Model Driven Development of Multimedia Applications

Heinrich Hußmann and Andreas Pleuß

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

Abstract. The development of multimedia applications is a branch of software development of increasing importance. Many advanced user interfaces integrate multimedia elements, and critical multimedia systems are emerging, e.g. in training and simulation areas. Multimedia user interfaces pose a number of new challenges to the software development process and technologies. Unfortunately, software engineering principles are mostly ignored in current practice of multimedia application development. This paper suggests a way of how the gap between multimedia applications and software engineering can be made smaller. A modeling language is presented which is based on UML 2.0, but is tailored for effective, platform-independent abstractions from concrete multimedia technologies. Using such a modeling language, we show it is possible to apply the paradigm of *Model-Driven Development* to multimedia applications, leading to better structured and more maintainable applications, and a much higher degree of independence from multimedia platform technologies. The language is described using a medium-sized case study, and concepts for prototypical tools supporting model-driven development of multimedia applications are outlined.

1 Introduction

Multimedia application development is a branch of software development which is growing rapidly but often not recognized as a subject of software engineering. The term *multimedia application* refers to program systems which are heavily based on a user interface using powerful graphical effects, which integrate sound and other time-dependent media in the user experience and which usually include a high degree of interactivity. Examples are games, teaching and training applications, and various simulation tools. Multimedia applications are slowly entering the critical-systems area, for instance in simulation and training applications (e.g. for operating or maintaining complex systems in aircraft).

Typical platforms for multimedia application development are development environments which integrate graphical design and programming. Rather popular development environments of this type are Macromedia Director and Macromedia Flash [1], but also emerging standards for the Web like the SVG (Scalable Vector Graphics) [2] format are moving into a similar direction by enabling

arbitrary JavaScript programming. "Programs" appear in such a development process only as small snippets in a scripting language which are attached to individual parts of the graphical design. Such a script may, for instance, control the movement of a graphical element on the drawing surface, or may handle user input. There is a clear trend towards development environments for multimedia applications which comprise standard software development technologies like object-oriented programming (e.g. the powerful ActionScript language included in the most recent versions of Macromedia Flash). So it is obvious that very complex systems are realizable on such platforms.

A serious engineering problem appears when highly complex multimedia applications including non-trivial application logic are designed in a naive way using the existing development tools. In fact, software *design* does not take place at all in such a scenario. Moreover, it is almost impossible for the developer to get a serious overview of the whole code contained in an application and the way how it interacts, since the code is scattered all over a working document the structure of which is dominated by the graphical design. (Tools like application browsers or separately stored code do not improve the situation essentially.) In principle, it is possible to use classical software design techniques to make e.g. a Flash/ActionScript application better structured and more maintenance-friendly. For instance, it is possible to apply the Model-View-Controller (MVC) paradigm [3] also to Flash/ActionScript applications. Interestingly, the literature (and even the online discussion forums) on Flash/ActionScript generally talk little about such design principles, and the practical realization makes it necessary to use a few design "tricks" which need to be applied consistently at many places of the code.

At this point it becomes obvious that the design of multimedia applications is an excellent candidate for the application of the principles of *Model Driven Development* (*MDD*, [4], [5], [6]). Model Driven Development relies on a process using high-level abstract representations of a system design and stepwise refinement into specific executable code. In particular, it distinguishes between a platform-independent level of design and the refinement onto platform-specific designs. Here are a number of benefits of a Model-Driven development style for the development of multimedia applications:

Platform independence: Often a basic design idea for a multimedia application can be described in an abstract way which is not yet specific about the platform (development environment and runtime system). Examples of platforms are here Macromedia Director, Macromedia Flash, SVG/JavaScript, native Java applications and many more. Using MDD, valuable design work can be re-used for different platforms, and the decision for a specific platform can be delayed.

Integration of graphical design and software design: As mentioned, multimedia applications often have only a graphical design but no software design. Using MDD, a software design can be described on a high level of abstraction, and – using appropriate abstractions from the graphical design – the combination of graphical and software design can be specified in a clear way. Ideally, the high-level models can serve as a means for better communication between graphical

designer and software designer, thus relieving a serious communication problem for multimedia projects.

Better structured applications: The resulting applications from a Model-Driven development benefit from the better thought-out design anyway. But moreover, and more specifically, an *automatic generation* of (parts of) the application code from abstract models enables a possibility which is otherwise not achievable: The generated code can be defined in such a way that it realizes structural patterns (like the above-mentioned MVC pattern) that make it much easier to extend or modify the code afterwards or even at runtime. The generation enforces the application of the design patterns at all relevant places in the generated code, a property which is very difficult to achieve with hand-crafted code. So, the effect of MDD is similar to aspect-orientation: Platform-specific "tricks" for better-structured code are "woven" into the application code automatically at the appropriate places.

In order to realize the vision of MDD for multimedia applications, many open questions have to be answered e.g. about the modeling language, the refinement levels, platform-specific models, and tools. In this paper, we will concentrate on an appropriate language for abstract models of multimedia applications, partially based on previous work by Stefan Sauer [7]. The modeling language we will present in the following is based largely on UML 2.0 [8] but also deviates from the standard slightly where adequate for the application domain. The paper is structured as follows: Section 2 presents an example application. In sections 3, 4, 5 and 6, four different modeling views of a multimedia application are considered: The static structure of the application including media assets, a simple abstraction of the graphical design usable for the software designer, the basic structure of the user interaction, and the internal temporal structure of the application. Section 7 provides an overview on the integration into MDD and code generation. Section 8 discusses related work and explains the innovative aspects of this work. Section 9 finally gives an outlook to the many further steps required, in particular the ongoing work on tool development.

2 Example Application

This section presents a multimedia application used as example throughout the whole paper. In contrast to most other application areas, games like e.g. a racing game traditionally make intensive use of multimedia effects to provide the user an impressive atmosphere. Therefore even a small sample extract of the application seems sufficient to point out the challenges of multimedia applications. For this reason, we choose a racing game as running example in this paper.

