Model-Driven Development of Interactive Multimedia Applications

Towards Better Integration of Software Engineering and Creative Design

Andreas Pleuß

München 2009

Model-Driven Development of Interactive Multimedia Applications

Towards Better Integration of Software Engineering and Creative Design

Andreas Pleuß

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Andreas Pleuß
geboren in Würzburg

München, den 25. Februar 2009

Erstgutachter: Prof. Dr. Heinrich Hußmann

Zweitgutachter:	Prof. Dr. Peter Forbrig (Universität Rostock)
	Prof. Dr. Jean Vanderdonckt (Université Catholique de Louvain, Belgien)

Tag der mündlichen Prüfung: 27. Mai 2009

Contents

List of Figures

List of Tables

Abstract

The development of highly interactive multimedia applications is still a challenging and complex task. In addition to the application logic, multimedia applications typically provide a sophisticated user interface with integrated media objects. As a consequence, the development process involves different experts for software design, user interface design, and media design. There is still a lack of concepts for a systematic development which integrates these aspects.

This thesis provides a model-driven development approach addressing this problem. Therefore it introduces the Multimedia Modeling Language (MML), a visual modeling language supporting a design phase in multimedia application development. The language is oriented on well-established software engineering concepts, like UML 2, and integrates concepts from the areas of multimedia development and model-based user interface development.

MML allows the generation of code skeletons from the models. Thereby, the core idea is to generate code skeletons which can be directly processed in multimedia authoring tools. In this way, the strengths of both are combined: Authoring tools are used to perform the creative development tasks while models are used to design the overall application structure and to enable a well-coordinated development process. This is demonstrated using the professional authoring tool Adobe Flash.

MML is supported by modeling and code generation tools which have been used to validate the approach over several years in various student projects and teaching courses. Additional prototypes have been developed to demonstrate, e.g., the ability to generate code for different target platforms. Finally, it is discussed how models can contribute in general to a better integration of well-structured software development and creative visual design.

Kurzzusammenfassung

Die hier beschriebene Arbeit geht von der vielbeschriebenen Forderung nach einem strukturierten Entwicklungsprozess für Multimedia-Anwendungen aus. Dazu wird mit der *Multimedia Modeling Language* (*MML*) eine graphische Modellierungssprache speziell für Multimedia-Anwendungen vorgeschlagen, sowie ein darauf basierender modellgetriebener Entwicklungsprozess. MML unterstützt eine strukturierte, explizite Integration von Software-Design, User-Interface-Design und Medien-Design.

Eine wichtige Zielsetzung der Arbeit ist die Einbeziehung etablierter Multimedia-Autorenwerkzeuge in den modellgetriebenen Entwicklungsprozess mit MML. Dazu werden aus den MML-Modellen automatisch Codegerüste generiert, die dann direkt im Autorenwerkzeug geöffnet und weiterverarbeitet werden können. Dadurch werden die Vorteile von Modellen und die Vorteile von Autorenwerkzeugen vereint.

MML wird unterstützt durch verschiedene Werkzeuge zur Modellierung und Code-Generierung, die über mehrere Jahre hinweg in verschiedenen Projekten in der Lehre eingesetzt wurden. Weitere prototypische Werkzeuge demonstrieren z.B. die Platformunabhängigkeit der Sprache. Abschliessend wird anhand weiterer Beispiel diskutiert, wie Modelle im allgemeinen zu einer besseren Integration von systematischer Entwicklung und kreativem, graphischen Design beitragen können.

Chapter 1

Introduction

With upcoming graphical and auditive capabilities of personal computers in the mid of the 90s, the term "multimedia" emerged to a hype. The idea of multimedia raised expectations on very intelligent and intuitive user interfaces making use of speech input and output, gestures and complex graphics to provide complex information in a very convenient way, like in science fiction movies. Related to that were overrated expectations on the content and the impact of multimedia systems, like in e-learning where some authors painted the scenario that people will learn much more continuously and efficiently due to omnipresent multimedia learning software. Like any "buzz word" the term "multimedia" was used for product promotion and hence in a very ambiguous way. For example, personal computers just equipped with a CD-Rom drive were called "multimedia systems".

Similar as described by the *Hype Cycle* by Gartner [Gartner], a phase of disillusion followed. Most existing applications could not meet the expectations. While the implementation of multimedia became easier by increasing support in toolkits and programming languages, its usage was still limited because of its complexity [Mühlhäuser and Gecsei96, Bulterman and Hardman05]. In addition, there was no common understanding of its precise meaning. This lead some authors to the statement that "*Multimedia is dead. In fact, it never really existed.*"[Reisman98] (see also [Hirakawa99, Gonzalez00]).

Multimedia Applications Today Nowadays, multimedia capabilities are used in a much more natural and mature way. One reason for this is certainly the evolved implementation support. It has become much easier now (on implementation level) to integrate multimedia and application logic. From the viewpoint of multimedia, this allows to enhance multimedia applications with more sophisticated interactivity and application logic. For instance, multimedia authoring tools like *Flash* [Flaa] nowadays include a powerful object-oriented programming language. From the viewpoint of conventional software development, the better integration leads to easier enrichment of applications with multimedia user interfaces. For instance, the framework *Flex* [Kazoun and Lott07] aims to enable software developers to integrate Flash user interfaces without knowledge on the Flash authoring tool.

A second reason is the evolution in user interface design. In earlier days, complex applications were mainly developed for professional context like business applications. Today, due to the general penetration of computers in all aspects of life, applications target also private everyday life situations. Hence, criteria like the application's usability, likeability, and also its entertainment factor become more and more important [Dix et al.03, Shneiderman and Plaisant04]. On the other hand, the increasing general awareness of the importance of usability leads also to more sophisticated user interfaces in general. Research in the Human-Computer Interaction area, like new interaction techniques or user

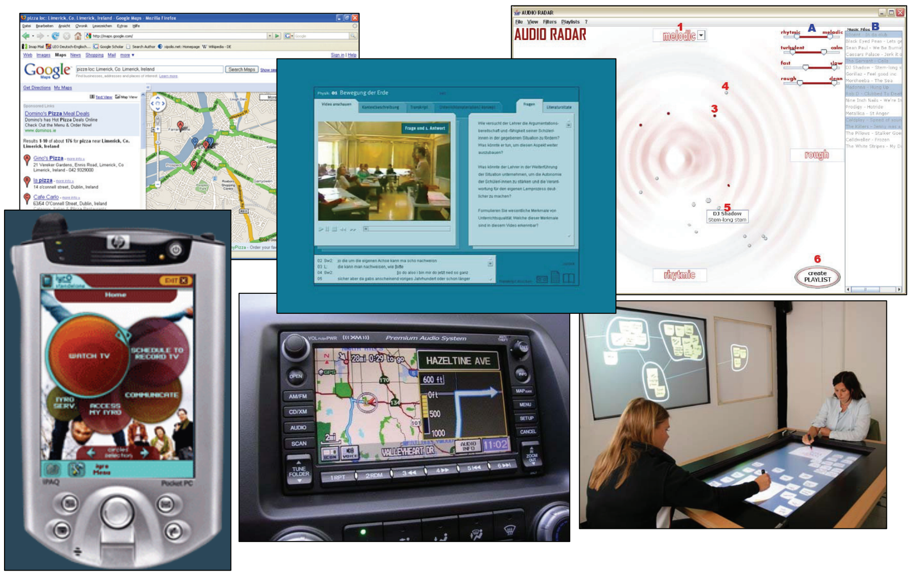


Figure 1.1: Examples for multimedia applications today. (From left to right, starting in the top row: Rich Internet Application [Goo], E-Learning [Jungwirth and Stadler03], Entertainment [Hilliges et al.06], Entertainment on mobile devices [Tavares], Infotainment in cars [Inf], Instrumented Environments [Bra])

interfaces for Ubiquitous Computing, break new grounds by investigating into much more intelligent and complex user interfaces using different media, new interaction techniques, and individual elements precisely tailored to the concrete tasks to be fulfilled by the user [Marculescu et al.04]. In this way, user interfaces step by step approach the initial vision described above.

Accordingly, the application area of multimedia has expanded to almost all areas of software applications. Classical examples include E-Learning, Entertainment, Computer Games, Simulations, and Arts. Today, additional areas are, for instance, Rich Internet Applications like the well-known *Google Maps* [Goo], Car Infotainment Systems, Entertainment Software, also on various devices beyond the Personal Computer, and various research prototypes for new user interfaces or for applications in ambient environments. Figure 1.1 shows some examples.

Multimedia Application Development A ‘multimedia application’ in the sense of this thesis is hence any kind of application with a multimedia user interface. ‘Multimedia user interface’ here basically means that the user interface is not restricted to standard widgets (like buttons, text fields, etc.), but makes use of individual graphics, animations, video, sound, 3D graphics, etc. The underlying application logic can be of any complexity. In particular, in an advanced, highly interactive application the media objects are tightly connected to the application logic: Examples like Google Maps demonstrate that media objects on the one hand can be used as elements for user input, on the other hand they are dynamically calculated and modified by the application logic.

For such kinds of applications there is still a lack of a systematic development process. Existing development methods for conventional software development provide only very limited support for specifying and designing complex media objects. In contrast, multimedia-specific development approaches focus on the creation of media objects but disregard Software Engineering principles. However, the integration of the different development tasks and artifacts is essential for a coordinated, systematic, and finally successful development process. This thesis aims to fill this gap by providing an integrated and tailored model-driven development approach.

Approach in this Thesis This thesis provides a model-driven development approach to address these challenges. To integrate the different developer groups and the artifacts they produce, it proposes a “design phase” to plan and specify the application preliminary to its implementation. For this, it uses a modeling language that integrates software design, user interface design, and media design into a single, consistent language. As no such language exists so far, it introduces the *Multimedia Modeling Language* (*MML*) which integrates different existing modeling concepts, like the Unified Modeling Language (UML) and concepts from model-based user interface development (MBUID), and adds new modeling concepts which are specific for interactive multimedia. The language is defined as a standard-compliant metamodel and is supported by a visual modeling tool.

MML models can be automatically transformed into code skeletons. As MML is platform-independent, it is possible to generate code for any target platform. In particular, existing multimedia authoring tools are supported as target platforms: The generated code skeletons can be loaded directly into the authoring tool where they can be processed and finalized by user interface and media designers. Hence, established professional tools for the creative, visual design are fully integrated into the model-driven development process. This is demonstrated in detail by a fully implemented transformation for the authoring tool Adobe Flash. In this way, the approach supports both: Systematic model-driven development for planning, structuring and integration of the different aspects of multimedia applications *and* professional support for the creative visual media and user interface design.

Thesis Structure The central theme throughout this work is the integration of creative, visual user interface and media design with systematic Software Engineering methods.

At first, chapter 2 examines the basic definitions and the current practice in context of multimedia application development. In turns out that the basic theme – the integration of software development and creative design – is reflected in all aspects of multimedia development. For instance, common implementation technologies are multimedia authoring tools like Flash or Director which strongly support the creative design. However, they do neither support a development process nor a systematic structure of the application in terms of Software Engineering.

Chapter 3 examines this problem in more detail. It turns out that existing literature as well as existing studies on industrial practice clearly confirm the lack of adequate systematic, integrated development methods. In particular, two specific challenges are discussed. First, the need to integrate different kinds of design usually performed by different experts: interactive multimedia application development requires software design, media design, *and* user interface design. Second, the final implementation is often performed with multimedia authoring tools which must be integrated into a development process.

Subsequently, the chapter discusses possible alternative solutions. As result, a model-driven development process seems to be most promising: Models are an excellent tool to plan the overall structure of the application and the coordination between different developer groups. In addition, a specific idea is proposed to integrate existing authoring tools into the model-driven process: From the models, it is possible to generate code skeletons directly for the authoring tools. Thereby, placeholders are generated for the concrete visual media objects and user interface objects. These placeholders can be filled out in the authoring tool, using the authoring tool’s established and powerful support for creative design. However, the placeholders are already integrated into the overall application by the code generated from the models. This means that the models are used to specify and automatically generate the application's overall structure and the relationships between its different parts while the authoring tool is used for the concrete creative design by filling out the placeholders. In this way, the strengths of models (systematic planning of the overall structure) and the strengths of authoring tools (creative design) are combined.

Chapter 4 then analyzes existing modeling approaches which can be used for such a model-driven approach. It turns out, that there is a large amount of modeling approaches which target some aspects of multimedia applications. The chapter gives a comprehensive overview and structures them into three identified research areas: user interface modeling, web engineering, and existing modeling approaches from multimedia domain. However, none of this approaches is sufficient to model interactive multimedia applications: they either support only standard user interfaces or only multimedia documents without advanced interactivity and application logic.

Due to the lack of a sufficient modeling language, the chapters 5 and 6 introduce the *Multimedia Modeling Language* (*MML*). It integrates the concepts found in the related work and extends them where necessary. Therefore, chapter 5 first presents the preliminary considerations for the language design and then introduces concepts for modeling advanced media properties which have not been addressed in existing modeling approaches yet. Subsequently, chapter 6 presents the resulting overall language under strong consideration of existing concepts from chapter 4. it is concluded by an overview on the modeling process and the implemented tool support.

On that base, chapter 7 shows the transformation from MML models into code skeletons for a multimedia authoring tool. The example platform here is Flash, as it is one of the most important professional multimedia authoring tools today. The chapter also illustrates by screenshots how to work with the skeletons directly in the authoring tool.

Chapter 8 addresses the validation of the proposed approach. First, several transformations to other target platforms are shown to demonstrate the platform independence of MML. The next section describes several projects for the external validation followed by a section on the internal validation.

Chapter 9 provides an outlook by generalizing the idea from this thesis to combine the strengths of models and of tools for visual creative design. The model thereby acts as a kind of "central hub". The idea is illustrated by a real-world project in industry which provides a first step into this direction.

Finally, chapter 10 summarizes the contributions of the thesis.

Chapter 2

Interactive Multimedia Applications

This chapter discusses the characteristics of multimedia applications and elaborates the required definitions for this thesis. Therefore, the first section discusses basic terms and definitions based on existing literature. As this thesis deals with the development of multimedia applications, the second section provides a more detailed look on multimedia applications from that point of view. In addition, a third section introduces existing technologies for implementation and selects the platform Flash as running example in this thesis. Based on the discussion above, the fourth section presents a classification of multimedia applications from the viewpoint of development. Finally, the fifth section summarizes the most import findings and derives a resulting definition for this thesis.

2.1 Basic Terms and Definitions

This section introduces central terms like *multimedia*, and *multimedia applications* and shows how they differ from associated terms like *multimodality* and *hypermedia*.

2.1.1 Multimedia

The term *medium* has its origin in the Latin word *medius* which means 'in the middle'. Today it has in general the meaning of 'intermediary' or 'mean' but is overloaded with many specific meanings in different contexts. Concerning the context of computer technology, the MHEG standard [ISO97b, Meyer-Boudnik and Effelsberg95] distinguishes between five categories of *technical media*. Figure 2.1 illustrates this classification with examples. The term *multimedia* applies to the category of *perception media*, which is related to the way how a human being can perceive a piece of information, e.g. visually or auditory. Visual media can then be further refined e.g. in text, still images, animations, and movies. Auditory media are music, sound, and speech [Steinmetz00]. The remaining human senses – touch, smell, and taste – are still rarely used for digital media today. These kinds of media are sometimes also referred to as *media types* [Engels and Sauer02] to distinguish from *media objects*. A media object is a concrete unit of information on one or more channels of information [Henning01], i.e. of a specific media type.

Each media type is associated with one or more dimensions: one to three spatial dimensions and one optional temporal dimension. Sound has one spatial dimension, images have two, and holograms have three spatial dimensions. In contrast to still images, a video has an additional temporal dimension.

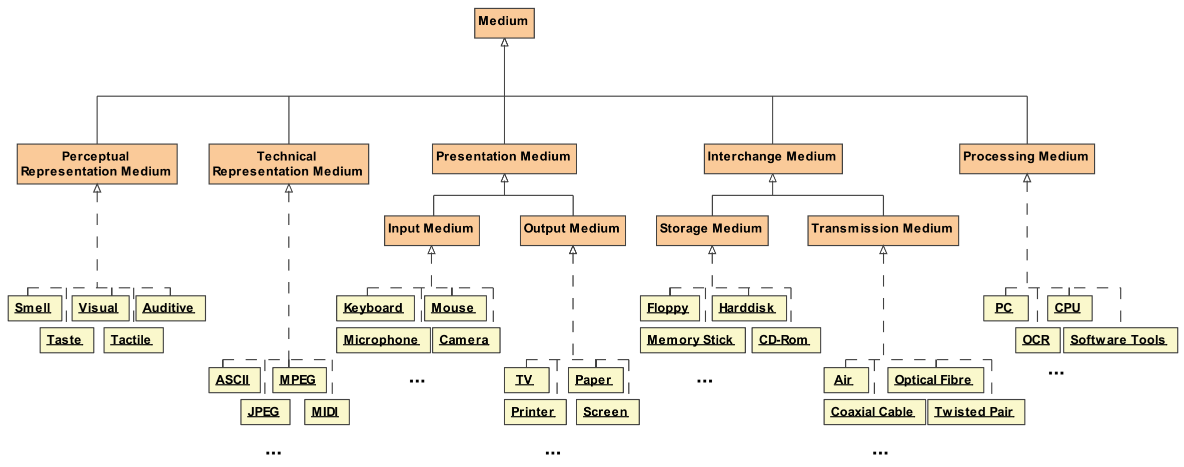


Figure 2.1: Technical media: Classes and examples based on [Hoogeveen97] and MHEG [ISO97b].

