.

Section 3.1 discusses the aspects directly related to the heavy usage of media objects. On that base, section 3.2 comes back to the aspects of conventional task based user interfaces and examines whether and how they change in multimedia applications.

3.1 Multimedia-Specific Aspects

The requirements are derived from the existing multimedia related modeling approaches described in section 1, as well as from the methods and artifacts mentioned in multimedia development related work like [4], [9].

Integration of specific media types. The core characteristic of a multimedia application is the integration of different media types. The choice of a media type specifies, on which perception channels information is presented to the user and how the user can interact with it. Thus, it is a fundamental decision which media type is used to achieve the optimal transmission of a given piece of information to the user. Often the choice of media type is a basic requirement from the customer, e.g. in a medical training application the customer may postulate to have a human organ presented by a 3D animation. It may also be possible that the choice of media type is already obliged by circumstances, like restriction of resources or the media objects available from third parties.

The usage of the complex media types should also be specified as soon as possible within the development process, as the production of media content often takes much effort and time. Dependent on the chosen type of media, the respective experts and tools need to be available.

The media type affects the other parts of the application, e.g. because of its specific interaction behavior. In summary, the model should allow to specify which concrete media objects to use in the application. We propose a solution for this requirement in section 4.

Inner Structure of Media Objects. Complex media types often consist of several sub-objects. For example an animation representing a car may consist of sub-animations for doors and wheels, which may move independently. Besides, also the whole car can move. Dependent of the sub-objects, they contain sub-objects themselves.

A user interface model should allow specifying such an inner structure, when the sub-objects have to be accessed by the program code of the application. On the one hand, the media designer has to take into account which parts of a media object should be accessed from code. Typically these parts are designed as sub-components. In the example above typically the wheels themselves are also designed as animations contained within the car animation. On the other hand, the application code programmer has to know how to access the required parts, e.g. their name and their path within the hierarchy of sub-objects. In summary, the specification of the inner structure of media objects is required to define the interface between application code and media objects and should

thus be supported by the modeling approach. We propose a solution for this requirement in section 4.

Spatial User Interface Layout. An issue to discuss is the spatial user interface layout. A vision of the layout can be the basic motivation for the whole multimedia application. It can be the core of the customer's requirements. For example a customer has the idea of a racing game application, where the screen shows a view on the track and in the foreground a specific instrument board.

On the other hand, it is contentious whether to include the spatial user interface layout into a model. Clearly, the structure of the user interface is important, e.g. which window contains which elements. This aspect is already covered by conventional user interface models in an abstract way. Spatial layout would add information about the size and position of elements. But this information can lead to platform dependent models, as the screen size of the target platforms can change significantly. Moreover, as demonstrated by conventional user interface models, interaction objects are implemented for different platforms using different widgets or probably different modes (e.g. on some devices auditory instead of graphical messages).

Moreover, it is not clear, whether the (semi-)formal specification of size and position adds valuable information to the model. To capture the vision of a screen layout, usually no exact and absolute values are required. Quite contrary, the final pixel precise adjustment is better performed in the implementation tool anyway. To sketch the idea of the layout, informal and quickly to handle methods are more suitable.

In summary, there is no urgent need for specifying the spatial layout in the model. As it usually adds no further (formal) information, it can be viewed as an additional optional view on the existing model. This additional view can be addressed e.g. by layout sketches. The quick creation of optional layout sketches can probably also be supported by an advanced modeling tool.

Synchronization. The temporal behavior of different time-dependent media objects can be related to each other. An example is an animation which should be synchronous with sound or a video which should start after another video has finished. Such synchronization issues often affects other parts of the application – e.g. other media objects or the program code – and should therefore be part of the model.

In the meanwhile, UML offers various mechanisms to model temporal behavior. Also activity diagrams, used for behavior modeling in UMLi, are suitable to denote the order of media objects and whether they can be interrupted by events. UML 2.0 also enables an advanced specification of temporal constraints.

3.2 Task-Related Aspects

Based on the media-specific requirements, the following section discusses the consequences for the conventional task-based user interface elements from section 2.