For the purposes of this paper we restrict the example to the following features: The application starts with a menu where the user may set various options, like the car and the track for his next race. After the user has started the race he has to steer his car over the track to absolve a number of laps. During the race the display shows beside the track some information on the player's state, like lap number, current speed, and current fuel status of his car (figure 1). When

the race is finished the player's score is displayed. Afterwards the application returns to the menu screen. The user can always refer to a help text, where the initially displayed content depends on whether it is called from the menu screen or from the race. For the example in this paper we omit further functionality. Of course, current professional racing game products provide much more functionality and options for the user as well as sophisticated application logic and complex multimedia effects for a realistic driving experience.

Sample implementations of racing games, using Flash/ActionScript technology, have been developed at University of Munich under the supervision of the authors. Seven teams with six or seven students each worked for 3 months to develop different racing games with distributed multi-user support and intensive use of sound and animations. The project was part of the course "Multimedia-Programmierung" in the summer term 2004. This paper is partially based on experiences from these projects.

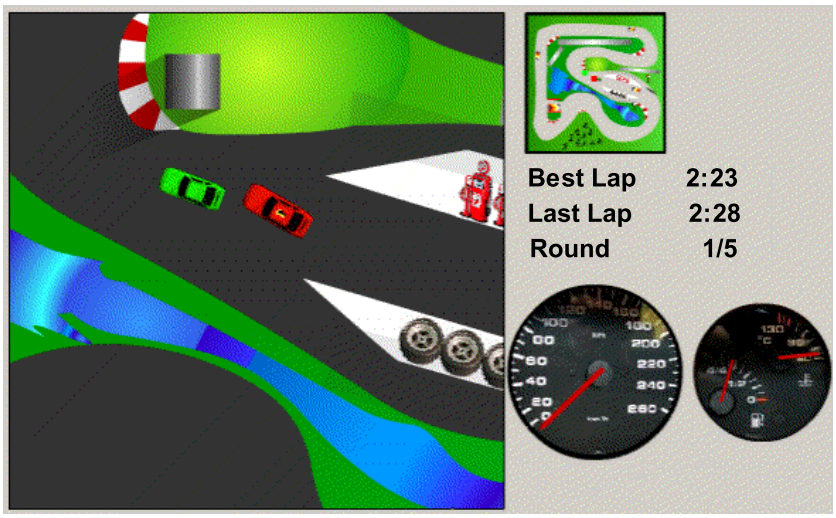

Fig. 1. A screenshot of the racing game example

3 Application Structure and Media Integration

The following section presents our modeling language for multimedia applications. The base part of the model is a model of the static application structure explained in this section. It contains the application's domain specific classes (application entities). As our approach uses the standard UML 2.0 diagrams, we base on UML class diagrams to model the application structure. We support all usual UML 2.0 class diagram elements, i.e. class properties like attributes, associations and operations and class relationships like generalizations. The classes

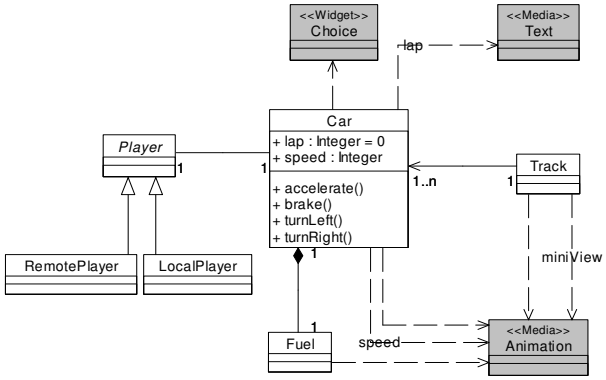

Fig. 2. Example classes and media integration

can be structured into packages and enriched e.g. with comments. Figure 2 shows an exemplary class diagram for the example application, including the classes `Car` (containing some exemplary properties), `Fuel`, `Track`, `Player`, `LocalPlayer` and `RemotePlayer` as well as relations between them. The grey colored elements and their relationships are explained below.

As the user interface presentation is the main characteristic of multimedia applications, it is an important design decision, how to represented the application entities on the user interface. In multimedia applications the user interface elements are mainly media objects, i.e. objects of a discrete or continuous media type as in multimedia standards like [9]. As an example, in a racing game a car may be represented by an animation and sound.

In many cases individual media objects are required for the application. Then they are designed with a multimedia authoring tool (e.g. an animation) or produced in a separate process (e.g. a video). These time consuming tasks have to be considered as early as possible in the development process. Especially for interactive applications the relationship between application logic and media object is essential. In the example above, the car animation moves according to the behavior defined in the car class and the car class instance is influenced by events occurring for the car animation. Besides media types, other customary UI elements like buttons, check boxes, lists etc. (so-called *widgets*) can be required analogous to media objects. For example in menu of a racing game a *choice* widget presents the user all available car objects for choice. As the user interface objects (media and widgets) are important assets of the application with direct relations to application entities, we include them and their relations to the application entities in the structural model.

In our approach the relationship between UI elements and application entities is a special kind of dependency, called *UI Representation*, which specifies that the UI element represents the application entity. It is defined as a relationship (instead of e.g. specifying the application entities as a type of the UI element itself), because often an application entity has multiple representations. The UI

Representation defines a new subtype of the specified UI element specialized for the presentation of the entity. This is necessary to enable multiple instances on the user interface, e.g. several car animations on the screen. These instances are specified in the presentation model in section 4.