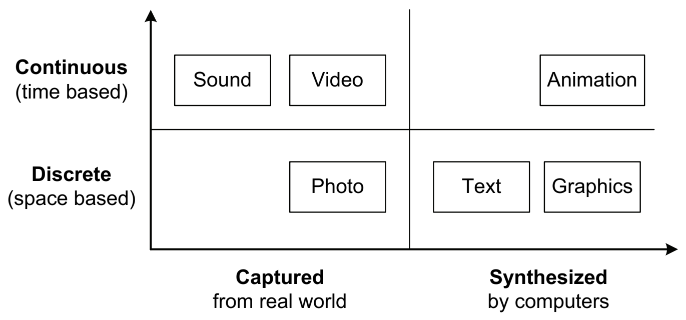


Figure 2.2: Classification of media according to [Fetterman and Gupta93].

A classification of media useful for the context of this work is provided by [Fetterman and Gupta93]: They distinguish on the one hand between discrete media and continuous media. *Continuous media* have a temporal dimension, i.e. their representation changes dynamically over the time, like audio, video or animation. *Discrete media* are time-independent, like text, image and graphics. On the other hand they distinguish between captured media and synthesized media. *Captured media* are captured via sensors and digitalization from the real world, like usually audio, video, and images (photos). *Synthesized media* are created with the computer, like text, graphics, and animation. Thus, synthesized media often have an explicit inner structure as it can be stored during the development process (for instance, a vector graphics consists of graphic primitives while for captured images this information is initially not available). A corresponding matrix categorizing media (types) is shown in figure 2.2.[1]

Various definitions have been provided for the term *multimedia*. A popular definition[2] is provided by Steinmetz and Nahrstedt [Steinmetz and Nahrstedt04]:

> A multimedia document is characterized by information which is coded in at least one continuous (time-dependent) and one discrete (time-independent) medium.

As observed by Boll [Boll01] most alternative definitions mainly emphasize on the integrative

[1]Although sound is often captured in practice, it can also be synthesized like MIDI files.
[2]sometimes referred to as de-facto convention [Zendler98]

aspect of multimedia documents like in the early work of [Bulterman et al.91]:

> Multimedia documents consist of a set of discrete data components that are joined together in time and space to present a user (or reader) with a single coordinated whole.

An analysis from Boll [Boll01] of existing definitions in research work and standards identifies as the main characteristics the integrative aspect in conjunction with temporal and spatial composition:

> A multimedia document is a media element that forms the composition of continuous and discrete media elements into a logically coherent multimedia unit. [Boll01]

2.1.2 Multimedia vs. Multimodality

A term related with multimedia is *multimodality*. In the context of human-computer interaction, *modality* refers to the human communication channels for input and output, like seeing, hearing, touch, taste, smell. Thus, multimodality basically has a very similar meaning than multimedia. According to some authors multimedia implies "multisensory", "multimodal", and "multichannel" [Hoogeveen97]. However, the common understanding in the research community often is often different. [Coutaz and Caelen91] provides a taxonomy of multimodality and discusses the difference to multimedia. According to them, multimodal systems analyze the semantics of information while multimedia systems only encapsulates the information into typed chunks. These chunks can be manipulated but a multimedia system does not consider their inner semantics. However, this distinction might not be always sufficient as today also typical multimedia applications (e.g. [Grana et al.05, Felipe et al.05, Ozcan et al.05]) and standards like MPEG-7 [Salembier and Sikora02] deal with the semantics of data.

Typical examples of multimodal applications which formed the common understanding of multimodality usually combine speech input with pointing or gestures [Bolt80, Cohen et al.97]. [Coutaz and Caelen91] mention this difference stating that “multimedia information is the subject of the task (it is manipulated by the user) whereas multimodal information is used to control the task”. Other work goes one step further into this direction and just associates multimodality as input and multimedia as output:

> Multimodal systems process combined natural input modes – such as speech, pen, touch, hand gestures, eye gaze, and head and body movements – in a coordinated manner with multimedia system output. [Oviatt99]

The understanding in this thesis conforms to this last definition which allows a clear distinction between the development support: Multimodality requires interaction design and complex algorithms for analyzing the semantics of user input while multimedia focuses on design and integration of media objects.

2.1.3 Multimedia Applications

The work described so far understands multimedia as kind of documents. Even later research approaches like [Boll01] restrict user interaction to navigation, adaptation of the presentation (e.g. to change the resolution of a movie), and basic player control (e.g. to play and pause a multimedia presentation). However, this is a very restricted understanding of “interactivity”. Some critical authors even claim that aspects like navigation can not be called interactivity at all as interactivity rather means “controlling the object, subject or contents” represented by the user interface [Schulmeister03].

Thus, the viewpoint of understanding multimedia as kinds of documents is clearly no longer sufficient today. In contrast, the original vision associated with multimedia, to create highly interactive, intelligent, and usable systems, often requires a tight integration with complex application logic. Examples are many of the most successful applications these days, like Google Maps [Goo] and others mentioned in section 1.

Basically, multimedia can be combined with any kind of application today. This means, that an application provides a (usually more sophisticated) multimedia user interface instead of standard user interfaces only. Typical reasons for this are [Hoogeveen97]:

- to enhance efficiency and productiveness of the user interface,
- to achieve more effective information and knowledge transfer, or
- an enhanced entertainment value.

This leads to the term *multimedia application*: Generally, an application (or application software) is defined as a kind of software which directly serves the user [Microsoft03]. A multimedia application in broadest sense is any kind of software which directly serves the user and provides a multimedia user interface.

According to [Steinmetz and Nahrstedt04], a multimedia application enables the user to interact with multimedia data.

A more comprehensive definition of multimedia applications is provided by [Engels and Sauer02]:

> Multimedia applications can be defined as interactive software systems combining and presenting a set of independent media objects of diverse types that have spatio-temporal relationships and synchronization dependencies, may be organized in structured composition hierarchies, and can be coupled with an event-based action model for interaction and navigation.

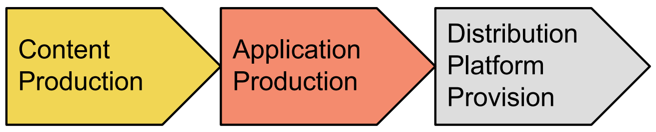
Consequently, multimedia applications integrate two aspects [Mühlhäuser and Gecsei96]: multimedia authoring (which corresponds to the document character) *and* software programming (like for conventional applications). This is an important characteristic which has strong impact on multimedia application development and has to be frequently taken up during this thesis.

Section 2.5 will come back to the term "multimedia application" and provide a definition for this thesis from the viewpoint of development.

2.1.4 Multimedia Applications vs. Web and Hypermedia Applications

Web and multimedia are often named in the same breath. This is probably caused by the fact that in the 90's both were dominating innovations and future applications were expected to support both [Lang01a]. In fact, web and multimedia applications have in common that they differ from traditional business applications, e.g. in terms of the target audience and their entertaining character [Balzert and Weidauer98]. In addition, common implementation technologies like Flash (sec. 2.3.3) support both, web and multimedia. Thus, companies focusing on these technologies often developed both kinds of applications. However, despite these definitely relevant commonalities in the development, web and multimedia are still independent properties of an application.

[Kappel et al.03] defines web applications as software systems which base on specifications of the World Wide Web Consortium (W3C, [WWWa]) and provides web-specific resources like content and services which can be used via a web browser. Often they are conventional business applications and do not provide any specific media objects. Moreover, until the last few years HTML was still the dominating implementation technology which provides only poor support for multimedia. On the



Examples:

Companies:	Media Industry	Multimedia Companies	Service Providers
Results:	Text, Image, Video, 3D Graphics	Multimedia Application	Infrastructure
Development Tools:	Photoshop, Illustrator, 3D Studio Max	Authoring Tools (see section 2.3)	C/C++

Figure 2.3: Multimedia applications in the value chain.

other hand, many multimedia applications have no additional need for web support compared to other kinds of application. Thus, in the context of application development, web and multimedia properties have to be distinguished.

Applications combining the hypertext concept with multimedia are referred to as *hypermedia* applications. [Lang01a] defines hypermedia information systems as the intersection of hypertext systems, web information systems, and multimedia systems, i.e. requiring hypermedia to fulfill all three properties. However, often hypermedia is described as superset of multimedia and hypermedia applications [Zendler98, Gonzalez00]. In practice, often the latter interpretation is used which means that the term hypermedia is applied to all applications with either hypertext and/or multimedia characteristics.

2.2 Multimedia Application Development

This work deals with the development of multimedia applications. Thus, a more detailed characterization of multimedia applications is conducted from that point of view to gain a better understanding of multimedia applications.

The upper part of figure 2.3 shows multimedia applications in the value chain and assigns examples to each step according to [Hussmann07, Osswald03, Steinmetz and Nahrstedt04]. *Content Production* means the production of the actual content, e.g. a movie or the learning content in a learning application. Often, the content itself has some economic value. As the content has to be represented by any media, Content Production in our context means the production of media objects, e.g. text, videos, 3D animations, etc.

Application Production means the production of the multimedia application from single media objects. It includes the selection and integration of media objects, creating the application's user interface and interaction, as well as the application logic. Finally, *Distribution Platform Provision* refers to those services which are necessary to run the multimedia application in the target environment. This includes the infrastructure of networks and target platforms, hosting on servers, etc.

"Multimedia Applications" as defined in this work are thus the result of the Application Production step. Infrastructure software and system software required for Distribution Platform Provision are *not* understood as multimedia application here. This means that a multimedia application (conforming to the definition of "application" given above) directly serves the user and thus has itself a user interface with multimedia properties.

The last row in figure 2.3 shows examples for implementation technologies typically required in each step. They are explained in the following.

The production of media objects in the first step requires very different processes and tools depending on the media type. Synthesized media is created and processed by specific authoring tools. For example 3D animations are created with 3D authoring tools like *3ds Max* [3DS] or *Maya* [May]. 2D animations are often created with *Flash* [Flaa]. Graphics are created for instance with *Illustrator* [Ill] or *CorelDRAW* [Cor].

Captured media is usually captured from the real world but further processing and post-editing is usually performed in authoring tools as well. Video is edited in video editing software like *Premiere* [Pre] or *Final Cut* [Fin]. Sound is edited (and can also be created) with audio editing tools and MIDI Sequencers like *Cubase* [Cub] or *Reason* [Rea]. Images are edited with image editing tools like *Photoshop* [Phoa] or *Paint Shop Pro* [Pai]. An comprehensive overview on such tools can be found e.g. in [Zendler98].

As shown by these examples, the production of media objects is supported by various very specific authoring tools for the different media types. These authoring tools focus on efficient, creative, and visual media design and are well established. At this point in the development, only the creative media design is important – programming and software development knowledge are hardly relevant there.

In contrast, software for the last step in the value chain, Distribution Platform Provision, is usually developed like conventional software using conventional programming languages like C/C++. This holds also for infrastructure software in the broadest sense like *Multimedia Database Management Systems* (*MMDBMS*, [Narasimhalu96, Zendler98] or player software like the *Flash player* [Adobed]. Such software provides general functionality independent from the actual content and the actual application. The development of such software requires mainly programming skills and only few (or no) creative visual design.

The development tasks in the Application Production step (the focus of this work) combine the aspects of both areas: On the one hand, creative design is required. The developers have to create a user interface and to integrate the content. Therefore, they often have to post-edit existing content or to create additional content. On the other hand, the developers have to create the application logic and the interactivity similar to conventional software development. Thus, authoring tools and conventional programming are both required in this step. An overview on the different implementation technologies and tools for application production is given in the next section 2.3.

2.3 Implementation Technologies and Tools

This section briefly provides an overview on typical technologies and tools for implementing interactive multimedia applications, i.e. for the second step in figure 2.3 (the focus of this thesis).

They can be classified into three categories [Engels and Sauer02]:

1. Frameworks and APIs,
2. Declarative Languages, and
3. Authoring Tools.

The next three sections give a brief overview on each category, with an emphasis on authoring tools, and and shows some selected examples.

2.3.1 Frameworks and APIs

There are several frameworks and APIs (Application Programming Interface) supporting the development of multimedia applications with conventional programming languages. They usually address specific programming languages or target platforms.

DirectX A popular example is *DirectX* from Microsoft [Microsofta]. It is a collection of APIs for the operating system *Windows* and the game console *XBox*. The APIs provide support for all media types, including creation and rendering of 2D and 3D graphics and animation (*DirectDraw*, *Direct3D*, *DirectAnimation*), playback and recording of audio (*DirectSound*, *DirectMusic*), playback and processing of video and other streaming media (*DirectShow*; now part of the platform SDK). The basic functionality of DirectX is rather low level and can be classified as system software. It abstracts from concrete hardware devices and allows the programmer to access hardware components and functionality via standard interfaces defined by DirectX. However, it also provides more high level multimedia functionality and is often directly used by developers. The API provides objects and interfaces in the style of the Microsoft *Component Object Model* (*COM*, [Box98]). It can be used with different programming languages like C++, Visual Basic or C#. DirectX is commonly used for game development [Sherrod06, Luna08] and other performance-dependent applications.

Java Media Framework An example for the programming language Java are the *Java Media APIs* [Microsystems] provided by Sun. Similar to DirectX they consists of different APIs for different media types. It includes among others the *Java Media Framework* (*JMF*) which supports video and audio, the *Java Advanced Imaging API* (*JAI*) supporting images, the *Java 2D API* supporting 2D graphics and images as well, and the *Java 3D API* supporting 3D graphics and animations. Some of them like Java 2D and Java Sound from the JMF have become part of the standard Java distribution. There is no specific support for 2D animations. The APIs are platform-independent. However, there are performance packs for some selected platforms (Windows, Linux, and Solaris) which provide access to the platform's native media libraries.

The programming style is rather heterogeneous depending on the media type. The JMF interprets the media objects as abstract data sources and data sinks which are processed in processing chains. They use several software design patterns to implement the concepts. The API structure and the available examples mainly target the development of applications to create and edit media objects – i.e. software for the first step in the value chain from section 2.2 – instead of applications with multimedia user interfaces. This might also be a reason why 2D animation is not supported although it is important for multimedia user interfaces and e.g. a core element in professional authoring tools like Flash (see sec. 2.3.3).

Piccolo Besides the Java Media APIs, several other Java frameworks exist which mainly focus on a specific media type. An example for 2D animations is *Piccolo* developed by Human-Computer Interaction Lab at University of Maryland [Bederson et al.04, of Maryland]. It is the successor of the framework *Jazz* provided by the same group and focuses in particular on zoomable graphical user interfaces. There is also a version for the programming language C# available.

In general, Piccolo structures user interface elements in terms of a *scene graph*, i.e. as a hierarchical structure of nodes (figure 2.4). Piccolo nodes are for instance a graphic (class PPath), an image (PImage) or text (PText)). A camera node (PCamera) represents a viewport to other nodes and can apply transformations on them. A canvas node (PCanvas) represents a canvas where e.g. conventional Java Swing widgets can be placed on. Piccolo also allows to create custom node types, for instance

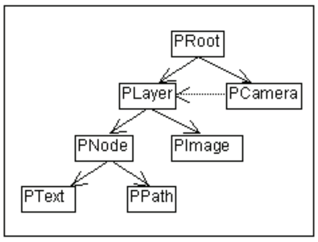


Figure 2.4: Structure in the Piccolo framework

by class inheritance. In that way the framework supports implementing graphics and animations for user interfaces on a high level of abstraction.

Java and Piccolo will be used as an example platform later in section 8.1.

2.3.2 Declarative Languages

Some languages exist which aim to support declarative specification of multimedia applications. They adhere to the viewpoint that multimedia applications can be seen as kind of documents (see sec. 2.1.3). Interactivity and application logic are sometimes supported by the declarative language itself but often added by external program code e.g. written in a scripting language.

Basically, it can be useful to use a declarative language in combination with an authoring tool, analogous to e.g. visual tools for HTML. This gives the developer the possibility to edit the documents either visually using the authoring tool or directly in the declarative code. As several declarative formats are also defined as official standard, this would cause the effect that the authoring tool becomes compliant to a standard. However, most existing professional authoring tools use proprietary binary file formats yet (see sec. 2.3.3). SMIL (explained in the following) is also supported by a commercial tool but this has not become very relevant in professional practice today.