Interaction Objects. Media objects can act as interaction objects, e.g. the user can click on an animation. However, there is still the need for conventional interaction objects. A multimedia application usually contains also conventional tasks. For example in the racing game application the user should be able to input his name, invoke the help or cancel the application. Such tasks are often outside the customer's media-specific vision of the application. They also usually require no specific media-type. In that case, conventional abstract interaction objects are useful to stay independent of the target platform and even the modality. The interaction may even be without any graphical design, e.g. just pressing a key on the keyboard. The designer should not be forced to make such decisions when it is not mandatory.

Another issue is that time-dependent media objects can invoke actions independently from the user. They can trigger time-related events, e.g. when they are interrupted or have finished. Dependent from the media type additional events are possible, e.g. a moving animation can trigger an event when it touches another animation or reaches a specific region on the screen.

In summary, abstract interaction objects should be part of the model. As media objects may also act as interaction objects, the model must integrate these two kinds of elements. Additionally, it must be considered that media objects can invoke actions independently from the user. We propose a possible solution in section 5.

Information Elements. All media objects present information to the user, which can be static or derived from the domain model. Thus, media objects act as information elements. To some extent, all information elements are also media objects, as they provide their information of course using any media type (e.g. text). However, for the same reasons as for interaction objects (see 3.2), the model should also provide abstract information elements. Likewise, the model must integrate media objects and abstract information objects in a consistent way. In section 5 we propose a possible solution for those requirements.

Presentation Units. A multimedia application will usually show different presentation units, similar to conventional applications. An extension arises from the dynamic nature of time-dependent media types like animation or video: they add an internal state to the presentation unit. This takes effect, if for example the presentation of the presentation unit is interrupted, e.g. to show a help window, and should be continued afterwards. A video or an animation should then potentially resume the state which it had before the interruption.

A solution for this problem is already proposed in [16]: an extended presentation unit is called *scene*. A scene can have attributes and methods like usual classes in UML diagrams. Attributes are used to realize the internal state of a scene. The methods include special entry-methods, which are invoked when the scene is entered. The initialization of a scene depends on the invoked entry-method and the method's parameters, which allow resuming the internal state.

4 Modeling the Media Objects

In section 3 we discussed the required aspects for the modeling approach. This section discusses the different model elements (denoted in italics) to model these aspects for the media-specific part.

4.1 Media Objects

As introduced in section 1 media types are classified into *temporal media* and *discrete media*. Some properties, like synchronization, occur only for temporal media objects. Discrete media types are *images*, *graphics* and *text*. The basic temporal media types are *audio*, *video* and *animation*. Further we distinguish *2D animation* and *3D animation*, because of their different structure. The production of 3D graphics and animation usually requires specialized tools and experts while 2D graphic creation is much simpler and even part of several multimedia authoring tools (e.g. Macromedia Flash).

4.2 Inner Structure

According to section 3 it is necessary to define the inner structure of media objects, to enable their manipulation through program code. By definition, only temporal media types can have a dynamic, code controlled inner structure. In particular (interactive) animations are usually closely linked to program code. Due to the complexity of 3D animation, several approaches to describe its structure already exist. We take them as base to derive the general concepts for our purpose. Afterwards we briefly sketch the structure of 2D animation, audio, and video.

Inner Structure of 3D Animation. A common concept in the 3D community for the description of 3D animation is the so-called *scene graph* (it is important to note that there is no direct relation to the scene concept described in 3.1). In the following we base on our work in [22] where we describe a platform-independent scene graph approach.

The nodes of a scene graph represent the visible, material *3D objects* themselves as well as components affecting their appearance. The latter ones are *light*, the current position of the viewer (referred to as *camera*), and predefined further possible viewer positions (*viewpoints*). The spatial information itself is also represented by a node: a *transformation* node performs one or more transforming operations – i.e. translation, scale, or rotation – to all its assigned sub-nodes. The nodes are connected by directed relationships which define the object hierarchy (usually as a tree). A transformation is always relative to its parent node, i.e. if the parent node is moved, its inner transformations are still valid.

It is important for our purpose that only nodes, which have to be accessed by application logic, are (explicitly) specified in our models. All other parts of the inner structure are omitted as implicit parts of the nodes.

If a node has multiple identical children (i.e. a car owns several wheels), it can be denoted in a compact way by just specifying one of the child nodes together with the actual number of children. Moreover, a keyword assigned to the relationship denotes whether the multiple children nodes are separate copies (keyword `copy`), i.e. can be modified separately, or whether they reference only the same single object (keyword `ref`), i.e. they are always exactly identical.