The example in figure 2 shows the UI elements `Animation`, `Text`, and `Choice`. For distinction with application entities they are branded with the keyword `Widget` respectively `Media`. Additionally they are grey colored. The notation of UI Representations corresponds to dependencies in UML. In the example, the entity `Car` is represented by animations for position and speed, a choice, and text. The entities `Fuel` and `Track` are represented by animations. UI representations can have an assigned name, which is optional for the first, and mandatory for further UI Representations between the same two elements (like for `Track` in figure 2). If no name is specified, by convention the name is composed of the name of the entity and the UI element, e.g. `Car_Animation`. Another convention allows the representation of a single property of an entity: in this case the name of the UI representation corresponds to the property's name. In the example the UI representation speed is used to introduce a speedometer, which represents only the car entity's `speed` attribute.

For the purpose of easy to handle models and an automatic generation of useful code with minimized effort, we define the UI elements as kinds of reusable components (e.g. classes) with internal implementation of common required functionality. This should be obvious for widgets, as they are usually provided by object oriented programming languages in class libraries. Media elements, at a first look, represent only a media type, i.e. there is no implementation. However, for the purpose of application development obviously the developer does not only refer to a media document itself, but he additionally wants it to be accessed by the user. E.g. a video should not only exist, it should also be possible to play it. Following the current state of the art, it should also be possible to pause, resume, stop, etc. the video. Furthermore events are required, e.g. to notify other application parts when the video is finished. For such sophisticated functionality usually a reusable player component is used, which can be customized by parameters. Likewise, this functionality should not always be modeled explicitly. Therefore in our approach a media element acts as a component, providing the required display functionality for a specified media type. As a consequence, it is possible to define attributes and operations as well as provided events for media and widget components. The modeler can reference these properties and events to define the behavior of the application (see section 5).

As an additional mechanism for structuring the application statically, we introduce *scenes*, which are a common metaphor in multimedia authoring tools. In our approach a scene is a state of the user interface, which displays a given screen layout over a period of time. For each scene, exactly one screen layout is defined (see section 4). As always exactly one scene is active, they can be referred to as the top-level states of the application (see section 6). The transition to another (possibly the same) scene corresponds to an initialization of the user

interface. Examples for scenes in the racing game are `Menu`, where the menu screen is displayed or `Race`, where the race is played.

As scenes structure the application, we define them as part of the structural model. A scene may have attributes and operations. Therefore we consider them as a special case of a class.

A scene's operations are restricted to so-called *entry-* and *exit-operations*. Entry-operations are called by each transition into the respective scene. They are responsible for the initialization of the user interface. As in interactive applications the scenes are not completely predefined, the entry-operations may have parameters for the scene. For example the race scene needs has to be aware about the decisions made by the user in the race menu (e.g. the selected car). Additionally a scene may have multiple exit operations (e.g. `startRace()` and `continueRace()`). Thus, the scenes are reusable in a flexible way.

Exit-operations perform a transition to another scene. For each possible transition an exit-operation is defined. In addition to the transition, exit-operations may perform some "clearing work", e.g. release resources used by the scene. Usually it is not mandatory for the modeler to specify exit-operations explicitly in the structural model, because they correspond to the possible transitions. Therefore it is possible to derive them automatically from the transitions specified in the temporal model as explained in section 6.

A scene may have attributes (usually private) to store parameters values of their entry-operations and to enable an internal state. Additionally a scene contains references to objects created within the entry-operation.

Finally, a scene may have attached a UI Representation. In some cases a UI element doesn't represent an application entity, but a scene itself. Examples are the help text, which represents the help scene itself or e.g. an intro video representing the intro scene (not part of our example application). Help text and intro video don't have any direct relationship to an application entity. Therefore we allow UI representations also between scenes and UI elements.

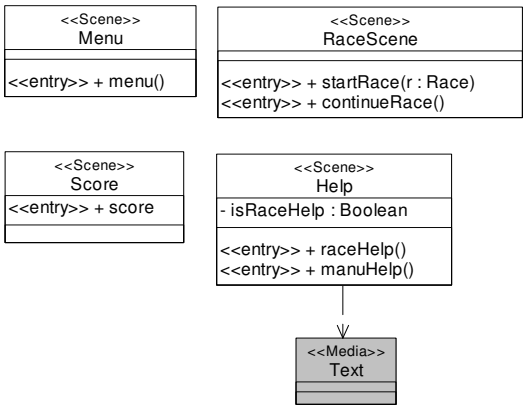

Fig. 3. Example scenes and media integration

Figure 3 shows an extract of the structural model containing the scenes `Menu`, `RaceScene`, `Score`, and `Help`. For distinction to application entities, scenes are branded with the keyword `Scene`. They contain entry-operations, which are branded with the keyword `entry`. Additionally the scene `Help` contains a private attribute `isHelpScene` which e.g. holds the information, which of the entry operations has been called. The `Help` scene is represented by a text object to display the help text.

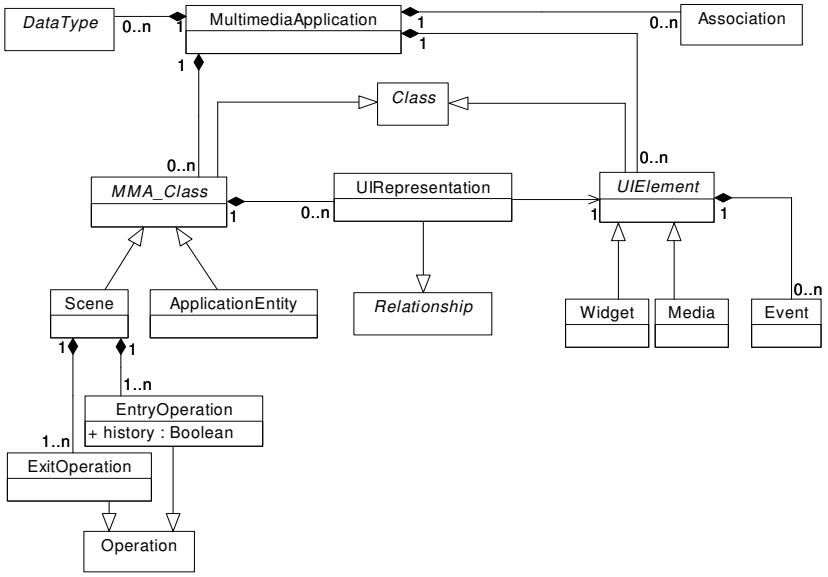

Fig. 4. Metamodel diagram for application structure and media integration

As a reference for the full abstract syntax of the static structure diagrams, we show the metamodel in figure 4. Its base is an excerpt of the UML 2.0 metamodel (not shown in this paper), which defines basic model constructs and all model elements necessary for UML 2.0 class diagrams. Metamodel elements with omitted attribute compartment are already defined in other parts of the metamodel – i.e. here in this figure the UML 2.0 excerpt. General information about our metamodel is also provided in section 7.