SMIL An important standard in this category is the *Synchronized Multimedia Integration Language* (*SMIL*, [Bulterman et al.05, Bulterman and Rutledge04]), an XML-based language defined by the World Wide Web Consortium (W3C). It supports the integration and synchronization of different multimedia objects into a whole multimedia application in declarative way. Basic concept is the spatio-temporal integration of media objects. For the spatial aspect, each SMIL document specifies multiple *regions* on the presentation screen. The different media objects then can be assigned to these regions. For the temporal aspect it is possible to define e.g. sequential or parallel presentation of different media objects.

It is possible to define event handling for user events like mouse clicks, e.g. to start and stop the presentation. However, complex behavior it not intended in SMIL. As SMIL is an XML format it can of course be accessed by scripting and programming languages like any other XML document, e.g. using JavaScript. However, there is no further support for scripting language integration into SMIL presentations. Thus, SMIL focuses on multimedia presentations but hardly on interactive applications as considered in this work. SMIL presentations can be played with browsers and with several media players like the RealPlayer or Quicktime Player.

There is also a commercial authoring tool for SMIL, called *GRiNS* [Oatrix, Bulterman et al.98], provided by the company Oatrix which has its background in the Human-Computer Interaction research group at CWI [CWI]. It allows to visually define the spatial layout and to define the presentation's temporal structure by visually arranging media objects on a timeline. It is also possible to view

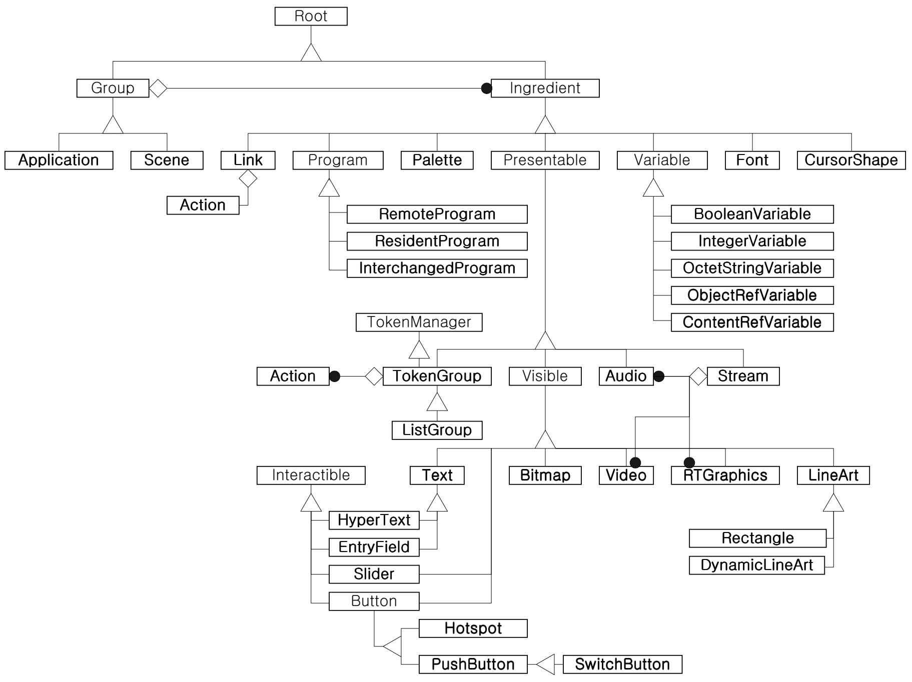


Figure 2.5: MHEG-5 class hierarchy from [MHE].

and edit the corresponding SMIL code directly so that developers can choose the preferred view for a task.

MHEG-5 Another standard in this category is the MHEG-5 standard. It is mainly used for interactive television. It supports audio, video, graphics, bitmaps, and text. It also provides interactive user interface objects like buttons, sliders, or text input, as well as different kinds of events. However, detailed specification of behavior is not defined in the standard and depends on the engine which executes the MHEG application [Marshall08]. MHEG defines a class hierarchy for multimedia applications shown in figure 2.5.

2.3.3 Authoring Tools

The third category are multimedia authoring tools. They are very important in practice as multimedia application development is a creative, visual task and should thus be supported by visual tools. Moreover, developers like multimedia designers and user interface designers often are not familiar with text-based declarative languages or programming languages. For these reasons, authoring tools are an important implementation technology in industrial practice [Britton et al.97, Tannenbaum98, Engels and Sauer02, Bulterman and Hardman05, Hannington and Reed07] and will be considered more in detail in this work.

Advanced authoring tools allow to embed programming code into the developed application to define the application logic. It is also possible that the authoring tools just encapsulates a declarative language or programming language. In that case the same result can be created by either using the authoring tool or by editing the respective language directly (see sec. 2.3.2).

An important challenge in designing a visual authoring tool is to find a suitable representation of the application's temporal behavior. Existing authoring tools are thus frequently classified according to their *authoring paradigm* (or *authoring metaphor*) used to solve this challenge. A common classification [Boles and Schlattmann98, Henning01, Engels and Sauer02] distinguishes between:

- *Frame-based* (*screen-based*, *card-based*) authoring tools,
- *Graph-based* authoring tools, and
- *Timeline-based* authoring tools.

The following sections briefly introduce each of these classes. An additional analysis and comparison of the different classes can be found in [Bulterman and Hardman05]. A reference model for authoring tools is presented in [Lorenz and Schmalfuß98].

Frame-based Authoring Tools

Frame-based authoring tools represent the application by different frames or screens which contain the actual user interface content and the spatial layout. When the application is executed the different screens are by default presented in sequential order. It is usually possible to control the order of the frames by additional scripting code or hyperlinks. Actually, all classes of authoring tools are frame-based but the other classes use an additional visual metaphor (flowchart or timeline) to specify the frame's order and duration.

The content of each frame is usually edited visually by dragging and dropping user interface elements provided by the tool. The properties of each user interface element are displayed and can be edited in a property window. Furthermore it is possible to add script code to the user interface elements, e.g. to handle user input.

There are many examples which fall into this category. A classical example is *Apple HyperCard* [Hyp]. In HyperCard the application is represented as a stack of cards. Each card corresponds to a screen shown to the user. It is possible to define a background for the cards containing user interface elements common to multiple cards. Supported user interface elements are images, buttons and textfields. Buttons and textfields can be associated with scripts which are specified in the built-in object-oriented scripting language *HyperTalk* [Apple88]. Scripts can either be event handlers associated with user interface events, like clicking a button or opening a new card from the stack, or functions to be called from other elements. The scripting code can control the application's navigation as well as read and write the properties of user interface elements.

Over the years there has been a large number of frame-based authoring tools. Most of them have the same basic functionality like HyperCard. A popular example with practical relevance is *Toolbook* [Too] originally produced by *Asymetrix* and now by *SumTotal Systems*. It is mainly used to create e-learning content like interactive multimedia presentations. Advanced features of the current version 9.5 are for instance: templates for various types of questionnaires, easy creation of software simulation (e.g. for user interface prototyping) through a screen recorder and a specific simulation editor view, and support for multiple target platforms like mobile devices.

Graph-based Authoring Tools

Graph-based authoring tools provide the same basic functionality like frame-based tools but in addition visually represent the application's temporal behavior in terms of a graph. The most popular example ([Britton et al.97, Hannington and Reed07] for this class is *Authorware* [Aut] from *Adobe* (formerly *Macromedia*) which uses flowcharts. Thus, most authors call this class more specifically *flowchart-based* authoring tools.

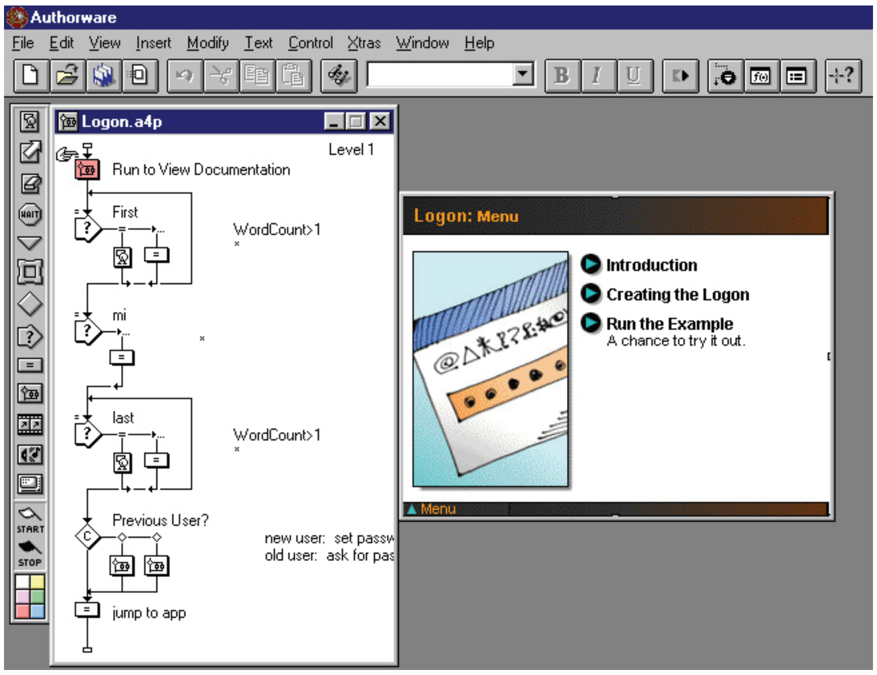


Figure 2.6: A screenshot of Adobe Authorware.

Figure 2.6 shows a screenshot of Authorware. The window on the left hand side shows the flowchart. The nodes in the graph are called *Icons* and define the content and behavior of the application to be developed. Several icons represent content to be displayed on the application's user interface: A *Display Icon* is associated with text and graphics. The *Movie Icon*, the *Sound Icon*, and the *Video Icon* represent other media content. In figure 2.6 the window on the right hand side shows the content associated with the *Display Icon* selected in the flowchart. There are also predefined components (*Knowledge Objects*) providing common functionality like multiple choice tests for e-learning applications.

Other icons can be used to manipulate existing content: The *Animation Icon* allows to define animations while the *Erase Icon* allows to remove content from the application's user interface. The *Decision Icon* is used to control branch the flowchart based on variable values or calculations. The *Calculation Icon* is used to calculate values and manipulate data or global variables. User interaction is supported by the *Wait Icon* which causes the application to wait for user input and the *Interaction Icon* which is used to branch the flowchart depending on the user's input. It is also possible to structure the flowchart hierarchically to handle complex flows.

Authorware includes a scripting language (*Authorware Scripting Language*, *AWS*) and supports also Java Script in the latest versions so that it is possible to script more complex functionality. Basically, the flowchart in Authorware visually defines the application's basic behavior and can be interpreted as a kind of visual programming. Of course, complex applications may require much scripting code so that the flowchart becomes less meaningful.

Similar to Toolbook presented above 2.3.3, Authorware is mainly used fro creating applications with limited interaction. The main application areas are e-learning applications and multimedia presentations like tutorials, help systems, or product presentations. In addition, it is often used for user interface prototyping [Britton et al.97, Hannington and Reed07].

There are only few other graph-based authoring tools besides Authorware. [Bulterman and Hard-

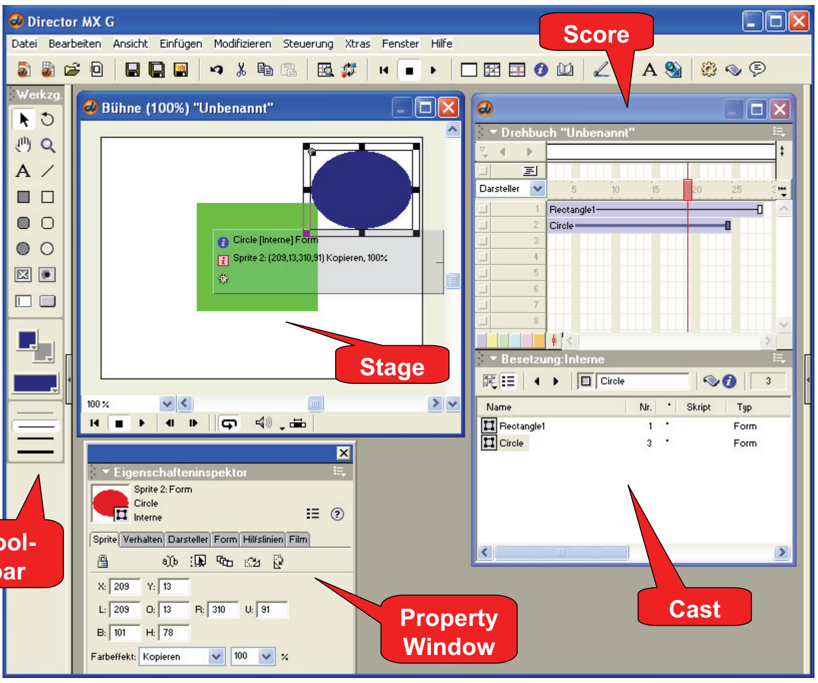


Figure 2.7: A screenshot of Adobe Director.

man05] mentions two approaches from academic area: *Firefly* [Buchanan and Zellweger05] provides automatic "temporal layout" for multimedia documents based on temporal relationships which can be specified as directed graph by the developer. *Eventor* [Eun et al.94] aims to combine the flowchart-based and the timeline-based (see below) authoring paradigm. In addition, [Bulterman and Hardman05] mentions that the concept of Timed Petri Nets has been discussed extensively in research literature as candidate for specifying temporal behavior, however, there is no implementation as an authoring tool yet.

Timeline-based Authoring Tools

Timeline-based authoring tools provide the same basic functionality like frame-based tools but additionally visually represent the application's temporal dimension by the metaphor of a timeline. Usually the timeline consists of several tracks. Each track is associated with some content of the application, e.g. a media object. The timeline visualizes the periods when a certain track becomes active, e.g. is displayed or played.

Many of todays most popular professional authoring tools are timeline-based. *Director* and *Flash*, both produced by *Adobe*, are the most important examples from this category [Hannington and Reed07]. Thus, they are both briefly introduced here. The subsequent comparison shows, that Flash is probably the most important professional authoring tool today. For this reasons it is selected as example platform for this thesis.

Director Figure 2.7 shows an annotated screenshot of Director. It uses metaphors from the area of movie production. The window on the bottom right hand side shows the *Cast Members*. These are arbitrary media elements to be used in the application. A *Sprite* is an instance of a media element (i.e.

instance of a Cast Member) on the stage. The stage (top left in fig. 2.7) represents the application's user interface.

The timeline is called the *Score* (top right in fig. 2.7). It is horizontally divided into *Channels* which are associated to Sprites. The channel in the score visualizes when and how long its associated Sprite appears on the user interface. In addition, there are special Effect Channels for effects and filters which apply to the whole Stage. In horizontal dimension the Score is divided into *Frames*. A frame represents a point of time. The *playback head* on top of the Score determines the frames currently displayed on the stage. It is possible to place the playback onto a frame to shows or edit its content. Playing the application (either for testing purpose in the authoring tool or when executing the final application) means that the playback head moves along the timeline and frame after frame is displayed (i.e. the corresponding content on the stage).

A *Keyframe* is a kind of frame where the associated content on the stage has been explicitly defined by the developer. The developer can specify any frame to be a Keyframe. The content of other (simple) frames is automatically derived from its foregoing Keyframe and can not be directly manipulated by the developer. It is possible to animate the content on the user interface by interpolations (called *Tweening* in Director). Tweenings are always defined between two Keyframes (an example for Flash is shown later in figure 7.2).

Director supports a scripting language called *Lingo*. It is an object-based programming language and can be used to control the application, its content and their properties, as well as to add interaction and application logic. Scripts can be added only as event handlers. However, Director triggers an event when the application starts which can be used as kind of "main" method.

Director applications are compiled into the *Shockwave* format which is interpreted by a player. The player is available as plugin for web browsers. It is also possible to export the application as an executable file which is often used to distribute Director applications as multimedia CD-Roms.

Flash The Flash authoring tools provides very similar authoring concepts like Director. In the last years it has become a very popular platform. The term "Flash" is often also used to denote the technology as a whole or the format of the resulting applications. In fact, Flash is the name of the authoring tool which uses a proprietary file format with the file extension FLA. For execution the FLA files have to be compiled into Shockwave Flash files (SWF)[3]. SWF is a binary file format interpreted by the Flash player which is available as plugin for web browsers.

Compared to Director, Flash provides exceeding support for creating 2D vector graphics and animations. An important concept are *MovieClips*. Each MovieClip owns a local timeline of its own. The frames on its internal timeline can contain any kind of content just like the application's main timeline, i.e. graphics, animations, audio, video, etc. It is also possible to hierarchically nest multiple MovieClips up to any depth. Once a MovieClip has been created by the developer it can be instantiated multiple times on the stage or within other MovieClips. In this way it is possible to create animations of any complexity.