Inner Structure of 2D Animation. The structure of 2D animations can be derived from the 3D animations. Light, camera and viewpoints are not relevant for 2D level. The remaining elements are 2D objects and transformations. The shapes contained in an animation can be animations themselves or static graphics. The latter ones are not relevant for our purpose, as we here specify only nodes which are manipulated through code.

Figure 2 shows the main model elements for the inner structure of 2D and 3D animations.

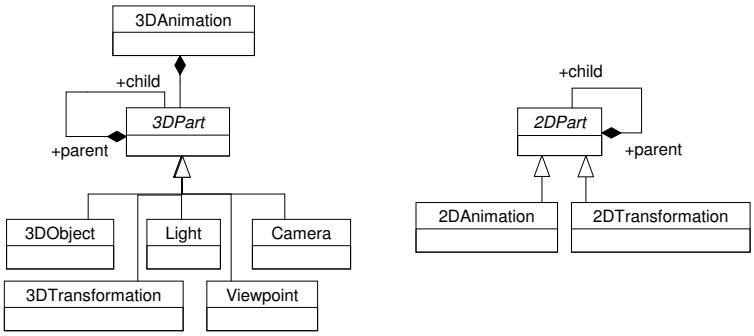

Fig. 2. Simplified metamodel for inner structure of 2D and 3D animation

Inner Structure of Audio and Video. The content of audio and video is rarely directly manipulated by program code. The general concepts can be summarized as follows: Audio can be composed of several tracks, e.g. for a left and a right speaker. Typical manipulations are the application of filters or the change of volume for one or more tracks. Moreover, an audio object can be composed of several samples, i.e. parts within its local timeline. Most actions on audio are time-related, e.g. jumping to a specific point on the timeline (*cue point*). To be independent from concrete audio objects we specify cue points by the semantics of their name (instead of defining concrete time values). The mapping from a cue point to a concrete time value can then be done during the deployment of the audio object, e.g. by the audio designer. Video can be handled in analogous way.

4.3 Example.

Figure 3 shows the media-specific part of the racing game example from section 2. The track is represented by an animation. It contains additional animations for obstacles and for the car. We clearly indicate the inner sub-objects here in the diagram by placing them within their topmost parent object. The car animation contains two front wheels. The inner structure is only specified insofar as required for the application code. For example the front wheels should be moved whenever the car drives through a corner. The annotations at the relationship between `Car` and `FrontWheels` specify that there are two front wheels which behave identically (and have thus not to be implemented as two independent objects).

Additionally the application should provide a cockpit view for the user displaying the current status of the car. This is realized by animations for the speedometer and the damage control. Moreover, the car is represented by sound. The other user interface objects of this screen are not contained in the diagram, as they require no specific media type.

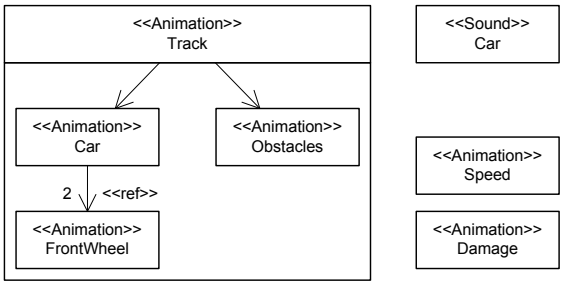

Fig. 3. Media-specific user interface elements of a racing game application

5 Integration of Media-Specific Aspect and Task-Based Aspects

In this section we integrate the media-specific part of the model from section 4 with the conventional task-based elements from section 2. In particular, we discuss how to fulfill the requirements of section 3.

5.1 Media-Objects as Interaction Objects and Information Elements

According to section 3 media objects are related to the conventional task-based user interface elements. All of them act as information elements.

Whether and how a media object can act as interaction element depends on the media type. Audio can usually not act as an interaction object, as it can

not be manipulated. Of course it is possible to record and parse audio using a microphone as accomplished at speech recognition. However, this does not relate to playing an auditory media object, and is therefore not discussed in this paper. The same holds for video, where a camera and gesture recognition are necessary for user inputs.