Top-level element of the metamodel is `MultimediaApplication`, which builds the container for the model and contains all other model elements. `Application-Entity`, `Scene` and `UIElement` are subtypes of `Class`, i.e. they may have all properties and relationships like classes. Additionally, `ApplicationEntity` and `Scene` may have assigned a `UIRepresentation` which references a `UIElement`. The operations of `Scene` are restricted to `EntryOperation` and `ExitOperation` (the restriction is not visible in the metamodel, but defined in additional well-formedness rules). The `history` attribute for `EntryOperation` is explained in

section 6. `UIElement` is specialized by `Media` and `Widget` and may additionally provide Events.

4 User Interface Presentation

A sophisticated user interface is one of the main goals of multimedia application development. Often the basic idea for an application is a vision of a user interface presentation. Also customers often describe a multimedia application by its desired screen layout. The application logic is often subordinate to these ideas and results from the decision, which elements to present to the user.

Therefore we include the *user interface presentation* in our model, which describes the spatial structure of the visual user interface. It is modeled at a simple level of abstraction usable for software engineers as well as designers. Thus, the UI presentation model may additionally be referred as base for communication between different developer teams, like software engineer and media designer. Besides, the presentation model shows the concrete instances of the UI Representations from the structural model and their allocation to scenes.

In our approach the spatial structure of the screen layout is directly notated by arranging UI objects on in a diagram representing a screen. We represent the UI objects by simple bounding boxes, i.e. primitive shapes like rectangles or ellipses. The concrete look of the UI elements and all additional UI design are not part of our models. We supposed these tasks to be done in adequate authoring tools under consideration of the presentation model. Thus, the presentation models can be referred as templates with the bounding boxes as placeholders. Ideally, tool support allows loading the models directly into the authoring tool, where the media designer fills out the placeholders and adds adornments.

As mentioned in section 3, each scene has exactly one initial UI representation. As consequence, exactly one presentation diagram is assigned to each scene. However, the presentation diagrams can be subdivided in several subdiagrams for different output channels, e.g. visual and auditory channels. In this paper, we restrict to the description of the visual channel, i.e. the screen layout. Like in multimedia authoring tools, the screen layout may be structured in z-direction by layers. On the layers the UI objects are arranged. The concrete UI objects can be referred to as instances of the UI Representations from the structural model (see section 3).

Figure 5 shows a presentation diagram for the scene `RaceScene`. A UI object is identified by a name and the type of its UI Representation (separated by ':'). If only one instance exists for a given UI Representation, the UI object name may be omitted, as shown in the example. A possible implementation of the presentation diagram is shown in figure 1.

Figure 6 shows the metamodel for the UI presentation model. The metaclasses `Scene` and `UIRepresentation` form the connections to the structural model in section 3. Besides `Graphics`, further subclasses of `OutputChannel` (like `Audio`), are omitted in this paper.

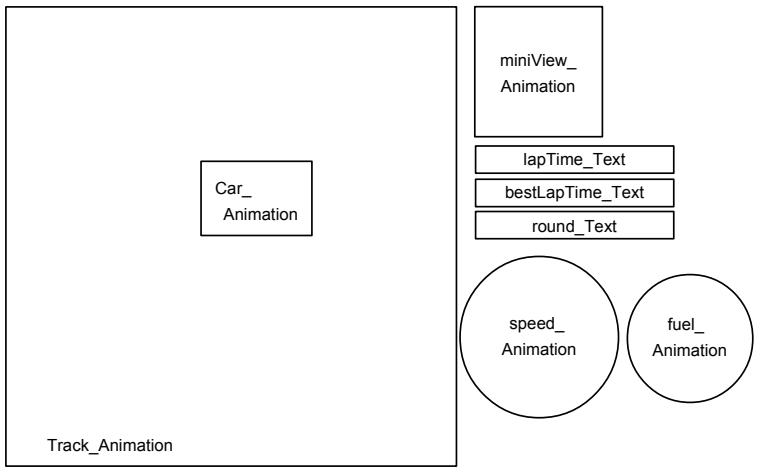

Fig. 5. Example presentation

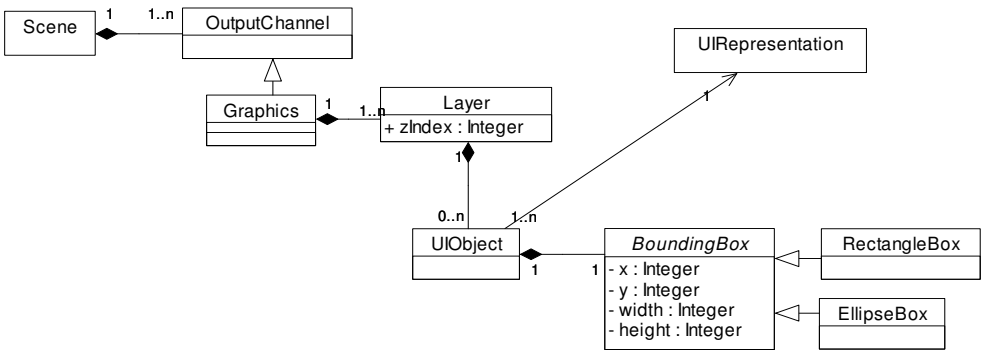

Fig. 6. Metamodel diagram for presentation

5 Interaction

A characteristic property of multimedia applications is the high degree of user interaction. It is highly related to user interface elements and controls the application logic. Therefore we introduce an interaction model.