The scripting language in Flash is called *ActionScript*. The first version of ActionScript was an object-based scripting language. It is close to the third edition of the ECMAScript standard [Ecm99]. ActionScript2 introduced in 2003 provides in addition object-oriented mechanisms. It is possible to define class files (similar to class files in Java) and to associate them with a MovieClip. This causes that every instance of this MovieClip is associated with a corresponding ActionScript object. Besides, scripts can be added to frames on the timeline or as event handler to media instances on the stage. ActionScript enables to control the whole application and its contents. It is thus possible to

[3]Not to be confused with the format of Director which is called *Shockwave* only.

develop a complete Flash application only by programming ActionScript and without usage of the visual authoring tool features. In the latest version of ActionScript, ActionScript 3, the step towards object-orientation has been completed and several inconsistencies from earlier versions have been removed.

More detailed information on the Flash authoring tool will be given later in chapter 7.1 as Flash is selected as example platform for this thesis. The next section explains the reasons why Flash is currently the probably most important authoring tool in industrial practice.

Director vs. Flash According to studies in industry [Britton et al.97, Hannington and Reed06, Hannington and Reed07] as well as personal experience, the most important authoring tools are Authorware, Flash, and Director. Authorware seems to be less optimal as typical example platform for this thesis as its graph-based paradigm is quite exotic and it is also rarely used to create highly interactive applications. The following paragraph will compare Flash and Director and give a brief overview on their background.

Director was one of the first products of the company *MacroMind* which was renamed to *Macromedia* in 1992 after a merge with the producer of Authorware (see sec. 2.3.3). The first version emerged in 1997 from *FutureSplash*, a tool for 2D animations by the company *FutureWave Software* which was acquired by Macromedia. While Flash originally addressed vector graphics and animations only, Director has always been developed as a multimedia authoring tool. Thus, in the nineties Director was the probably most popular authoring tool for professional multimedia developers [Britton et al.97, Hannington and Reed07].

In the end of the nineties Macromedia put their focus towards web technologies and emphasized on Flash. After Macromedia was acquired by Adobe in 2005 this trend continues. Since 2002 only two new versions of Director have been released (2008: version 11) while four version of Flash have been released during the same period (2008: version 10, called *Flash CS4*). As mentioned above major revisions have been made on ActionScript which has now emerged to a fully object-oriented Java-like programming language. In general, the technology around Flash seems to move more towards better support for programming and software development. An important example is the *Flex* framework [Kazoun and Lott07, Gruhn07]. This framework enables conventional software developers to develop applications with Flash user interfaces in a conventional Eclipse-based development environment independent from the Flash authoring tool. Flex is intended to be used for development of so-called *Rich Internet Applications* (see sec. 4.2). However, the drawback in terms of creative media design and user interface design is that there is no visual development support for Flex.

According to a study published by Adobe the browser penetration on desktop PCs currently lies at already around 99% [Adobee]. Moreover, Adobe aims to establish Flash as platform for multi-platform development[4]. The *Flash Lite* player is a lightweight version of the Flash player for mobile phones and other devices with limited computing power (see 8.1.3).

In summary, Director is the more traditional multimedia authoring tool and is still relevant. However, there seems to be a clear trend that Adobe relies more and more on Flash. Flash provides various new features and extensions, like ActionScript 3 or multi-platform development, which are not available for Director and seems thus more promising for the future. Thus, Flash is chosen in this work as best example for an up-to-date professional authoring tool.

[4]The media informatics group takes part in a development project initiated by Adobe which makes use of multi-platform support

2.4 Classification

The introduction in chapter 1 has already mentioned several typical examples for multimedia applications today. This section aims to provide a more detailed understanding of the spectrum of multimedia applications by a suitable classification.

2.4.1 Existing Classifications

Two kinds of classifications can be found in the literature: The first type are classifications based on the application domain. The second type are classifications based on multiple facets. Both are explained in the following.

Classifications based on the Application Domain A large part of the existing literature on multimedia applications addresses the spectrum of applications by lists of examples which are classified into some larger example domains. For instance, [Boll01] lists the areas:

- Multimedia teaching and training
- Distributing and trading of multimedia content
- Mobile multimedia applications

[Tannenbaum98] identifies six application areas:

- Scientific Data Analysis, Research and Development, Experimentation, and Presentation
- Instruction in School and Elsewhere
- Business Applications
- Entertainment
- Enabling Technology for Persons with Special Needs
- Fine Arts and Humanities

As part of a detailed taxonomy (explained below) [Hannington and Reed02] proposes:

- **Multimedia information systems:** databases, information kiosks, hypertexts, electronic books, and multimedia expert systems
- **Multimedia communication systems:** computer supported collaborative work, videoconferencing, streaming media, and multimedia teleservices
- **Multimedia entertainment systems:** 3D computergames, multiplayer network games, infotainment, and interactive audio-visual productions
- **Multimedia business systems:** immersive electronic commerce, marketing, multimedia presentations, video brochures, and virtualshopping
- **Multimedia educational systems:** electronic books, flexible teaching materials, simulation systems, automatic testing, and distance learning

While these classes are certainly typical for multimedia it still raises the question whether they are complete and how these classes would be located within the spectrum of all possible application software. The thesis [Kraiker07] supervised by the author of this thesis examines this question. In a first step, Kraiker selected common taxonomies for software in general mainly aggregated from the taxonomies in [Klußmann01, Staas04]. In a second step, Kraiker sorted the examples given in [Tannenbaum98] into this taxonomy. It turns out that multimedia applications can be found (more or less, depending on the interpretation) in *all* classes of application software.

Faceted Taxonomies As a classification purely based on the purpose is not always sufficient some literature proposes detailed faceted taxonomies. An often cited taxonomy can be found in [Heller et al.01]. They propose three dimensions:

- *Media Type* with the values *Text*, *Sound*, *Graphics*, *Motion*, and *Multimedia*,
- *Media Expression* with the values *Elaboration*, *Representation*, and *Abstraction*. This refers to the degree of abstraction, i.e. whether content is for instance represented by a lifelike photo or by an icon.
- *Context*, which does not contain discrete values but a collection of categories for qualitative questions that can be asked about a software product and categorize it in a non-quantitative way. The six proposed categories concern the *audience*, *discipline*, *interactivity*, *quality*, *usefulness*, and *aesthetics* of a multimedia product.

In contrast to [Heller et al.01] which focus more towards aesthetics, the taxonomy in [Hannington and Reed02] assumes the viewpoint of development. Thus, it is the most important for this thesis. It provides a large number of facets to describe all aspects which might influence the development of a multimedia application. Altogether they propose 21 facets together with possible values. The facets span from general facets used in other taxonomies, like the application domain, over facets like the delivery target platform (online, offline, etc.), navigation (linear, hierarchical, etc.), security requirements (access levels, authorization, etc.), up to very detailed properties like used media formats (JPEG, GIF, etc.), user interface widgets (button, checkbox, etc.) or authoring tools used for development (Flash, Director, etc.). A listing taken from [Hannington and Reed02] showing all facets and possible values is attached in appendix A.

The facets by [Hannington and Reed02] from above provide a very detailed understanding on properties in multimedia application development. Taking them all into account would be useful for instance to compare two concrete existing multimedia products. However, its level of detail is much too high to achieve a compact overview on the whole spectrum of multimedia applications. Thus, as intended by the authors in [Hannington and Reed02], it is possible to customize the taxonomy by selecting only those factes which are most important for our purpose. The next section elaborates such a taxonomy for this thesis.

2.4.2 A Classification for this Thesis

For this thesis it seems useful to aim for a taxonomy which covers the whole spectrum of multimedia applications from the viewpoint of development but is still manageable. A central observation in the foregoing sections was that multimedia application development is strongly affected by two different fields, media design and software programming. Many tasks, developer roles, tools, implementation technologies, etc., depend on the expression of these two aspects for a given application. Thus, it is useful to use this central theme also as main idea for a compact taxonomy.

A reasonable proposal by [Kraiker07] is to use the two facets *Domain* and *Interactivtiy* for this purpose. When looking at [Hannington and Reed02] (appendix A), it turns out that indeed most other facets are less important for our purpose: The taxonomy for multimedia applications here should base on the conceptual properties of applications themselves, i.e. their requirements on a certain level of abstraction. However, most facets in [Hannington and Reed02] actually describe either:

- the concrete solution chosen by the developers (*solution space*, *navigation*, *interface, programming*),
- the concrete development itself (*operations*, *design technique*, *authoring tools*, *skills*),

- or technical details (*state*, *duration*, *size*, *format*).

Thus, it is reasonable to omit them here. Delivery Platform and Security both indeed describe application requirements but they are considered here as too specific to really influence the development in general.

The remaining two facets from [Hannington and Reed02] are Media and Origin. Media means the media type used in the application. This facet indeed influences the development process as e.g 3D graphics requires very different experts, tools, and concepts than e.g. a mainly text-based application. Origin refers in [Hannington and Reed02] to the origin of a media artifact with the values: *Acquired*, *Repurposed*, and *Created*. This also influences the development as it makes a difference whether media artifacts must be created in a possibly very complex design process or whether they are just taken from an external source and must be integrated (only).

For sake of simplicity it is possible here to omit the value "Repurposed": Either it requires some effort to adapt the media then it is similar "Created". Otherwise it is close to "Acquired". Thus, for our purpose the values are substituted by two more generic values: *Received* means that a media object must not be designed within the development process but is taken from an external source, like another company, an existing application, or by the user at runtime (e.g. in a video editing application the videos are provided by the user herself). *Designed* means that the media object is designed as part of the development process.

However, there is an additional value which is useful in our context which can be called *Generated*. An example is Google Maps which contains complex graphics. This graphics is neither designed by graphic designer nor taken from an external source – it is generated directly from geographical data instead. This makes a significant difference, as it requires no media design but complex programming instead.

In summary, we the following facets are used, in order of their importance:

Domain Gives a basic idea on the application's purpose and its required domain concepts. Possible values: *Business*, *Information*, *Communication*, *Entertainment*, *Education* (see sec. 2.4.1).

Interactivity Influences the degree of programming vs. authoring. Possible Values ([Hannington and Reed02, Aleem98, Heller et al.01]):

- *Passive:* The user has no control like in a movie.
- *Reactive:* Provides limited response for the user within a scripted sequence. For example the user can select between some predefined graphics.
- *Proactive:* Allows the user to play a major role in the design and construction of situations, typically by manipulating values. For instance, the user can initiate changes to the properties of a graphics, like color, shape, rotation, position, etc.
- *Directive:* Allows the user to control the content of the application (in addition to manipulate values). For instance, the user can create her own graphics.

Media Origin: Influences design vs. programming vs. integration only. Possible Values: *Received*, *Designed*, *Generated*.

Media Types: Influences kind of design/programming/integration. Possible Values: *Audio*, *Video*, *Graphics*, *2DAnimation*, *3DAnimation*, *Text*, *Image*.

Table 2.1 shows the spectrum of multimedia applications in terms of the classification. The table columns and rows represent the first two facets, 'Domain' and 'Interactivity'. In the Interactivity facet

Interactivity / Domain	*Business*	*Information*	*Communication*	*Edutainment*	*Education*
Directive	Authoring Tool R G		CSCW System R	City-building Game D G	Electronic Circuit Simulation D G
Proactive	Car Configurator D	Navigation System G	Video Conference R	Car Racing Game D	Flight Simulator D G
Reactive	Online Shop D	Encyclopedia R		Media Player R	Medical Course D

Media Origin:		
	R	*Received*
	D	*Designed*
	G	*Generated*

Table 2.1: Overview on the spectrum of multimedia applications

the values 'Passive' has been omitted for simplicity as this work is on interactive applications. The values of the third dimension, 'Media Origin', are indicated inside the table cells by the letters 'R' for 'Received', 'D' for 'Designed', and 'G' for 'Generated'. The fourth dimension, 'Media Type', is omitted for simplicity, as it has the lowest influence here.

The table contains an example for each class defined by the primary two facets. Of course, classifying the examples is to some extent subjective. In particular, the media origin often depends on the detailed functionality. Often, an application combines two or three types of media origin. For instance, the media in authoring tool are mainly provided by the user (e.g. a video in Flash) or generated (e.g. graphics created in Flash). But an authoring tool could additionally provide predefined media which then might be designed. Similarly, the media origin for other applications depends on the detailed example. Nevertheless, it is not that much important here which kind of application uses which media origin but rather that all kinds of media origin frequently occur and are often also combined within the same application.

One can see in the table that applications using "generated" media are mostly proactive or directive which seems quite logical. For communication applications no "reactive" example was found. This makes probably sense as for communication software the content must by definition be influenced by the user. In turn, there is no "directive" examples for information software, as such examples are usually classified as communication software. However, again the classification is quite subjective.

In general, it is not the intention here to provide a new contribution in terms of the preciseness of a taxonomy but rather to provide a reasonable and meaningful overview. It will be used later in section 8.3.3 to evaluate the solution proposed in this thesis.

2.5 Conclusions for this Thesis

This section specifies interactive multimedia applications as understood in this thesis by collecting the conclusions from the foregoing sections.

Depending the kind of application, the previous sections elaborated the following properties:

- A multimedia application basically is any application with a multimedia user interface (sections 2.1.3 and 2.4).
- A multimedia application as understood here directly serves the user and is not just infrastructure software (sec. 2.2).
- The term “multimedia application” does not necessarily mean a multimodal application (sec. 2.1.2).

The term multimedia still needs some more discussion here as this thesis assumes the viewpoint of development. From that point of view, an interactive multimedia application usually involves media design and user interface design (sec. 2.1.3). It is implemented with specific implementation support for multimedia (sec. 2.3). However, when looking at advanced multimedia applications, media objects are not always designed as part of a media design process but can also be generated at runtime (sec. 2.4).

In contrast some to existing definitions for “multimedia” (sec. 2.1.1), it is not so important here, whether and how many different media types are integrated (e.g. one continuous and one discrete) but much more the integrative aspect itself (sec 2.2). Moreover, in contrast to some document-oriented points of view, integration is not restricted to spatio-temporal behavior but rather concerns also media objects and application logic.

This leads to the following definition:

> An interactive multimedia application is any application with a multimedia user interface which means that
>
> - it has a non-standard user interface but uses (to a relevant amount) media objects like audio, video, image, graphics, 2D- and 3D-animations,
> - which are tightly coupled to the application logic,
> - and are possibly designed in a media design process.

Thereby, the coupling between media objects and application logic is may include:

- Creation, deletion, modification of media objects by application code at runtime,
- Creation of events from media objects propagated to the application logic (i.e. usage of media objects for interactivity).

Chapter 3

Problem Statement and Proposed Solution

This chapter provides an overview over the approach presented in this thesis. The first section discusses the problem to be addressed in this thesis based on literature in particular existing studies on industrial practice. It turns out that multimedia application development still lacks of adequate development concepts as known in software engineering. The section also outlines the specific challenges which clearly distinguish multimedia application development from other areas. On that base, the second section discusses the solution space and identifies a model-driven development approach as the most promising solution. The third section briefly introduces the main concepts of model-driven development necessary to understand the subsequent chapters. Finally, the fourth section briefly illustrates the main ideas of the solution by an example application which is also used as running example in the further parts of the thesis.

3.1 Current Problems in Multimedia Application Development

Multimedia has still not found its way into common applications and many multimedia applications are still far away from its full power. One reason for this is certainly the still very high costs and efforts required for creation of sophisticated and fully integrated multimedia user interfaces [Bulterman and Hardman05]. Research has mainly focused on multimedia services and system technologies like network and databases – which are certainly necessary foundations – but too sparsely addressed sufficient support for more advanced multimedia application development [Engels and Sauer02]. An earlier article from industry illustrates the situation: In “The Killing Fields” [Kozel96] Kathy Kozel, a popular multimedia developer and evangelist for the authoring tool *Director* heavily complains about the situation in multimedia development where the total absence of systematic methods and processes leads to inscrutable projects and excessive long working times.

This is to some extent approved by the work by Kerstin Osswald [Osswald03] who provided one of the most in-depth studies focusing on companies developing interactive multimedia applications in Germany. From 3000 candidate companies 30 were selected based on rankings to find those with either the highest business volume in this area or which stand out for their creative innovations. Finally, 22 companies agreed to take part in semi-structured interviews which took about 2 hours. The study examines the development process and its specific tasks and artifacts in the multimedia companies. As it was found that no adequate multimedia development process exists, Osswald integrates the identified tasks and artifacts with an iterative development process from conventional software engineering like the *Rational Unified Process* [Jacobson et al.99].

As it turned out in the study, companies are basically willing to use more systematic development concepts but in practice use, if any, mainly very basic or legacy concepts – probably because of the lack of concepts well adjusted for multimedia development. For example, more than 80% of the respondents stated to apply the waterfall model. On the other hand, project size and complexity are considerably increasing which suggests that optimized and systematic methods will be even more important in the future. The average working time per week was found to be still between 50 and 60 hours. As Osswald summarizes, companies themselves often call their process as ad-hoc implementation. With the increasing size of today's projects (project budgets exceeding one million Euros are no longer rare) and their increasing complexity causes that developers get totally lost within the bunch of development tasks (see preface in [Osswald03]).