However, all visual elements, including video, appear on the screen and can therefore receive user events, e.g. when selected by a pointing device. Thus, all visual objects can act as action invokers. As animations can dynamically change their content dependent on the application logic, they can additionally act as editors. An example is the car animation which represents e.g. the current rotation of the car. The user manipulates the animation to edit the rotation value.

5.2 Media-Objects as Trigger

As mentioned in section 3, temporal media objects can also invoke actions without direct intervention from the user. It depends on the media type which types of triggers are possible. The triggers can be derived from the 3D domain, where they are represented by *sensors*. According to [22] common sensor types are *touch*, *proximity*, *visibility*, *collision*, and *time*. Touch and proximity sensors are not relevant here, because they describe events related to interaction with the user.

Visibility sensors trigger an event when objects became visible for the user. Collision sensors react, if two objects collide with each others. Both can occur for moving objects, i.e. (2D and 3D) animations.

Time events can occur for every temporal media type, namely when it reaches a specific point on its local timeline. This can be the end of the timeline or a specified cue point. All sensors can be assigned to a whole media object as well as to sub-objects from its inner structure.

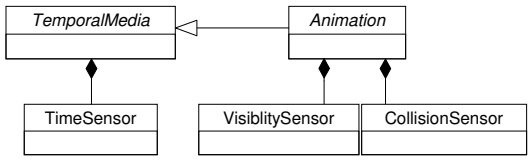

Fig. 4. Simplified metamodel for sensors

5.3 Modeling Example

The example shows the integration of the task-based user interface elements from figure 1 and the media objects specified in figure 3. The dashed arrows denote that a media object realizes an abstract user interface element. An abstract

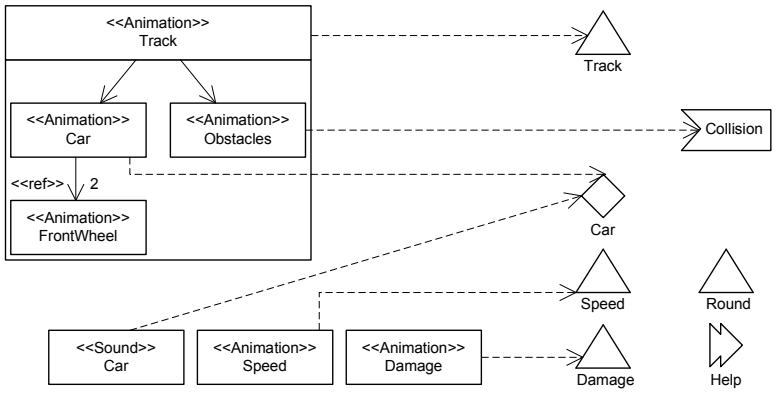

Fig. 5. Integration of task-based and media-specific user interface elements for the racing game example

user interface element can be realized by multiple media objects. If required for clarity, abstract editor elements can be decomposed in inputter and displayer. In the diagram, this would be possible for the car editor (but not shown here in the diagram). Abstract user interface elements, which are not realized by media objects, have to be realized during the implementation phase by appropriate platform specific solutions (e.g. widgets).

To describe the behavior of the user interface elements, the UMLi activity diagrams (see section 2) can used in the same way as before. The only difference are the sensors from media objects. They can be represented by UML *AcceptEventActions* to be used in the activity diagram. For example in figure 5 the `Obstacles` animation provides a collision sensor, which waits for the occurrence of collision events on the obstacles.

6 Model-Driven Development

In [16] we proposed the overall framework for a model-driven approach for multimedia applications. The current paper extends it by a platform-independent user interface model. It can be transformed into platform-specific models, using the concepts of *model driven development* (e.g. [23]).

Typically, multimedia applications are implemented using authoring tools like Macromedia Flash, which emphasize powerful support for the creation, integration, and deployment of media objects. However, they poorly support concepts for structuring the application logic and control. For example, interactive user interface elements often require the assignment of a script snippet to the respective element. As a result, script snippets are scattered all over the application. The consistent application of established software engineering concepts, like e.g. the Model-View-Controller paradigm [24], is often only possible with deep experience and under consideration of implementation "tricks".