The interaction model is based on events. They are provided by UI element classes from the structural model. The purpose of the interaction model is to provide an overview, which concrete UI objects (defined in the presentation model in section 4) of a scene listen to which events and which other application parts are influenced by the event handling. As the applications are expected to be highly interactive, in many cases UI objects listen to multiple events, e.g. whether the mouse pointer is over a specified area within the UI object, the mouse is pressed or the mouse is released. Additionally, the behavior connected to an event may heavily depend on conditions, e.g. in the racing game the car is controlled by key event whereby different keys trigger different behavior. To provide an easy to handle overview over the different cases, we use tables to denote the interaction. For each scene there is a table containing the interactive UI objects, the events they listen to, conditions over the events, and the triggered event handling.

The event handling consists of one or more actions to achieve the required behavior triggered by the event. The actions are usually method calls, as the event handlers do not contain any application logic themselves. Target objects of an action are the triggering UI object itself, other UI objects in the scene, the application entity represented by the UI object or the scene class (e.g. to call one of its exit-methods). While the operations of application entities often contain sophisticated application logic (like the movement of a car according to its parameters, including collision detection), the event handling usually consist of a few method calls. Thus, the complete event handling methods can easily be specified completely in the model. As a consequence, a code generator can automatically generate the complete interaction code. This can be very helpful, especially as the interaction code consists of small code snippets which must be put on various different places within the application code.

As event handling methods are intended to be defined completely in the model, in some cases additional actions besides method calls are required. Therefore we include the UML 2.0 actions in our model approach, which allow complete behavior specifications. Basically, the provided constructs would also be suitable to define the operations of the application entities. However, as we explain in section 6 they are preferably implemented directly in an adequate development environment with the help of platform specific features and the ability for an immediate testing of their effects on the user interface.

Table 1 shows an example for the interaction model denoted in a table. The UI object `Car␣Animation` listens to the `KeyPressed` event. The event handling is dependent of a constraint, e.g. if the pressed key is '8' the method `accelerate()` from the `Car` entity is called. The `MouseClicked` event at `Track␣Animation` triggers a call of the scene's exit-method `exitToHelp()`.

UIObject	Event	Constraint	Action
car_Animation	KeyPressed(c)	c = '8'	car.accelerate()
		c = '2'	car.brake()
		c = '4'	car.turnLeft()
		c = '6'	car.turnRight()
track_Animation	MouseClicked()		exitToHelp()

Table 1. Example interaction

Figure 7 shows the metamodel for the interaction model. An `EventOccurence` refers a `UIObject` (from the presentation model) and an `Event` (from the structural model). The corresponding `Interaction` consists of an optional `Constraint` and a set of `Actions`. The metaclass `Action` is specialized by various subclasses, not shown in this paper, analogous to the metaclass `Action` in UML 2.0.

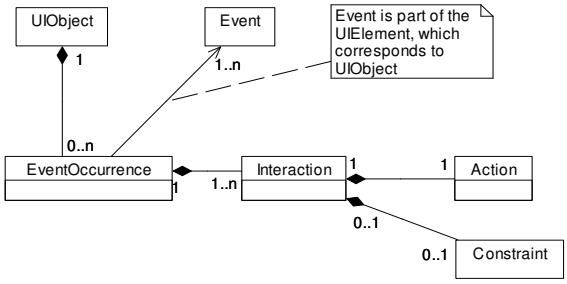

Fig. 7. Metamodel diagram for the interaction model

6 Temporal Structure

Another important aspect of multimedia applications, which is highly related to the user interface, is the temporal model. In conjunction with the presentation model, it covers the complete spatio-temporal presentation of the application. As multimedia applications are supposed to be highly interactive and especially through the integration of various time-dependent media objects, the overall temporal behavior of the application is very complex and can only be modeled with a high amount of effort. However, the scenes introduced in section 3 – which were also used to structure the spatial UI presentation (see section 4) – are a well suited level of granularity. As a scene corresponds to a period of time, the order of the scenes defines the overall temporal presentation of the application. However, as the applications are supposed to be interactive, the order of the scenes is usually influenced by the user. Therefore our approach models the possible temporal relationships between the scenes, referred as *temporal structure*.

For the modeling we use an adapted version of UML state charts. We consider the scenes as states, as at each point of time during the application execution exactly one scene is active (scenes, entry- and exit-methods are defined in section 3). A transition between two scenes (source scene and target scene) indicates that after the source scene was active, the target scene may be the next active scene. The execution of a transition corresponds to the execution of an exit-method in the source scene, followed by the execution of an entry-method in the target scene. Each outgoing transition corresponds to exactly one exit-method in the source scene. Thus, a scene's exit-methods are usually derived from the temporal structure model (instead of modeling them explicitly in the structural model). To allow adaptable scenes, they may have multiple entry-methods which may provide parameters. Hence it is possible to specify for a transition, which entry-method of the target scene it addresses. Parameter values for entry-methods are not specified in the model, as they are usually computed in the code of the source scene.

Figure 8 shows an example for the temporal structure model. The scenes - `Menu`, `RaceScene`, `Score`, and `Help` - correspond to the structural model. The arcs between them represent the transitions. Each transition has attached the name of the addressed entry-method; e.g. `startRace()` of `RaceScene` is called, if the user leaves from the `Menu` to the `RaceScene`.