Since the upcoming of larger multimedia applications in the 90's research literature frequently admonishes the missing systematic approach in multimedia development, e.g. [Dospisil and Polgar94], [Rahardja95], [Arndt99], [Hirakawa99], [Rout and Sherwood99], [Gonzalez00], [Aedo and Díaz01], [Engels and Sauer02]. A comprehensive summary is e.g. provided in [Balzert and Weidauer98]: One of the main problems is the missing support for pre-implementation phases which leads to an ad-hoc implementation. This leads to unstructured results which are very hard to understand, maintain, and extend. There are also no sufficient concepts which help to build a bridge between the initial requirements and the implementation. Furthermore, the results of requirements are quite informal, either textual or in form of informal diagrams like storyboards, whereby they can easily become ambiguous, inconsistent, on different levels of abstraction, difficult to process and can not be used for (semi-)automatic transitions. In all, according to [Balzert and Weidauer98] the situation in multimedia development can be compared to the state-of-the-art in software engineering in the early 70s. [Engels and Sauer02] states:

> From a software engineering perspective, the problem with the current state of multimedia application development is not only the absence of sophisticated, yet practical multimedia software development process models, but also the lack of usable (visual) notations to enable an integrated specification of the system on different levels of abstraction and from different perspectives.

and comes to the conclusion that

> The implement-and-test paradigm used during multimedia authoring resembles the state of software development before leading to the software crisis of the 1980s.

Some authors move a step further and introduce the term of *hypermedia crisis*[1]:

> Hypermedia development is currently at the stage software development was at thirty years ago. Most hypermedia applications are developed using an ad hoc approach. There is little understanding of development methodologies, measurement, and evaluation techniques, development processes, application quality, and project management. [...] We are potentially about to suffer a hypermedia crisis. [Lowe and Hall99]

Such statements are heavily criticized by Lang, who has already been co-author of one of the largest studies so far, described in [Barry and Lang01], which results were more ambiguous. He performed a new study together with Fitzgerald [Lang and Fitzgerald05] to find out 1) the extent

[1]The term hypermedia here refers to web and multimedia applications, see section 2.1.4. Additional information about the companies examined here is also given at the end of this section.

Hybrid, customized, or proprietary in-house method or approach (not further specified)	23%
Traditional “legacy” software development methods and approaches or variants thereof, such as Structured Systems Analysis and Design Methodology (SSADM), Yourdon, Jackson Structured Programming (JSP), System Development Life Cycle, or Waterfall	22%
Rapid or agile development methods and approaches, such as Rapid Application Development or Extreme Programming	18%
Approaches that focus on the use of tools and development environments, such as PHP, Java, etc.	15%
Object-oriented development methods and approaches, such as Rational Unified Process or object-oriented analysis and design	11%
Approaches that focus on the use of techniques, such as Storyboards, Flowcharts, Wireframes, or UML	8%
No method used or development approach is ad-hoc	8%
Specialized nonproprietary methods for Web and hypermedia systems development, such as Fusebox, Web Site Design Method (WSDM), or OOHDM (see section 4.2)	5%

Table 3.1: Applied development methods in web and multimedia industry [Lang and Fitzgerald05]

to which the problems characterizing the alleged “hypermedia systems development crisis” actually exist in practice, and 2) which, if any, mechanisms developers use to guide and control hypermedia systems development. Therefore they sent questionnaires to web and multimedia development companies in Ireland which were responded by 167 companies. The findings to question 1) were that according to the questionnaires there is no evidence for a “crisis”. However, one problem of the survey, also described by Lang, is that there is no evidence that the answers of the respondents are too optimistic. (Most participants estimated different development aspects with “minor problems” or “moderate problems” instead of “no problems” or “major problems”).

The questions for the study’s second issue were open ended and provided very ambiguous results. Table 3.1 shows the answers structured into categories like in [Lang and Fitzgerald05].

When asked about their general opinion on structured development methods, 94% agreed that planning is essential, and 80% agreed that plans and working methods should be clearly documented. 69 % agreed that ad-hoc methods generally result in poor systems. The suggestion that “documented working methods are pointless” was firmly rejected by 79%.

Lang and Fitzgerald conclude that most companies do already use software engineering concepts and that major problems in web and multimedia development do not exist. According to their interpretation in this article, there is neither a hypermedia crisis nor are academic hypermedia approaches accepted in industry. They summarize that the academic view of this area is far away from industrial practice.

One can object that the survey findings about approaches applied in industry are difficult to interpret. For example, stating a tool or programming language as applied “development approach” can lead to the assumption that the company does not really apply an approach besides ad-hoc programming. “In-house approach” is also difficult to interpret as well. In particular, the usage of “legacy” software engineering methods, not adapted to web and multimedia application area, suggests that, even if it might work, there is at least much space for optimization.

These objections are approved by another more in-depth follow-up study by Lang and Fitzgerald [Lang and Fitzgerald06] where they examined more in detail the applied approaches and methods. They find out that “old” software engineering concepts are mainly used

> [...] even though a substantial cohort, including some who actually use such methods, consider them somewhat impractical. [Lang and Fitzgerald06]

On the other hand

> The level of formality of development processes was found to be negatively correlated to the level of severity of problems raised by a number of selected development issues, suggesting that formalized processes and procedures can help reduce the incidence of such issues. [Lang and Fitzgerald06]

In summary, the truth certainly lies somewhere in between. The situation in multimedia and web development might be not as bad as sometimes described. Clearly, many companies are successful since many years and do also improve their development processes over time. On the other hand, well-structured and more formal concepts are clearly seen to be valuable and existing methods are definitely not optimal. Thus, despite of the personal opinion on the situation in industry ("crisis" or just a need for optimization), there is clearly space for optimization by providing more suitable methods for web and multimedia development.

As Lang and Fitzgerald underline, new approaches must be practically usable [Lang and Fitzgerald06]. According to them this requires easy usage, good documentation, and a good reputation in industry. While the last point is often difficult to achieve for academics and also professional tool support can only be provided by a respective tool vendor company, at least ease of use and applicability clearly must be ensured as much as possible by academic proposals.

The studies of Lang and Fitzgerald includes web and multimedia companies. In the last study the ratio of multimedia companies was about 14%[2]. While conventional standard web applications are very well understood today and already comprehensive modeling approaches exist (see section 4.2), the conclusions above are even more important in multimedia development, as this area comprises additional complexity. The following two sections discuss the two essential challenges specific for multimedia application development.

3.1.1 Interdisciplinary Roles

The development of interactive multimedia applications is characterized by the integration of knowledge, tools, and experts from different areas. For example [Tannenbaum98] describes that multimedia production includes among others: acting, animation, arts and graphics, audio recording and editing, computer programming, copyright law, directing, engineering, graphic design, human factors analysis, instructional design, legal analysis, marketing and packaging, morphing, motion videography, networking, producing, script writing, software design, stage and set design, still imaging, storytelling, systems analysis, technical writing, text design, text formatting, text layout, user interface design, video editing, and virtual reality.

While some aspects are obviously part of conventional application development as well, many of them are specific for multimedia objects. Some authors ([Mühlhäuser and Gecsei96][Gallagher and Webb97]) initially distinguish between two kinds of categories: *software design* and *media design*. But today it is commonly accepted that *user interface design* is an additional own aspect of interactive application development [Dix et al.03, Shneiderman and Plaisant04]. It is not necessarily part of software and media design and should thus be explicitly considered as an own category ([Gonzalez00, Wolff05]), in particular as the user interface is eminently important in multimedia domain.

Thus, in this thesis the development tasks are subsumed into three categories of design:

[2]companies stating "E-Learning/CBT" or "multimedia" as their primary business

Software Design refers to development tasks required to conceive and produce the conventional software part of the application, i.e. the application logic, according to standard software engineering principles like in [Sommerville06, Balzert98]

User Interface Design refers to development tasks required to conceive and produce the user interface of the application according to principles of human-computer interaction like described in [Dix et al.03, Shneiderman and Plaisant04].

Media Design refers to media-specific development tasks required to conceive and produce the media objects and their integration.

Of course, the detailed tasks of each category depend on the company and the concrete project. In particular, media design includes very different tasks depending on the specific media types, e.g. video production or 3D graphics design. A fourth category of tasks, usually part of any kind of project, is *project management* which includes tasks required for the coordination between the different developer groups (see more detailed developer roles provided by Osswald [Osswald03]). Some authors, like Gonzalez [Gonzalez00], mention additional tasks like systems design which includes the design of hardware components. Such low-level tasks are not considered in this work as the focus here lies on the abstraction levels required for application development.

The coordination of the different developer groups and their artifacts integration of their different results is clearly a requirement specific for interactive multimedia applications. It is often claimed as one of the main challenges which have to be addressed by multimedia application development approaches (e.g. [Morris and Finkelstein96, Hirakawa99, Rout and Sherwood99, Hannington and Reed02]). Balzert [Balzert and Weidauer98] reports by own experience that the implementation of interactivity by different teams is tedious and often inconsistent. When multimedia user interfaces become more sophisticated, the dependencies between media objects, user interface, and application logic increases. For example, media objects can be used for additional user input, e.g. by clicking on an image or dragging and dropping an animation. In particular, media objects do not only act as monolithic objects but require a specific inner structure which can be connected to application logic. For example, the user can trigger functionality by selecting a specific part of a graphic, e.g. in a map. In turn, the application logic often must be able to access inner parts of a media object, for instance when some parts of an animation are moved according to application logic (example see figure 3.8). Synchronization between media objects may require knowledge about their inner structure as well, e.g. if some action on the user interface should be triggered while a specific scene within a video is displayed.

Such interrelations are a critical bottleneck within the application development and require careful coordination between the different developer groups. Analogous to conventional software development it is mandatory to specify the "interfaces" between the different artifacts which should become connected. "Interface" here means not necessarily an interface or component model on implementation level – in is already an significant help if there is a systematic for a fix agreement between the developer groups how their results must be composed so that they fit together. Development support for specifying these interfaces on an adequate level of abstraction can greatly increase efficiency in development and maintainability of the application.

In summary, a development method for multimedia applications should support the integration of media design, user interface design, and software design.

3.1.2 Authoring Tools

Usually multimedia applications are developed using authoring tools [Engels and Sauer02, Bulterman and Hardman05]. They are necessary for the creative design tasks in media design and user interface design. A well-known problem of authoring tools is the trade-off which has to be made: "On one hand, a tool with programming structure built-in may ease software maintainability, but it entails harder and longer learning curve for the end users, who are likely to be the main users of the tools. On the other hand, an authoring tool that is free of programming structure may be intuitive and easy to use, it is also susceptible to maintenance problems if the software is not properly designed." [Rahardja95].

Authoring tools like Flash, which is probably one of the most widespread and advanced professional authoring tools today, show that this situation has not changed today. On the one hand, Flash is very well established because of its support for creative design and it is used by many user interface and media designer for various tasks. On the other hand, structuring the application is still very difficult. Script snippets can be added directly to objects on the user interface, which allows quick and easy specification of functionality and event handling. With increasing number of user interface objects, the script snippets are scattered all over the application. The latest versions of Flash allow also object-oriented code in separate class files. However, by the historic evolution of Flash, object-oriented concepts can not always be used consistently. Often it remains unclear how to integrate object-oriented code with the different kinds of user interface objects in a structured way. Even structuring the application to some degree requires good knowledge of software engineering principles and very disciplined adherence to conventions in Flash. As already noticed by [Rahardja95] shifting the problem to the developer by at least enabling him to apply patterns and templates can reduce the problem but can not be the final solution. For other authoring tools like Director, where the integrated scripting language has not evolved much, the situation is even worse.

Another important problem of authoring tools (also described by [Balzert and Weidauer98]) is the lack of version and configuration management which is still a problem today even in professional tools like Flash and Director. The version management in Flash does only allow locking of files, but does not provide any other standard functionality like file comparison or file recovering. The situation is similar for Adobe Director. Use of external version management tools can not improve this situation significantly, because authoring tools usually use a proprietary binary file format.

Another problem to be mentioned is the large dependency on the company producing the authoring tool as changes of the implementation platform requires implementing the application completely new.

In summary, development support for multimedia applications must support authoring tools as they are very well established for the creative design tasks, but it should provide support for better structuring and maintenance of the applications.

3.2 Analyzing the Spectrum of Possible Solutions

This section discusses possible solutions and elaborates a solution approach. As discussed in section 3.1 there is absolutely a need for a better integration of software engineering principles into multimedia development. Possible solutions must fit to the specific properties and challenges of multimedia development, which are in particular their highly interdisciplinary character and the integration of authoring tools.

Formal Methods A classical branch in software engineering are formal methods based on mathematical logic like the general purpose language Z [Potter et al.96] or algebraic specification languages

[Wirsing90, Broy et al.93]. They allow a very well-structured development process with a high degree of automation and validation of the developed system. Some formal approaches for multimedia development have already been proposed [Blair et al.97, Sampaio et al.97]. The disadvantage of formal methods is the high effort required for learning as well as for applying the method, which results as trade-off from the feasibility of automatic validation. This will pay off mainly for safety critical systems which are only a very small part of multimedia systems. Furthermore, the focus of a multimedia application often lies on weak, non-functional requirements regarding aesthetics, realistic effects, efficient information transfer, and usability. They are difficult to measure and formalize – if at all, then only with a very high effort. The already existing methods in multimedia development, like prototyping and user tests, are usually more adequate for validating such requirements. Finally, like discussed in section 3.1, approaches for multimedia development must be lightweight and easy to use. Formal methods will hardly be accepted in this area.

Agile methods Agile methods [Poppendieck and Poppendieck03, Eckstein04] can be regarded as counterpart of formal methods in software engineering. By their lightweight character and their close relationship to prototyping and user tests, they are obviously candidates to be applied in multimedia development. As these approaches do not require any high-level design artifacts like models (in terms of a pre-implementation design like with UML), choosing an agile approach would cause that the problem of missing models becomes obsolete. Also, agile approaches fit optimally to high occurrence of requirement changes in multimedia development. However, there are also some serious difficulties when applying such approaches to multimedia application development.

In the annually course "Multimedia Programmierung" (multimedia programming) at the University of Munich, supervised by Prof. Hußmann and assisted by the author of this work, students have to develop a Flash application over three months in teams of 5 to 7 people. In the 2004 edition we conducted an experiment and forced the students to use an agile development process. The concrete agile approach to be applied was *Extreme Programming* (*XP*, [Beck and Andres04, Stephens and Rosenberg03]). In its original version, XP uses twelve interwoven practices which have to be adhered strictly in order to compensate the missing formality and ensure the quality of process and results. While some of these work well very in multimedia projects, others are hard to apply.

An important practice is testing. XP requires writing automated tests before implementing a feature (unit tests and functional tests). At any time of the project the unit tests have to run for 100%, and all of them have to be run every time new code is added or code is changed. This is also interwoven with the practice of refactoring which requires that when adding new code the overall code is (if necessary) re-structured to have at any time a well-structured overall system which is as simple as possible and does not contain duplicate code. These two XP practices are essential to compensate the missing design specification. However, in multimedia programming automated tests are only possible to a limited extent. As explained above, the requirements are often weak and can not be measured directly. Thus, refactoring becomes more difficult as well. In addition, refactoring is much more difficult when using multimedia authoring tools, as (see above) structuring the application in general is difficult and requires specific expert knowledge and additional conventions. Also, other issues of multimedia, like missing version management systems, are problematic as well. Thus, it is doubtful whether the code-centric practices of XP can really ensure a good structure of the overall application.

Another problem is that the production of complex media objects can require a relatively long period of time. This constricts the XP ideas of very small, manageable increments and iterations. In fact, multimedia developers often have to work in parallel for longer periods which contradicts the XP ideas and rather profits from preceding specifications. In general, XP does not provide specific support for the problem of integrating the interdisciplinary experts in multimedia development.

Altogether, XP has some striking commonalities with multimedia development but in fact the practices are too much code-centric to be applied directly to multimedia-specific conditions and directly improve the existing situation.

Approaches based on Visual Modeling Languages Another conventional approach in software engineering considered to be in the middle between formal methods and extreme programming are iterative processes including a software design phase where the system is specified in a semi-formal way e.g. using visual modeling languages. A typical example is the *Rational Unified Process* (*RUP*, [Jacobson et al.99]) combined with the Unified Modeling Language (UML, [Rupp et al.07, Hitz et al.05] see also section 3.4). The intensive usage of modeling languages has gained increasing popularity in the last decade and is sometimes referred to as de-facto standard for object-oriented software development.

An adoption of RUP specific for multimedia applications, called the *SMART* process (see section 3.1) has been introduced by [Osswald03] based on her comprehensive study. However, the main problem – as indicated by research literature as well as by the critical study of Lang and Fitzgerald – lies in the missing modeling concepts for multimedia applications. UML is not sufficient because it does neither support concrete concepts for modeling the user interface (see sec. 4.1) nor for modeling media. Thus, research literature claims to introduce customized modeling concepts and some first approaches have already been proposed (see section 4.3).