Considering the mentioned strength and weaknesses of multimedia authoring tools, they seem to be dedicated to a model-driven approach. The model is much better suited to design the overall application structure and behavior. On the other hand, the authoring tools are suited best for realizing the media objects and the concrete user interface implementation. As a consequence, we transform the platform-independent models directly into code skeletons for the authoring tools and omit platform-specific models. The code skeletons contain placeholders (gaps or default objects) for those parts of the application, which are not specified in the platform-independent model. The placeholders have then to be filled out or replaced within the authoring tool.

The models proposed in [16] allow generating code for the complete overall structure of the application. The structural model specifies classes, attributes, and method signatures. The abstract user interface model allows the definition of the relationships between user interface elements and the structural model. The interaction model (corresponding to activity diagrams in UMLi) allows the generation of event handling code for the user interface elements.

The implementation of the methods from the structural model (i.e. the method bodies) is not part of the model. In multimedia applications methods often affect the user interface. Thus, the purposes of those methods are not only "hard" goals, like the correct computation of a value, but also "soft" goals, like esthetics. For example in a racing game a class `Car` provides methods which are responsible for the user's driving experience. Values and parameters often have to be found out by "trial and error" and should be optimized for the target platform. Thus, the methods are implemented directly within the authoring tool.

The media objects in the model are transformed into placeholders (e.g. bounding boxes), which have then to be replaced in the authoring tool. The abstract user interface objects can be transformed into widgets for the respective target platform. A rule based transformation, like in [21], is well supported by the MDA concepts for transformations, like parameters and constraints, as explained e.g. in [25].

7 Conclusion and Outlook

The approach described in this paper proposes a contribution for the model-driven development of multimedia applications. As the user interface is usually the core feature of this type of application, the concepts described in the paper can constitute the basement of multimedia modeling.

Beside the contributions to our modeling approach, the main contributions presented here lie in the general results for multimedia user interfaces. It is not contentious that multimedia applications require specific solutions addressing the heavy usage of media objects, as described by the existing research work. But in addition, the user also has to perform conventional tasks for controlling the application and its content. As a consequence we take here into account the results from conventional user interface modeling and integrate them with the media-specific aspects.

As a second general contribution we provide a fundamental discussion about the involved requirements for modeling user interfaces containing media objects. On that base we propose an abstract and platform-independent modeling approach for media objects and their inner structure.

As a consequence, the whole multimedia application models are platform-independent. We propose a model-driven approach generating directly code skeletons from the platform-independent models. The code skeletons contain gaps which are completed in the authoring tool. It is fundamental that the completion requires only tool abilities which the authoring tools are best in: the creation and deployment of media objects, the user interface layout and the platform specific definition of code on well-defined places predefined by the platform-independent model.

We have specified a MOF-compliant metamodel for our approach. On that base we have built a modeling tool for our models implemented on *Eclipse* [26] and related technologies like the *Eclipse Modeling Framework*. The tool provides simple tree-editors to create and edit models according to the metamodel. More sophisticated graphical diagram editors are currently under development. Moreover, we have a code generator producing *SVG/JavaScript* code skeletons from our models. At the moment we develop further generators, especially for Flash. For the Flash authoring tool we currently develop a plug-in to provide additional support for processing the generated code skeletons, e.g. navigation between the gaps in the generated skeletons and support for a round-trip engineering.

We are preparing the evaluation of our approach in student projects. In particular we provide an annual teaching course "Multimedia-Programmierung" (multimedia programming) where students have to develop in teamwork multimedia applications of middle size, e.g. in the last year a multiplayer racing game application implemented with Flash.

References

1. Tannenbaum, R.S.: Theoretical Foundations of Multimedia. Freeman, New York (1998)
2. Hannington, A., Karl, R.: Towards a Taxonomy for Guiding Multimedia Application Development. In: 9th Asia-Pacific Software Engineering Conference (APSEC 2002), 4-6 December 2002, Gold Coast, Queensland, Australia. IEEE Computer Society (2002)
3. Macromedia: Macromedia, http://macromedia.com/ (2004)
4. Mallon, A.: The Multimedia Development Process, http://ourworld.compuserve.com/homepages/adrian_mallon_multimedia/devmtpro.htm (1995)
5. Hirakawa, M.: Do Software Engineers Like Multimedia? In: IEEE International Conference on Multimedia Computing and Systems (ICMCS) 1999 Proceedings. Volume 1. IEEE Computer Society (1999) 85–90
6. Arndt, T.: The Evolving Role of Software Engineering in the Production of Multimedia Applications . In: IEEE International Conference on Multimedia Computing and Systems (ICMCS) 1999 Proceedings. 1 edn. IEEE Computer Society (1999)
7. Rahardja, A.: Multimedia Systems Design: A Software Engineering Perspective. In: International Conference on Computers and Education (ICCE) 95 Proceedings. IEEE Computer Society (1995)