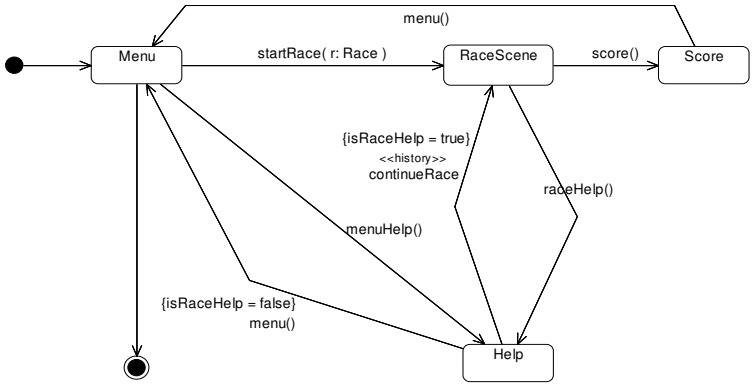

Fig. 8. Example for the temporal structure

Additionally it is possible to attach a constraint to a transition. Transition are may only be performed, if associated constraints are evaluated to `true`. Constraints refer to properties of the respective source scene. In the example (figure 8), it depends on the property `isRaceHelp` of the `Help` scene, whether leaving the `Help` leads back to the `Menu` or to the `RaceScene`.

To model start and termination of the application, we use (analogous to UML state charts) a start and an end state. The transition from the start state defines implicitly a start method without parameters in the target scene.

Finally, two different kinds of semantics for a scene entry exist. In most cases, a scene starts with an initial state, which depends only on the parameters of the entry-method. However, it might be required to resume the scene's internal state, which it had when it was active before (similar to history states in UML state charts). For example, if the user calls the help function during a race, he wants to continue the race afterwards. To model this situation, entry-methods may be branded with the keyword `history`. As a consequence, whenever a scene is left via a transition which may later lead to a history entry, the scene has to keep is internal state. In figure 8, the entry-method `continueRace()` of `RaceScene` has assigned the keyword `history`, to specify that the previous race can be resumed.

Besides temporal structure on scene level, the detailed behavior within the operations of application entities and scenes has to be defined. Basically, the UML 2.0 behavior diagrams are sufficient for this task. However, as we also mentioned in section 5, the implementation of these operations is not part of our modeling approach. As they often contain non-trivial code, modeling the operations would require a high effort and result in very large models. Especially for the purpose of code generation, a complete specification of the operations would be necessary, because an operation code containing scattered semantic gaps will usually not help the user. Additionally, to achieve complex effects combined with optimal performance requires in many cases platform-specific code. Furthermore, the operation code often has to be executed and tested immediately to ensure the estimated effects on the user interface. Therefore, the operations of entities and scenes are preferably implemented directly within an appropriate development environment.

If the modeler desires a visual model of an operation, e.g. to increase the understanding of very complex operations, he may use the UML sequence diagram. In contrast to previous versions, in UML 2.0 the sequence diagram provides both a generic description of the operation's logic as well as the possibility to specify time related constraints, e.g. for synchronization tasks. Therefore they may be used for the (optional) modeling of the detailed behavior of the application, if required.

Figure 9 shows the metamodel for the temporal structure. The grey colored metaclasses show the basic principle derived from UML state charts. As a `State` may act a `Scene` as well as the pseudo states representing the start and the end of the application execution (`StartState` and `EndState`). A `Transition` corresponds to the execution of an `ExitMethod` (respective `ApplicationStart` for the `StartState`) and the execution of an `EntryMethod` (respective `ApplicationExit` for the `EndState`). Additionally a `Constraint` may be attached to the transition.

7 Overall Approach and Tool Support

In this section we integrate the presented views of the model and explain the integration into MDD, including tool support and code generation.

The four views presented in the preceding sections are highly interrelated by the model elements which occur in several diagrams. The structural model is

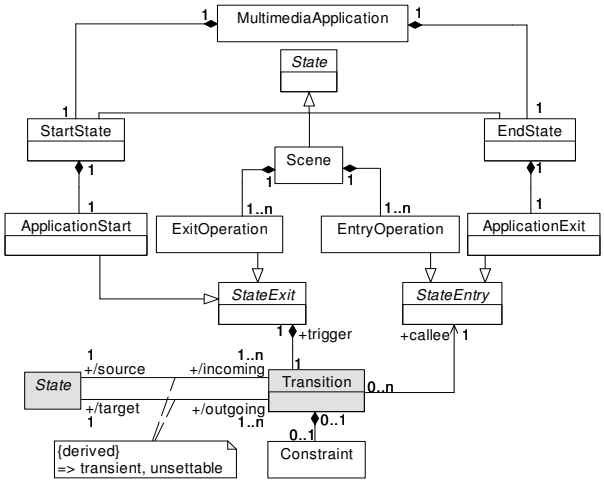

Fig. 9. Metamodel diagram for predefined behavior

the base part of the model, as all other parts are related to it. A presentation diagram is assigned to a scene. Within a presentation diagram, the UI elements reference the UI Representations from the structural model. The events in the interaction model connect UI elements of the presentation diagram to the structural model by calling operations. Additionally the events itself have to accord to their definitions in the UI element classes. The temporal structure model is more separated. On scene level, only scenes and their entry- and exit-methods from the structural model are required. Each sequence diagram for the description of detailed behavior is assigned to an operation.

For the definition of our modeling language, we use a metamodel, i.e. describing of the abstract syntax in terms of a model itself. The Object Management Group (OMG) is an industrial consortium defining various standards related to modeling (including the UML), especially in context of MDA. *MDA* (*Model Driven Architecture*, [4], [5], [6]) is the OMG-specific term for MDD. The OMG also provides a meta-metamodel, called *Meta Object Facility* (*MOF*), which serves as base for all MDA related technologies. Our metamodel is compliant to MOF and therefore compatible to other MDA related technologies and tools.

We provide tool support, based on the *Eclipse* [10] technology, an open source project providing various frameworks to build high quality editor-based applications. Existing Eclipse-based applications are e.g. development environments for Java like *IBM Websphere Studio* or UML modeling tools like the latest versions of *Rational Rose*. Our metamodel is also compliant to an Eclipse-specific implementation of MOF (called *Ecore*) resulting in a modeling tool compliant to other Eclipse technologies. Currently, our tool provides simple editors to create and edit models in the terms of our modeling approach. Graphical diagram editors (using the *Eclipse Graphical Framework*) are under development, whereby we

partially reuse existing Eclipse-based UML tools to support the UML-compliant parts of our modeling language.