Models provide – at least partially[3] – the advantages of formal methods like ambiguousness of specifications and the possibility of their automatic processing e.g. for validation or code generation purposes. On the other hand they are more accessible for developers by their representation as visual diagrams. (For example UML class diagrams are frequently used also by developers without specific software engineering knowledge.) As models provide a relatively clear and abstract overview they are in general very well suited for integration and communication between developers. In multimedia projects models can be used to act as a kind of contract which specifies the interfaces for the different artifacts of the different developer groups.

An important concept is the ability to automatically generate code or code-skeletons from models. While usage of models in general is often referred to as model-based development, the so-called model-driven development goes a step further and proposes to use various models throughout the whole development process up to the final implementation which is completely generated from the models. The upcoming field of Model Driven Engineering proposes tools and standards for model-driven development which provide additional advantages for the development. Besides, advanced tool support these include e.g. well-defined concepts for maintaining, customizing, and combination of modeling approaches and code generation. An overview on model-driven engineering is provided in the next section 3.4.

Automatic code generation from the models can not only improve significantly the efficiency in the development but also can increase the quality as the generated code adheres consistently to the specification. Furthermore, specific expert knowledge about the implementation platform can be put into the code generator so that it is automatically available for all future projects. This is in particular very helpful for multimedia projects where implementation platforms, like authoring tools, require very advanced knowledge about structuring the code.

Often, modeling languages provide platform-independent models. From those, models for different specific platforms can be derived which are finally transformed into platform-specific code. This

[3]Modeling languages are often referred as *semi-formal* as often their semantics is not defined formally, see sec. 3.4

means that the general concepts for an application have to be specified only once and can be reused for an arbitrary number of implementation platforms. For multimedia applications this will be even more important in the future as visions like ubiquitous computing [Weiser99] suppose that user interfaces run on a large spectrum of different devices and platforms. For this reason model-based approaches are considered as very promising for development of future user interfaces in general [Myers et al.00]. Platform-independent models also help for the problem described by Balzert that multimedia application developers are highly dependent from authoring tools vendors and must completely re-develop applications if support by a specific authoring tool becomes insufficient.

In summary, the discussion shows that among conventional approaches from the software engineering area a model-driven approach advantages clearly seems to prevail. Certainly, other concepts, like Agile Development approaches, might be useful for multimedia development as well, if adequate adaptations could be found. However, the study of Lang and Fitzgerald [Lang and Fitzgerald05] suggests that the general acceptance for more formal approaches is not as low as expected. A promising future direction can be Agile modeling Processes [Rumpe04, Rumpe06] which might achieve a combination of both advantages for multimedia development.

The following section discusses the cornerstones of an adequate model-driven development approach customized for multimedia applications.

3.3 Proposed Solution: A Model-Driven Approach

This section shows how a model-driven development approach can meet the challenges of multimedia applications development identified in section 3.1. The approach provides a visual modeling language as base for the communication between the different developers groups. As described in chapter 4, various modeling approaches already exist in the fields of media design, user interface design, and software design. This work is based on these approaches, reuses established concepts where possible, and integrates the concepts from all three areas. In that way, the models allow the media designers, user interface designers, and software designers to specify the interfaces and interrelationships between the different aspects of the application.

The proposed modeling language, called *Multimedia Modeling Language* (*MML*), is platform-independent and object-oriented. Furthermore, it enables code generation for various platforms. Thereby it strongly adheres to the concepts of model-driven engineering, enabling relatively easy adaptations or combination with more specific existing modeling concepts e.g. for context-sensitive user interfaces (see section 4.1).

A core idea of the approach is the integration of authoring tools. From the models it is possible to generate code skeletons customized for the authoring tool which can then be directly loaded and processed within the authoring tool. The overall structure and the interfaces and relationships between user interface elements, media objects and application logic can be specified in the model and be generated automatically in a consistent way. The concrete creative design and layout and the implementation of the detailed behavior is not specified within the model – instead, just placeholders are generated which have then to be filled out in the authoring tool using its established features for creative design and the platform-specific code constructs for the detailed behavior. So, models and authoring tools are both used for what they are best at: models for structuring, communicating and specifying the overall structure and the interfaces between different parts, authoring tools for creative design. In this way the advantages of models and authoring tools are combined. As mentioned above, the proposed code generators can also contain specific expert knowledge about the implementation platform.

The provided modeling language is as lightweight as possible. The concept to restrict modeling on the overall application structure, as described above, contributes to this goal very well. Furthermore, also the detailed behavior is not specified in the models, as this is much more efficiently realized within the authoring tools or programming environment. Anyway, an efficient implementation usually requires platform-specific constructs which would be very tedious to specify within the models. In particular for multimedia applications the code often causes effects on the user interface. Concrete algorithms and parameter values to achieve the desired effects can often not be specified just on a theoretical base but have to be found out by try and error and by examination of the running system. Thus, such details are omitted in the models. The models are reduced as much as possible to the essential elements to keep them as easy to use as possible. In that way the approach adheres to the requirements of Lang and Fitzgerald [Lang and Fitzgerald05] and [Voss et al.99] (see above) which postulate that solutions must be lightweight and easy to use.

In summary, the approach introduced in this thesis provides the following solutions to address the specific problems and challenges in multimedia application development:

1. Integration of media design, user interface design, and software design into a single, consistent modeling approach.

2. A level of abstraction which enables code generation but ensures a lightweight approach.

3. Advanced integration of authoring tools.

The following section first introduces the current state-of-the-art in model driven development and the respective area of model driven engineering. Afterwards an illustrating example scenario for the proposed approach is shown. Both sections build the background for the more detailed descriptions in the following chapters of this work.

3.4 A Short Introduction into Model-Driven Engineering

This section briefly introduces terms like models, model-driven development, and related terms which are important later in this thesis. It briefly summarizes the basics and the current state-of-the-art in this area.

3.4.1 Models, Metamodels, and Modeling Languages

Models are an essential concept in all scientific areas.

> Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality. [Rothenberg89] (according to [Bézivin05])

Examples are a human blood circulatory model in medicine or a globe for geographical information. Both represent only *certain aspects* of the system. For example, it is not possible to use a globe for measuring the temperature on a certain point on the earth. In addition, models show given aspects

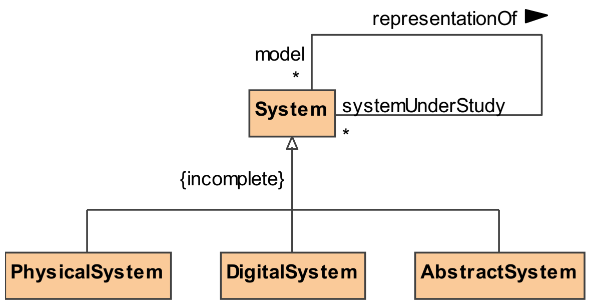


Figure 3.1: *RepresentationOf* relationship according to [Favre04c]

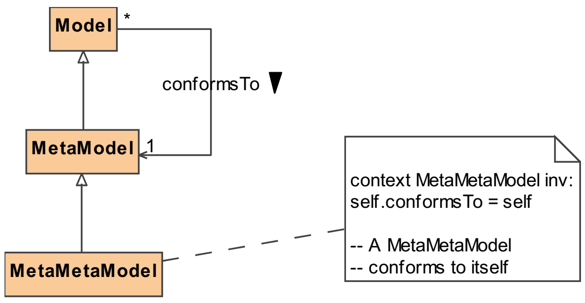


Figure 3.2: *ConformsTo* relationship according to [Kurtev et al.06]

of a system at different levels of *abstraction*. For example, geographical maps exist in many different scales for different purposes (see [Bézivin05]).

A short and common definition of models is provided in [Seidewitz03]:

> A model is a set of statements about some system under study.

A model itself can also be interpreted as system. Thus, a model can be represented by another model, e.g. on another level of abstraction. Analogously, a map can also be seen as a representation of another more detailed map of the same territory. Figure 3.1 shows these relationships: A system can be a model of another system. Kinds of systems include physical systems, digital systems, and abstract systems. The relationship between a model and the system it represents (system under study) is called *representationOf*.

Each geographical map requires – at least implicitly – a legend, to interpret the elements in the map. This concept can be generalized and applied to models which must also conform to a specified modeling language. According to the spirit in the modeling community, modeling languages are defined by models themselves, which are called *metamodels*. As shown in figure 3.2, each model *conforms to* a metamodel which is a model itself. Thus, a metamodel has to be conform to a metamodel, too, which is thus called *meta-metamodel*. The existence of a common meta-metamodel enables to compare, merge, transform, etc. between different metamodels and the corresponding models (see e.g. [Fabro et al.06] for advanced operations on metamodels). To avoid unlimited number of meta-steps, the meta-metamodel is its own reference model, i.e. it conforms to itself.

A more formal definition of these relationships is given in [Kurtev et al.06]:

- A directed multigraph $G = (N_G, E_G, \Gamma_G)$ consists of a set of nodes N_G, a set of edges E_G and a mapping function $\Gamma_G : E_G \to N_G \times N_G$
- *Model* is then defined as a triple (G, ω, μ) where

- $G = (N_G, E_G, \Gamma_G)$ is a directed multigraph,
- ω is itself a model (called the reference model of M) associated to a graph $G_\omega = (N_\omega, E_\omega, \Gamma_\omega)$
- $\mu : N_G \bigcup N_\omega$ is a function associating elements (nodes and edges) of G to nodes of of G to nodes of G_ω

The *conform to* relationship then corresponds to the relation between a model and its reference model. A meta-metamodel A metametamodel is a model that is its own reference model (i.e. it conforms to itself). A metamodel is a model such that its reference model is a meta-metamodel. A (terminal) model is a model such that its reference model is a metamodel [Kurtev et al.06].

An advantage of visual modeling languages is that they provide a visual notation in terms of diagrams. Often a modeling language provides different kinds of diagrams providing different *views* onto the model. A visual representation can provide a higher degree of usability as it provides much more possibilities to encode information and can often be percepted, understood, explored, etc. much more efficiently by humans than a purely textual notation [Tufte01, Ware04]. Certainly, as mentioned by Green [Green00], a visual notation alone will not improve usability. Instead, the notation has to support the given developer tasks as good as possible, which can be analyzed e.g. using the cognitive dimensions framework by Green [Green00].

Software systems usually should reflect information and processes of the real world. In software development, models are thus used to specify or prescribe the software system to be developed, but also to describe the system under study (e.g. a business process which should be supported by the software). In software engineering, many different kinds of models are used (see e.g. [Burmester et al.05] for an overview). A common distinction is made between *General Purpose Languages* (*GPL*) which aim to support many different application domains and *Domain Specific Languages* (*DSLs*) like shown at [Metb] which aim to provide optimized support for a delimited set of tasks, e.g. modeling a very specific kind of application, maybe even for a specific company.

The most important example of a visual modeling language is the *Unified Modeling Language* (*UML*, [Rupp et al.07, Hitz et al.05]), a general purpose language which integrates different types of diagrams originating from different areas in software engineering. It has been defined by the *Object Management Group* [OMGb], an industrial consortium, and has today become an industrial standard. UML is usually used in common software engineering methods like object-oriented analysis and design [Booch et al.07]. However, UML is independent from the development process and the concrete usage of UML models is explicitly not part of the UML specification. The concepts described above apply also to the UML: The UML is described by a metamodel [Obj07c, Obj07d] where UML models should conform to.

It is important to notice that a metamodel specifies only the abstract syntax of a modeling language. However, like any other (textual) language in computer science, a modeling language can be formally specified by defining its syntax and semantics. A discussion is provided in [Harel and Rumpe00]. For modeling languages like UML, the metamodel is supplemented with *well-formedness rules* – using the *Object Constraint Language* (*OCL*) [Obj06b, Warmer and Kleppe03] specified by the OMG as well – which further constrain the abstract syntax. The semantics and the concrete syntax – i.e. the graphical or visual notation – are defined only informally by natural language (although there are several initiatives to achieve a formal definition for UML, like [Broy et al.05, Reggio et al.01]). Examples for formally defined visual modeling languages are Petri Nets [Murata and Murata89] or State Machines [Harel and Naamad96]. However, in model-driven development the (execution) semantics of modeling languages is indirectly defined by transformations which map a modeling language onto a formally specified language, like e.g. Java code.

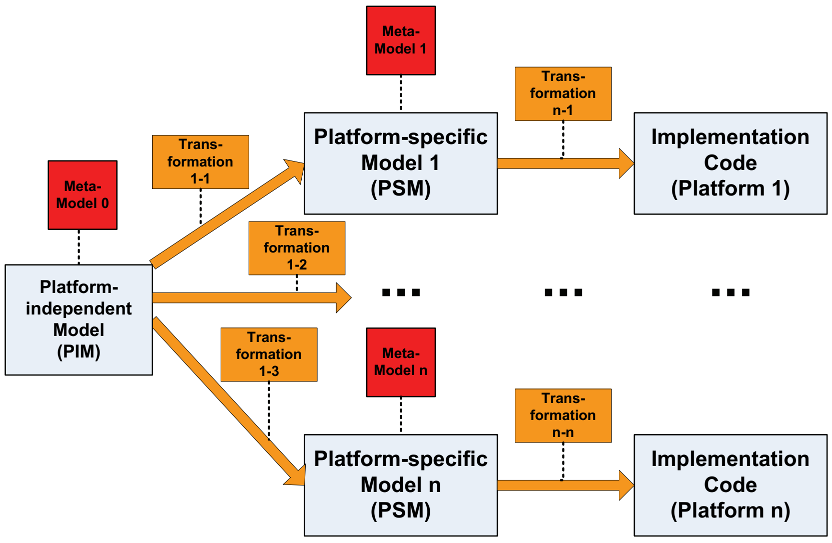

Meta-Model 1
Trans-formation n-1
Platform-specific Model 1 (PSM)
Implementation Code (Platform 1)
Meta-Model 0
Trans-formation 1-1
Trans-formation 1-2
Platform-independent Model (PIM)
...
...
...
Trans-formation 1-3
Meta-Model n
Trans-formation n-n
Platform-specific Model n (PSM)
Implementation Code (Platform n)

Figure 3.3: The basic idea of MDA

3.4.2 Model-Driven Development

In a *model-based* development process, the models are used to bridge the gap between the real world, i.e. the requirements on the software system, and its final implementation. This allows explicitly capturing and understanding the system and planning and structuring the implementation. *Model-driven development* (*MDD*, or model-driven software development *MDSD*) goes a step further and uses models as primary artifacts in the development process. More specifically, the primary artifacts in a model-driven development process are

1. models which conform to metamodels and

2. explicit transformations between them.

The OMG specifies the *Model Driven Architecture* (*MDA*, [Miller and (Eds.)03]), a concrete framework for the realization of model-driven development. The current working definition [OMGa] defines MDA according to [Obj04b]:

> MDA is an OMG initiative that proposes to define a set of non-proprietary standards that will specify interoperable technologies with which to realize model-driven development with automated transformations. Not all of these technologies will directly concern the transformations involved in MDA.
>
> MDA does not necessarily rely on the UML, but, as a specialized kind of MDD (Model Driven Development), MDA necessarily involves the use of model(s) in development, which entails that at least one modeling language must be used. Any modeling language used in MDA must be described in terms of the MOF language, to enable the metadata to be understood in a standard manner, which is a precondition for any ability to perform automated transformations.

Figure 3.3 shows the general idea of MDA: during the development process different models (conform to metamodels) are used, starting with abstract, platform-independent models (*PIM*) via transformations to an arbitrary number of different platform-specific models (*PSM*) which are finally transformed into the final implementations.

A model transformation is "the production of a set of target models from a set of source models, according to a transformation definition" [Sottet et al.07a]. They are executed by a *transformation engine*. [Czarnecki and Helsen06] provides a classification of transformation languages. Important classes are graph-transformation based approaches and hybrid approaches.

Graph-transformation-based approaches specify transformation rules in terms of typed, attributed labeled graphs [Rozenberg97]. A rule consists of a left-hand-side (LHS) and a right-hand-side (RHS). The LHS is matched in the source model and replaced by the corresponding RHS in place. The LHS consists of the matched pattern in terms of a graph, conditions, and some additional logic. Basically, the graph patterns can be expressed in the abstract or the concrete syntax of the source and the target language. Examples for Graph-transformation-based approaches are *AGG* [Taentzer00, AGG], *Atom3* [Vangheluwe et al.03, ATo], and *VIATRA* [Csertán et al.02, VIA].

The hybrid approaches combine several different paradigms, like declarative and imperative statements. Two important languages are classified by [Czarnecki and Helsen06] into this category: QVT and ATL. *QVT* (*Query/Views/Transformations*, [Obj07a]) is a standard defined by the OMG supporting queries on models, views on metamodels, and transformations of models. It includes two declarative components on different abstraction levels, *Core* and *Relations*, and a mapping between them (*RelationsToCoreTransformation*). Imperative logic is supported by the imperative component *Operational Mappings* and the component *Black Box* which allows the integration of complex algorithms written in any other language. As QVT standard definition is currently finalized, first implementations are still under development, e.g. as part of the *Eclipse model-to-model transformation* (*M2M*) project [Eclc]. A hybrid language with tool support is the *Atlas Transformation Language* (ATL, [Jouault and Kurtev05, AMM]). It also provides declarative and imperative statements and is relatively close to the QVT standard (see discussion in [Jouault and Kurtev06]). It has been integrated into the Eclipse M2M project as well. In particular, there is a library with ATL transformations at [ATLa]. More details on ATL can be found in section 7.