8. Bianchi, A., Bottoni, P., Mussio, P.: Issues in Design and Implementation of Multimedia Software Systems. In: Proceedings of IEEE International Conference on Multimedia Computing and Systems (ICMCS '99), Florence, Italy, Volume I. IEEE Computer Society (1999) 91–96
9. Osswald, K.: Konzeptmanagement - Interaktive Medien - Interdisziplinäre Projekte. Springer, Berlin (2002)
10. Engels, G., Sauer, S.: Object-oriented Modeling of Multimedia Applications. In Chang, S.K., ed.: Handbook of Software Engineering and Knowledge Engineering. Volume 2. World Scientific, Singapore (2002) 21–53
11. Trætteberg, H.: Model-based User Interface Design. PhD thesis, Norwegian University of Science and Technology, Oslo (2002)
12. Hirzalla, N., Falchuk, B., Karmouch, A.a.: A Temporal Model for Interactive Multimedia Scenarios. IEEE MultiMedia **2** (1995) 24–31
13. Bertino, E., Ferrari, E.: Temporal Synchronization Models for Multimedia Data. IEEE Transactions on Knowledge and Data Engineering **10** (1998) 612–631
14. Arya, A., Hamidzadeh, B.: Face Animation: A Case Study for Multimedia Modeling and Specification Languages . In Deb, S., ed.: Multimedia Systems and Content-Based Image Retrieval. Information Science Publishing (2003)
15. Sauer, S., Engels, G.: Extending UML for Modeling of Multimedia Applications. In Hirakawa, M., Mussio, P., eds.: IEEE Symposium on Visual Languages 1999 Proceedings. IEEE Computer Society (1999)
16. Hußmann, H., Pleuß, A.: Model-Driven Development of Multimedia Applications. In: Talk at 'The Monterey Workshop 2004 - Workshop on Software Engineering Tools: Compatibility and Integration', Submitted for Proceedings. (2004)
17. Szekely, P.: Retrospective and Challenges for Model-Based Interface Development. In Vanderdonckt, J., ed.: Computer-Aided Design of User Interfaces. Presses Universitaires de Namur, Namur, Belgium (1996)
18. da Silva, P.P., Paton, N.W.: UMLi: The Unified Modeling Language for Interactive Applications. In Evans, A., Kent, S., Selic, B., eds.: UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings. Volume 1939. Springer (2000) 117–132
19. Wiecha, C., Bennett, W., Boies, S.J., Gould, J.D.: Generating Highly Interactive User Interfaces . In Bice, K., Lewis, C.H.a., eds.: Proceedings of the ACM CHI 89 Human Factors in Computing Systems Conference. April 30 - June 4, 1989, Austin, Texas, New York (1989)
20. Paternó, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In Howard, S., Hammond, J., Lindgaard, G., eds.: Proceedings Interact'97. Chapman & Hall (1997)
21. Vanderdonckt, J.: Automatic generation of a user interface for highly interactive business-oriented applications. In Plaisant, C., ed.: Companion Proceedings of CHI'94. ACM Press, New York (1994)
22. Vitzthum, A., Pleuß, A.: SSIML: Designing Structure and Application Integration of 3D Scenes. In: Proceedings of the tenth international conference on 3D Web technology. ACM Press, New York (2005)
23. Frankel, D.S.: Model Driven Architecture. John Wiley (2003)
24. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System Of Patterns. Volume 1. John Wiley, West Sussex, England (1996)
25. Kleppe, A., Warmer, J., and, B.W.: MDA Explained. Addison-Wesley (2003)
26. Eclipse: The Eclipse Project, http://www.eclipse.org/ (2004)