From the models we can generate code easily. Possible target languages are in principle all programming languages which allow a multimedia user interface, like Java or C++. As there are several implementation technologies adapted for multimedia applications, usually one of them will be the first choice, like Flash/ActionScript or SVG (Standard Vector Graphics) in combination with JavaScript. Merely document-oriented formats with very restricted programming abilities, like SMIL or MPEG, are less suitable for our approach.

In the following we describe the general concepts for the code generation. The models specify all parts of the application, apart from the concrete design of the UI objects and the operations of scenes and application entities, as these parts are preferably implemented in a development environment (see sections 3 and 4). This means that all parts of the application, which have to be filled out manually, can easily be found at well-defined places. The sequence diagrams for optional documentation of operations are not taken into account for the code generation.

To structure of the application implementation as a whole, we use the MVC paradigm. The application entities form the *model* part (in the meaning of MVC). The classes for UI elements are usually already implemented in a class library delivered with the target language, otherwise they have to be included in the *model*. The *view* is directly derived from the presentation diagrams. The scene classes act as *controller*. A scene class owns the UI objects of the *view* and also the event handler methods according to the interaction model. Additionally the scene's entry- and exit-methods perform the control about the overall application.

We implement the code generators also using Eclipse and in particular the *Java Emitter Templates* (JET) technology, which allows a template based code generation. As consequence, the generated code is encapsulated in separate template files (in human-readable format) and can easily be adapted or refined. The generators are integrated in our modeling tool. As our tool supports import and export of models in *XMI* format (the OMG standard for model exchange between tools) the generators may also be used independently from our modeling tool. Currently we provide the generation of SVG/JavaScript. A generator for Flash/ActionScript is under development.

8 Related Work

Regarding the development of multimedia applications, the lack of adequate software engineering methods is already a well known problem, discussed e.g. in [11], [12], [13], or [14].

For model-driven approaches in general, the de-facto standard for visual modeling is the UML [8]. We use it as base for the modeling of multimedia applications and define our extensions as much as possible consistent to UML 2.0. As UML is a general object-oriented approach, for many specific application areas

extension are defined (often as OMG standards itself). As we show in this paper, important aspects of multimedia applications can not be modeled with UML in an adequate way. These are especially the integration of media types (see section 3), the spatio-temporal structure of the user interface (see section 4 and 6) as well as interaction (section 5). Several work exists addressing the modeling of single of these aspects.

The modeling of temporal behavior and synchronization is addressed e. g. in [15] or [16]. As the approaches focus on very detailed description of time relations, we did not include them in our approach (see section 6).

The UML extension UML for interactions (*UMLi*, [17]) models interactive user interfaces. The approach is explicitly restricted to form based applications. Based on the restricted set of UI elements and events, they model presentation structure, interactions, and resulting behavior in a very compact and straightforward way. An extension for support of media objects and more flexible interactions would result in highly overloaded models. The UML profile for interaction design [18] concentrates on architectural and structural aspects.

Work on the development process multimedia applications itself without the use of multimedia-specific modeling concepts is presented in [19] or [20]. A mainly document-oriented view on multimedia applications, i.e. without considering application logic, is e.g. provided in [21] and [22].

A large area of research is the model driven development of web applications. Several approaches allow to specify almost the complete application within the model (except the concrete UI layout) and provide code generation, e.g. *UWE* ([23], [24]), *OO-H* ([25]) or [26]. As a result, they are also highly optimized for web applications and base on the restricted set of user interface elements and interaction for web browsers. For this reasons, they are less suitable for multimedia applications, as complex UI elements are not supported, similar to UMLi.

The most related approach is *OMMMA* (*Object-oriented Modeling of Multimedia Applications*, [27], [7]). The approach provides an integrated modeling of multimedia applications based on UML. They provide four views of the model. Application structure and user interface structure are modeled similar to our approach in a class diagram respective presentation diagram. Interaction is modeled using UML state charts. Extended UML sequence diagrams are used to model temporal behavior. While on a first look our approach seems very similar to OMMMA, in detail we provide several changes for the purposes of easy to handle models and code generation.

OMMMA refers media objects as abstract media types without including their implementation related features. The modeler has to model required properties and events (like a method `play()` for a video – see section 3 for further explanation) explicitly by hand. The relationships between media types in the structural model and the UI objects in the presentation diagram are not directly visible in OMMMA models, but only with help of the sequence diagrams. Widgets are not part of OMMMA's structural model. The interaction in OMMMA is described by a UML state chart. As we argue in section 5, a compact notation

like tables is suited best for modeling the events. Additionally in a state chart, it is difficult to find out, which UI element triggers which events. Furthermore, the modeling of a whole application in a state chart, as supposed in OMMMA, may result in very complex models which are difficult to handle. For the modeling of the temporal behavior OMMMA uses sequence diagrams. They are mandatory, as they additionally contain the relationships between structural model and presentation model. As we discuss in section 6, the modeling of each operation in a sequence diagram is not adequate for our purposes. Additionally to the mentioned differences to OMMMA, we extend the concept of scenes (with regards to their implementation) by introducing entry- and exit-methods or the possibility to assign UI elements directly to a scene (see section 3).

To summarize, there is currently no modeling language comparable to our approach except OMMMA. As OMMMA provides important insights for the modeling of multimedia applications, we use it as a base for our approach. We add new ideas, refinements, and simplifications and develop tool support.