A transformation can be seen as a model as well. Thus, QVT has been defined in terms of MOF-compliant metamodels. As transformation languages and transformation engines are defined systematically they provide additional useful properties (in contrast to e.g. proprietary code generators). Basic properties a transformation should fulfill are [Kleppe et al.03]:

1. *Tunability*, i.e. the possibility to adapt the transformation by parameters for the transformation.
2. *Traceability*, i.e. the possibility to trace one element in the target model back to its causing element in the source model.
3. *Incremental consistency*, i.e. information added manually to the target model is protected and not overwritten if the transformation is executed again later (e.g. after the model has been updated).

Another property with low priority is *bidirectionality*, which means that the transformation can be applied not only from source to target but also backwards from target to source. However, this property is rarely applicable in practice and thus often ignored.

The OMG specifies various concepts for model-driven development, including *XML Metadata Interchange* (*XMI*, [Obj07b]) an XML-based exchange format for any kind of MOF-based model, the *Object Constraint Language* (*OCL*) [Obj06b], a formal language allowing the definition of constraints and conditions in models, and the *QVT* language (*Query/View/Transformation*, [Obj07a]) which allows specifying queries on models, views on metamodels, and model transformations. The OMG

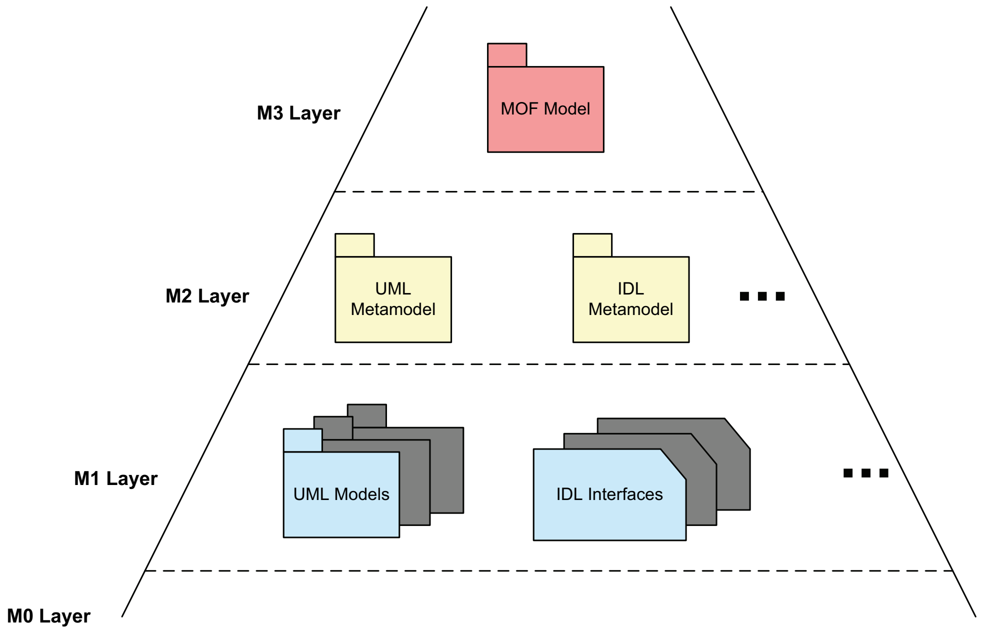


Figure 3.4: OMG Metadata Architecture according to [Obj05]

specifications base on the common meta-metamodel defined by the OMG, called *Meta Object Facility* (*MOF*, [Obj06a]). All OMG modeling languages like UML and others (e.g. CWM [Obj03]) are defined by a MOF-compliant metamodel. Traditionally the OMG illustrates the relationships between models, metamodels, and the MOF by the "four layer metadata architecture" shown in figure 3.4: Layer M3 contains the meta-metamodel MOF. Layer M2 contains the MOF-conformant metamodels defined by the OMG or third parties. Layer M1 contains the models which conform to the metamodels of M2. Finally, M0 contains the data of the real-world which is described by the models of M1.

In the past, the four-layer-architecture caused several problems and misunderstandings. In earlier versions the relationships between the model layers were called instantiation relationships, e.g. a class in an UML model was supposed to be an instantiation of the metaclass "Class". This kind of instantiation has not been distinguished from object-oriented instantiation as defined e.g. within UML models. An object in a UML diagram would then be instance of multiple elements: On the one hand an instance of a class in the UML model and on the other hand an instance of the metaclass "Object" in the UML metamodel. Several publications, e.g. by Atkinson and Kühne [Atkinson and Kühne01, Atkinson and Kühne03], discuss such problems and show that they lead to inconsistencies, concluding that the basic definitions have to be improved, e.g. by introducing the conformance relationship as defined above. It is also important to understand, that layer M0 contains only the real-world objects at runtime. The objects in a model (e.g. in a UML object diagram) are only snapshots of them and part of the model and thus reside in layer M1. Moreover, as also mentioned in the latest MOF specification, the number of model layers is not necessarily restricted to four layers.

In general, Bézivin [Bézivin05] shows that modeling concepts have to be clearly distinguished from object-oriented concepts. Moreover, the more general paradigm of models can replace object-orientation. While the existing paradigm of object-orientation "Everything is an Object" failed to enclose all important concepts from software engineering, the new paradigm "Everything is a Model" seems to be general enough for this purpose. This is illustrated by fig. 3.5 showing the application of MDE concepts on different *technical spaces* as described in [Kurtev et al.02, Kurtev et al.06]. As

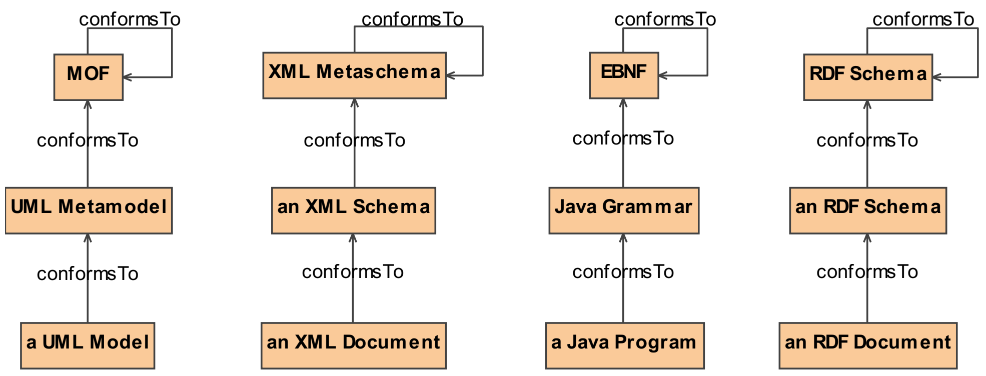


Figure 3.5: Technical Spaces according to [Kurtev et al.06]

in practice often one single technology is insufficient, the concepts from MDE can be used to bridge between them [Kurtev et al.06]. Another important related technology are ontologies ([Bechhofer et al.04, Bodoff et al.05] which can be integrated into the MDE concepts as well [Gasevic et al.07, Obj07e].

The area of *Model Driven Engineering* (*MDE*, see e.g. [Pla]) deals with the general concepts of MDD. The problems discussed in context of the MOF four layer metadata architecture show that the foundations of MDE requires further investigation and a more precise definition. Favre discusses the MDE theory ([Favre04b, Favre04a, Favre and Nguyen05]) and specifies them in terms of a model, the so-called *megamodel* [Favre04c] (see also e.g. [Bézivin et al.04]) where the extract in fig. 3.1 is taken from.

3.4.3 Practical Application and Profiles

From the practical point of view, Voelter [Stahl et al.07] mentions three different ways of defining a modeling language:

1. Definition of an independent metamodel
2. Definition of an own metamodel based on existing metamodels, e.g. the UML metamodel
3. Definition of an extension of the UML metamodel using the built-in extension mechanisms of UML

The first approach is useful for DSLs which are compact and/or do not have much in common with other metamodels. Besides, the MOF defined by the OMG there are several other meta-metamodels supported by tools, like *Ecore* which is part of the *Eclipse modeling Framework* [Steinberg et al.08], or *KM3* [Jouault and Bézivin06].

For languages providing concepts similar to those included in UML it is often useful to reuse the UML metamodel: Besides efficiency reasons, this can also increase the quality and the interoperability of the resulting metamodel. Technically, reuse on metamodel level can be achieved by importing or merging packages from the UML metamodel and adding new classes, e.g. using generalization relationships.

For the third approach, mainly so-called stereotypes are used. A *Stereotype* is part of the UML specification itself and allows the customization of UML metaclasses without changing the UML

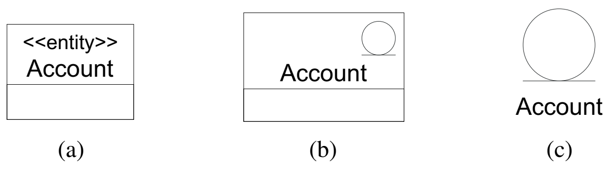


Figure 3.6: Notation options for UML stereotypes.

metamodel. An example is a stereotype `entity` for UML classes which specifies that a class represents a business concept which should be persistent. It extends the UML metaclass `class`. A stereotype is denoted in a model like the extended UML element but marked with either a stereotype label (fig. 3.6a) or, if available, an icon (fig. 3.6b). Alternatively the UML notation can be replaced completely by the stereotype icon (fig. 3.6c).

The advantages of stereotypes is that they are a more lightweight mechanism than changing the metamodel and are supported by many UML modeling tools. A disadvantage is the restricted flexibility as it is only possible to extend existing metaclasses, but not e.g. to add additional ones. A collection of stereotype definitions for a specific purpose is called *Profile* [Pleuß02].The pros and cons of Profiles versus DSLs have been subject of many discussions (e.g. [Desfray00, Kent05]). Of course, there is a large difference between a very specific DSL for a specific company and a Profile which is very close to UML. Nevertheless, there are many cases where the difference is not that large and the intermediate solution from above fits as well. [Abouzahra et al.05] proposes an approach for automatic transformation between models compliant to a given specific metamodel and models compliant to a given analogous Profile.

A more detailed discussion of alternative ways to customize UML is provided in [Bruck and Hussey07].

Tool Support

A large number of tools supports the UML as the de-facto modeling standard. [Jeckle04] lists more than hundred UML tools. Widespread commercial tools are for instance *Magic Draw* [No Magic], *IBM Rational Modeler* [IBM] (the successor of *Rational Rose*), *Poseidon* [Gentleware], and many others. Many of them provide also support for UML Profiles and other customization mechanisms.

Domain-specific modeling languages are traditionally supported by *Meta-CASE* tools [Isazadeh and Lamb97], like *MetaCase* [Meta] or *GME2000* [GME]. They provide an integrated support for definition of DSLs, creation of customized modeling editors, model validation, and code generation. Most of them use proprietary concepts for language definition and transformations. The *Microsoft DSL tools* [Cook et al.07] can be put into this category as well.

Several tool projects are under development which aim to support the latest concepts in MDE. An important example are the sub-projects of the open source tool *Eclipse* [Eclb] devoted to modeling. The *Eclipse Modeling Framework* (*EMF* [EMFb, Budinsky et al.03]) supports automatic generation of an Java implementation from a given metamodel. It also automatically generates a simple tree-editor which can be used to create models conforming to the given metamodel. More sophisticated, visual modeling editors can be created for Eclipse using the Eclipse *Graphical Editing Framework* (*GEF* [GEF]), a framework supporting 2D graphics and graphical editors in general. As an alternative the Eclipse *Graphical Modeling Framework* (*GMF* [GMF]) allows to specify visual editors in terms of a model and generate their implementation from the models. These projects together with many others devoted to modeling are subsumed in the *Eclipse Modeling Project* [Ecld] like tools for model-to-model transformations, model validation, model weaving, etc. For example, the *M2M* [Eclc] (sub-)project provides the transformation languages ATL and Procedural and Declarative QVT.

Consequently there are three typical kinds of support for implementing a custom visual modeling tool:

1. Extension mechanisms of existing (UML) modeling tools
2. Meta-CASE tools for domain-specific languages
3. Frameworks and APIs from the area of Modeling Driven Engineering

Benefits

The overall goal of MDE is increasing the productivity in the development process. [Atkinson and Kühne03] distinguishes between short-term and long-term productivity of artifacts created in the development process. Short-term productivity depends on the value which can be derived from an artifact in the development process. This is increased by MDE through automatic code generation. The long-term productivity depends on how long a created artifact stays valuable. It is increased by MDE as design knowledge is explicitly specified and documented in terms of abstract models as well as by the possibility to reuse platform-independent models for new platforms and devices.

More in detail, advantages of MDE combine advantages of a design phase using visual modeling languages with those of code generation. Compared to the whole spectrum of possible development processes, MDE provides benefits 1) resulting from a design phase compared to development without any design, 2) resulting from the usage of abstract, platform-independent, visual models compared to development without models, and 3) resulting from code generation compared to approaches without code generation.

However, there is an additional class of benefits resulting from MDE: The involved concepts are systematically defined in terms of models, metamodels, and transformations and can thus be easier managed, reused, and combined. Tools and frameworks can be reused for different application purposes. These properties do not improve the development of an application itself but the development of development support, i.e. they apply to a "metalevel" similar like metamodels.

Thus, expected benefits of MDE can be subsumed as follows[4]:

1. Benefits resulting from the existence of a design phase:
   - *Reducing the gap between requirements and implementation:* A design phase aims to ensure in advance that the implementation really addresses the customer's and user's requirements.
   - *Developer coordination:* Previous planning of the developed system enables the developers to coordinate their work e.g. by dividing the system into several parts and defining interfaces between them.
   - *Well-structured systems:* A design phase provides explicit planning of the system architecture and the overall code structure. This facilitates implementation itself as well as maintenance.
2. Benefits resulting from the use of visual abstract models:

[4]Within the categories ordered by the temporal occurrence in the development process

- *Planning on adequate level of abstraction:* Modeling languages provide the developer concepts for planning and reasoning about the developed system on an adequate level of abstraction.
- *Improved communication by visual models:* The visual character of modeling languages can lead to increased usability (understanding, percepting, exploring, etc., see sec. 3.4.1) of design documents for both author and other developers.
- *Validation:* (Semi-)Formal modeling languages enable automatic validation of the design.
- *Documentation:* Models can be used as documentation when maintaining the system.
- *Platform-independence:* Platform-independent models can be reused or at least serve as starting point when implementing the system for a different platform. This includes development platforms like a programming language or component model, as well as deployment platforms like the operating system or target devices.

3. Benefits resulting from code generation:

   - *Enhanced productivity:* Generating code from a given model requires often only a teeny part of time compared to manual mapping into code.
   - *Expert knowledge can be put into the code generator:* Expert knowledge – e.g. on code structuring, code optimizations, or platform-specific features – can once be put into the code generator and then be reused by all developers.
   - *Reduction of errors:* Automatic mapping prevents from manual errors.

4. Meta goals: Easier creation and maintenance of development support

   - *Knowledge about creation of modeling languages*: MDE concepts and definitions reflect existing knowledge about modeling, modeling languages, and code generation.
   - *Frameworks and tools:* Tools like Eclipse Modeling Tools (p. 41) provide sophisticated support for all steps in MDE like creating and processing metamodels, creating modeling editors, and defining and executing transformations.
   - *Maintenance of modeling language and transformations:* Systematic and explicit definition of metamodels and transformations facilitates maintenance of modeling languages and code generators.
   - *Reuse of metamodels and transformations:* MDE compliant explicit metamodels and transformations can easily be understood and reused by others.

Like in many areas of Software Engineering, there is unfortunately only few empirical data on the effect of MDE. Some experiments and reports from industry on the productivity of Domain Specific Languages (in the broadest sense) can be found e.g. in [Kieburtz et al.96, Long et al.98, MetaCase99]. The lack of systematic validation is probably also caused by the fact, that software development methods and processes are often hard to validate (see "Validation Techniques in Software Engineering" in section. 8).

3.5 An Illustrating Example Scenario

This section provides an example to illustrate the main ideas of the solution. The example application introduced here will also be used as running example for the following chapters of this thesis.