9 Conclusion and Outlook

The modeling language which was presented above is massively based on UML 2.0 and related technologies (like meta-modeling and MDA), but a model in our language is significantly different from a standard model of an arbitrary software application. It is our belief that multimedia applications, due to the high involvement of presentation designers (graphical design, sound design etc) and due to the time-dependent nature of some presentation media, need specific concepts not found in standard modeling languages. To achieve a good interface towards the presentation designers, specific diagrams for presentation and interaction have been proposed above. To reflect the temporal flavor of media, a special temporal structure has been introduced which structures the application into scenes. Within a scene, presentation follows a specific presentation diagram, so the structuring into scenes also assists communication with presentation designers. To summarize the model concepts more abstractly, the classical static structure of the application is enhanced by a sophisticated spatio-temporal structure adequate for multimedia presentations.

Concepts for a tool suite covering modeling and code generation were sketched above. A first prototypical set of such tools is available, however there is much space for improvement. For instance, cross-platform code generation is not yet supported (currently only one platform, dynamic SVG, is supported). But work on other platforms is in progress, and we expect in particular to be able to report success on the integration of the popular Flash platform soon. There are two obvious strains for further work: Detail improvements of modeling language and tool, and evaluation of the approach. Regarding the first aspect, many small improvements are planned, e.g.: Notation for multiple UI objects in presentation diagrams, layout managers, support for UI objects moving their position on the screen (usually animations), constraints and properties of different output devices. Regarding the evaluation of the approach, we plan to use the new

technology in student projects carried out by students of the "Media Informatics" study program at our university, and to compare empirically the achieved performance and quality of results with a more conventional, tool-driven and platform-specific style of multimedia application development.

Besides these more short-term goals for further work, the introduction of a very high abstraction level for multimedia applications also makes it possible to freshly discuss the old question of whether multimedia applications are inherently different from other, more standard pieces of software. The abstract modeling concepts may make it possible to define a formal, mathematical semantics for multimedia applications, and may therefore contribute to the long-term goal of better understanding the underlying principles of this class of computer programs.

10 Acknowledgements

We would like to thank all students taking part in the multimedia development projects for their help to gain additional insights to multimedia development as well as K. Leichtenstern for the help on the tool development.

References

1. Macromedia, http://www.macromedia.com. (2004)
2. W3C: Scalable Vector Graphics (SVG) 1.1 Specification. 2003
3. 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)
4. Object Management Group, MDA, http://www.omg.org/mda. (2004)
5. Frankel, D.S.: Model Driven Architecture. John Wiley (2003)
6. Kleppe, A., Warmer, J., and, B.W.: MDA Explained. Addison-Wesley (2003)
7. 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)
8. Object Management Group: UML 2.0 Superstructure Final Adopted Spec. (2003)
9. Moving Pictures Expert Group (MPEG): MPEG-4 (2002)
10. The Eclipse Project, http://www.eclipse.org. (2004)
11. 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
12. 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)
13. Rahardja, A.: Multimedia Systems Design: A Software Engineering Perspective. In: International Conference on Computers and Education (ICCE) 95 Proceedings. (1995)
14. 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

15. Bertino, E., Ferrari, E.a.: Temporal Synchronization Models for Multimedia Data. TKDE **10** (1998) 612–631
16. Wahl, T., Wirag, S., Rothermel, K.: TIEMPO: Temporal Modeling and Authoring of Interactive Multimedia. In: International Conference on Multimedia Computing and Systems. IEEE Computer Society (1995) 274–277
17. 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
18. Nunes, N.J., e Cunha, J.F.: Towards a UML profile for interaction design: the Wisdom approach. In Evans, A., Kent, S., Selic, B.a., eds.: UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings. Volume 1939. Springer (2000) 101–116
19. Boehm, B.W., Egyed, A., Kwan, J., Madachy, R.J.: Developing Multimedia Applications with the WinWin Spiral Model. In Jazayeri, M., Schauer, H., eds.: Software Engineering - ESEC/FSE '97, 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on Foundations of Software Engineering, Zurich, Switzerland, September 22-25, 1997, Proceedings. Lecture Notes in Computer Science. Spinger (1997)
20. Depke, R., Engels, G., Mehner, K., Sauer, S., Wagner, A.: Ein Vorgehensmodell für die Multimedia-Entwicklung mit Autorensystemen. Informatik: Forschung und Entwicklung (1999) 93–94
21. Marculescu, R., Pedram, M., Henkel, J.: Distributed Multimedia System Design: A Holistic Perspective. In: 2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France. Volume 2. IEEE Computer Society (2004)
22. Mines, R.F., Friesen, J.A., Yang, C.L.: DAVE: A Plug and Play Model for Distributed Multimedia Application Development. In: Proceedings of the second ACM international conference on Multimedia. ACM Press (1994)
23. Koch, N., Kraus, A.: Towards a Common Metamodell for the Development of Web Appliactions. In Lovelle, J.M.C., Rodríguez, B.M.G., Aguilar, L.J., Gayo, J.E.L., Ruíz, M.d.P.P., eds.: Web Engineering, International Conference, ICWE 2003, Oviedo, Spain, July 14-18, 2003, Proceedings. Volume 2722 of Lecture Notes in Computer Science. Springer (2003)
24. Hennicker, R., Koch, N.: Modeling the User Interface of Web Applications with UML. In Evans, A., France, R.B., Moreira, A.M.D., Rumpe, B., eds.: Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists, Workshop of the pUML-Group held together with the UML2001, October 1st, 2001 in Toronto, Canada. Volume 7 of LNI. GI (2001)
25. Gómez, J., Cachero, C., Pastor, O.: Conceptual Modeling of Device-Independent Web Applications. IEEE MultiMedia **8** (2001) 26–39
26. Muller, P.A., Studer, P., Bézivin, J.: Platform Independent Web Application Modeling. In Stevens, P., Whittle, J., Booch, G., eds.: UML 2003 - The Unified Modeling Language, Modeling Languages and Applications, 6th International Conference, San Francisco, CA, USA, October 20-24, 2003, Proceedings. Lecture Notes in Computer Science. Springer (2003)
27. 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