The example application is a racing game application. It is important to understand that the approach described here is not restricted to game applications – it can be applied to all the different

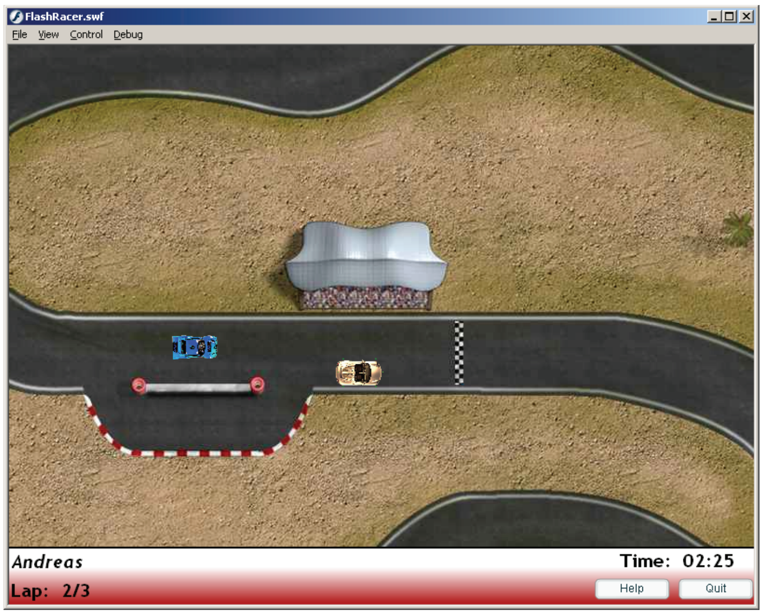


Figure 3.7: Screenshot from the example Racing Game application.

domains explained in section 2.4 (this will be shown later in section 8.3). On the other hand, not all kinds of games are suited to be developed with Flash: High-end 3D games are an exception within multimedia development as they require specific development methods and experts and are not produced in conventional multimedia authoring tools. However, technically less sophisticated games are indeed produced with authoring tools like Flash or Director [Bilas05, Besley et al.03], like produced for advertisement [Wernesgrüner], specific platforms [Phob], or even for conventional commercial purposes [Tap].

For the purpose of this thesis there are two important reasons while a small game serves best as example application: First, the requirements of common games are easy to understand without specific domain knowledge. For example, a learning application would require much previous explanation about its content and the media components which should be used. Second, games naturally make intensive use of all aspects of a multimedia application – individual media objects, application logic, *and* interaction – while other kinds of multimedia application sometimes cover only some of these aspects.

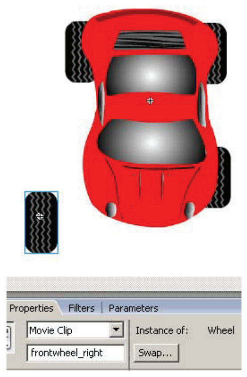
Figure 3.7 shows a screenshot of the application. It shows a screen with the actual race. The application consists of several other screens like a start screen, a menu, a help screen, etc. Several different Flash implementations (including multiplayer support) developed by students under supervision by the author of this thesis can be found at [MMPa]. A nice tutorial how to create racing game applications with Flash is also given in [Besley et al.03].

In a model driven development process with MML, first the requirements would be analyzed in conventional way like with any other development process. This can include first prototypes to gain knowledge about the application domain and discuss basic decisions with the customer, but also e.g. task models like *ConcurTaskTrees* [Paternò et al.97] to specify the general user tasks. During the design phase, MML models are used to specify the different aspects of the application. This includes the domain logic, the user interface in terms of abstract user interface elements, the interaction, and the media objects.

Specifying the application's media objects in particular means defining their interface to the appli-

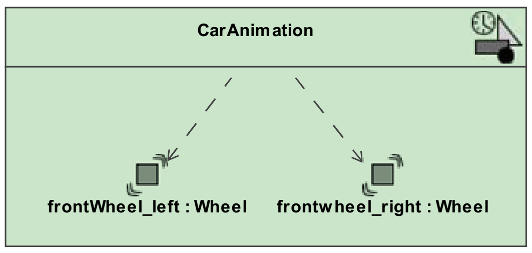
(a) Wheels when driving straight ahead.

(b) Wheels when driving a corner.



Properties Filters Parameters
Movie Clip Instance of: Wheel
frontwheel_right Swap...

(c) Wheels hence defined as independent parts

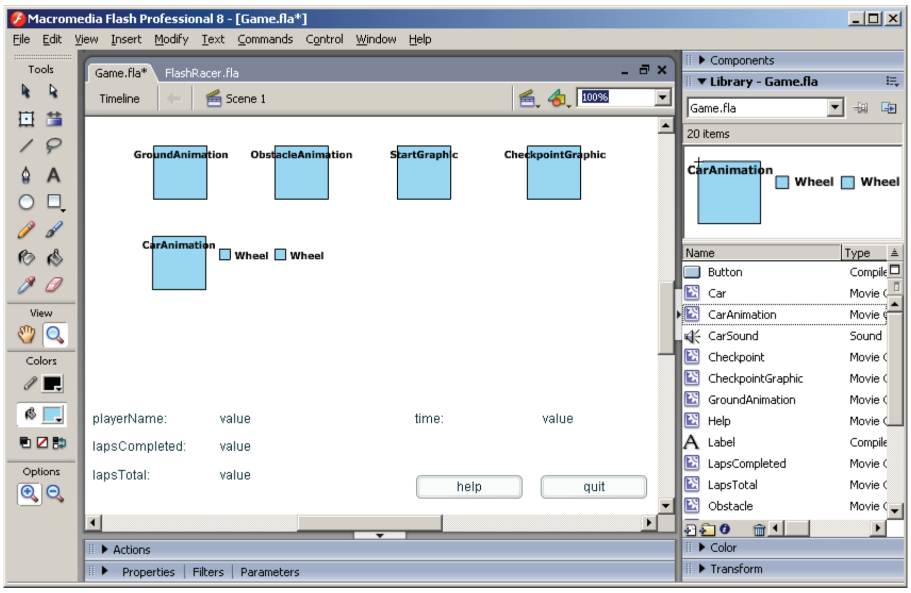


(d) Inner structure of media objects specified in an MML model.

Figure 3.8: Example: Wheels of a racing car defined as independent inner parts.

cation logic. For example, in a car racing game it might be desired that the car's wheels turn when the car drives a corner (figure 3.8). As the player controls the car interactively, the movement of the car is not a predefined animation but controlled by program code. For example, turning the car to the right requires in Flash a scripting command which increases the car's rotation angle like: `myCar._rotation += 5`. To achieve an effect like in figure 3.8b commands have to be added to turn the front wheels: `myCar.frontwheel_right._rotation += 5;`. This requires that the front wheels are not just part of the animation's graphics but instead they must be independent (sub-)animations with an own name to be accessed by the scripting code. These properties of the application can be specified in MML (figure 3.8d). The media designer can then design the media objects according to these requirements. Already the systematic documentation of such agreements and naming conventions between different developer groups can improve the development process significantly. Moreover, code generation from models enables to ensure and validate that the implementation conforms to them in a consistent way.

From the MML models, code skeletons can be generated for different platforms, in particular for authoring tools like Flash as in the example. It is possible to open the generated code directly in Flash and finish the implementation using the powerful abilities of the authoring tool. Figure 3.9 shows skeletons generated for from an MML model for the racing game. For the media components which require an individual design, placeholders are generated which contain an inner structure as defined in the MML model. They are filled out by the media designer. The user interface designer can finalize the user interface using the visual editor facilities, e.g. by visually laying out the generated components and replacing or adapting components where the generated ones are not sufficient. For the application logic, scripting code skeletons are generated which are finalized by the software designer



Name	Type
Button	Compile
Car	Movie (
CarAnimation	Movie (
CarSound	Sound
Checkpoint	Movie (
CheckpointGraphic	Movie (
GroundAnimation	Movie (
Help	Movie (
Label	Compile
LapsCompleted	Movie (
LapsTotal	Movie (
Obstacle	Movie (

Figure 3.9: The generated skeletons including placeholders can be directly opened and edited in the Flash authoring tool.

(programmer).

The generated placeholders can be edited without any restrictions or additional effort. The designers can freely use all available tools and features of Flash as usual. As overall structure and navigation of the application are already generated, the application can be started and tested at any time; independent from missing parts. The different developer groups can work in parallel as all interfaces between their artifacts to develop are already defined within the MML model. In addition, they are supported by a Flash plugin which helps them to finalize the implementation and allows tracing changes in the structure back to model, i.e. supports a *round-trip-engineering*.

Except the round-trip engineering which has been investigated mainly on conceptual level, full prototypical tool support has been developed within this thesis for all above-mentioned steps.

Chapter 4

Related Modeling Approaches

This chapter summarizes existing modeling approaches related to this thesis. While existing technologies and approaches for multimedia application development are introduced in chapter 2 and the general foundations on model-driven development in section 3.4, this chapter focuses on existing modeling concepts relevant for this work.

Currently only a very small number of modeling approaches for interactive multimedia applications exists. Section 4.3 describes them in detail. However, there is a large number of approaches which cover one or more aspects important for this thesis. These approaches originate from different research communities with different backgrounds. While the research work within each community is tightly related, there is sometimes a gap between the different communities. For instance, approaches for modeling multimedia does barely rely on knowledge from user interface modeling area – although the user interface is an important part of a multimedia application. Thus, this chapter aims to classify the existing work according to three identified communities and gives a summary on their general backgrounds and their points of view.

A large number of concepts for modeling user interfaces in general originates from the field of Human-Computer Interaction. Section 4.1 provides an overview about them. The area of Web Engineering addresses the automatic generation of Web applications based on models. It is briefly introduced in section 4.2. Finally, section 4.3 describes research efforts directly focusing on multimedia aspects.

4.1 User Interface Modeling

A user interface is part of any kind of application. Since windows-based graphical user interfaces (GUI) became common in the 1980's, user interfaces of applications became quite unary [Myers et al.00]. The same user interface concepts and tools – mainly GUI toolkits and GUI builders – have been used across all domains in software development independently from the kind of application. In contrast to other areas in computer science, user interface concepts and tool must not only fulfill technical requirements. Rather, the user's abilities and needs must be carefully considered which requires knowledge in cognitive sciences like physiology, sociology, or even philosophy [Rosson and Carroll02]. The research field of *Human-Computer Interaction* (*HCI*, [Hewett et al.92]) addresses these issues. Consequently, researchers in the HCI domain come from very different backgrounds. Approaches aiming for more systematic development, like model-based development, are usually not the main focus on large HCI conferences like CHI [CHI, Rosson and Gilmore07] or INTERACT [INT, Baranauskas et al.07], but they are frequently present. In addition, a community directly addressing

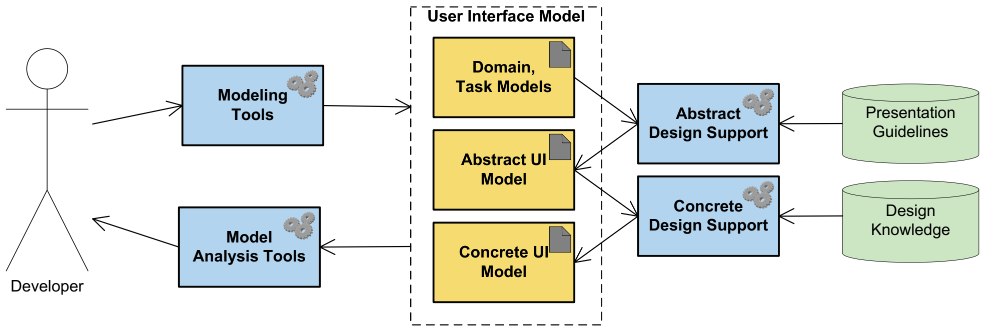


Figure 4.1: Model-based User Interface Development Environments according to [Szekely96, da Silva00]

user interface modeling has evolved. The most popular events in this specific area are CADUI [CAD, CAD07], DSV-IS [DSV, Doherty and Blandford07], and TAMODIA [TAM, Winckler et al.07] which is held in 2008 together with HCSE at EIS [EIS].

The following sections will first introduce the common concepts in user interface modeling, show some selected examples (*ConcurTaskTrees*, *UsiXML*, *Dynamo-Aid*, and *UIML*), and finally discuss the relationship to Software Engineering concepts like model-driven engineering.

4.1.1 General Concepts

A large amount of user interface modeling approaches has been proposed over the years from different directions. Early examples of declarative languages allowing the abstract specification of user interfaces can be found in 1985 with Cousin [Hayes et al.85] and ADM [Schulert et al.85]. They evolved to *Model-Based User Interface Development Environments* (*MB-UIDE*) which used different kinds of models to guide the developer from abstract specifications to the final user interface. Figure 4.1 shows the model-based development process and the common kinds of models according to overview papers like those by Szekely [Szekely96] or da Silva [da Silva00].

The most abstract kinds of models commonly used are task model and domain model. The *task model* specifies the tasks which are activities either by the user or the system which have to be accomplished to reach the user's goals. Examples for kinds of task models are *CTT* (*ConcurTaskTrees*, [Paternò et al.97, Paternò99], see section 4.1.2) or *TKS* (*Task Knowledge Structure*, [Johnson and Johnson89, Johnson91]). An overview on task models is provided in [Limbourg et al.01].

The *domain model* (also *called application* model or *data model*) specifies the structure of the application logic. It can be specified for instance in terms of a UML class diagram or an Entity-Relationship Diagram [Chen76].

The *Abstract User Interface Model* (*AUI*) is specified based on the task model and the domain model. It is sometimes composed of an *abstract presentation model* and a *dialogue model*. It specifies the user interface in an abstract and platform-independent way in terms of abstract interaction objects. *Abstract Interaction Objects* (*AIO*, introduced in [Vanderdonckt and Bodart93]), sometimes also called *interactors*, are user interface objects which enable the user for instance to input data or select an object on the user interface. They can be seen as an abstraction of widgets and are independent from any visual representation, platform or modality. The AIOs are grouped into *Presentation Units* (also called *Presentations*, *Views*, or *Interaction Spaces*) which can be seen as the abstraction of a window in a graphical user interface. For the abstract presentation model, no common diagram used throughout different approaches exists.

The dialogue model specifies the dialogue how to interact with the AIOs. It can be specified for example in terms of State-Transition diagrams [Wasserman85], or Petri Nets [Palanque et al.93]. An comparison of different possibilities for modeling the dialogue is provided in [Cockton87]. Some approaches use only the task model, or a refined version of it, to specify the interaction and do not use an additional dialogue model. Thus, the term abstract user interface model refers sometimes to the abstract presentation model only.

Finally, the *Concrete User Interface Model* (*CUI*) realizes the AUI for a specific modality in terms of concrete widgets and layout.

While most existing approaches and tools comply more or less with this general framework, the concrete kinds of models and diagrams used by them varies significantly. Several surveys list and compare the models used in the different existing approaches, e.g. Griffiths [Griffiths et al.98], Schlungbaum [Schlungbaum96], daSilva [da Silva00], and Gomaa [Gomaa et al.05]. In addition, [Limbourg04] compares various approaches regarding further properties like mappings and transformations between the models and methodological issues. MB-UIDEs surveyed in at least three of these comparisons are ADEPT [Markopoulos et al.92], AME [Märtin96], FUSE [Lonczewski and Schreiber96], GENIUS [Janssen et al.93], HUMANOID [Szekely et al.92], JANUS [Balzert95], MASTERMIND [Szekely et al.95], MECANO [Puerta96], MOBI-D [Puerta and Maulsby97], TADEUS [Elwert and Schlungbaum95], Tealleach [Griffiths et al.99], TRIDENT [Bodart et al.95], and UIDE [Foley et al.91]. Altogether, 34 approaches were compared. This shows that there exists a really large number of relevant proposals. The missing common agreement about the best models and diagram types to be used is one of the main problems of MB-UIDEs [da Silva00, Clerckx et al.04] and might be one of the reasons why user interface models have not gained stronger popularity until now.

Even if user interface modeling has not become widely established until now, paradigms like Ubiquitous Computing [Weiser99] or Ambient Environments [Wisneski et al.98] which strongly influence the Human-Computer Interaction area might significantly increase the importance of user interface modeling: Future applications are expected to run on various different devices, like mobile devices, public displays or computers embedded into items of every-day life. Moreover, expectations include that users will be able to seamlessly switch applications between devices and be able to share information everywhere they are. To provide the desired degree of usability, applications must be able to adapt to the various contexts of use which includes the user, the devices, and further influencing properties of the environment like the user's location [Schmidt et al.99, Coutaz and Rey02]. In such a scenario it will be very difficult to design the user interfaces for all the different contexts and devices by hand. Moreover, an application's user interface should be able to adapt to new kinds of devices and contexts of use which have not been prospected at design time. Thus, it will be necessary to specify user interfaces on a higher level of abstraction from which the user interfaces adapted to the current context can be derived. User interface models will then play an essential role [Myers et al.00].

Current approaches in the user interface modeling area focus on such capabilities. As a result, the general MB-UIDE framework from figure 4.1 has been extended by the CAMELEON reference framework [Calvary et al.03] shown in figure 4.2. It adds generic models required to capture the context of use of the application like *user model*, *platform model* and *environment model*. The *evolution model* specifies the how the application switches between different configurations according to relevant context changes. The *transition model* specifies how these transitions between different configurations should be performed avoiding discontinuities between them. These models, together with the task model and the domain model (here: *concepts model*) are the initial models which have to be specified by the modeler during the design process. From these models the specification of Tasks-and-Concepts, Abstract User Interface and Concrete User Interface is derived.

The horizontal translations illustrate the different configurations of the application according to

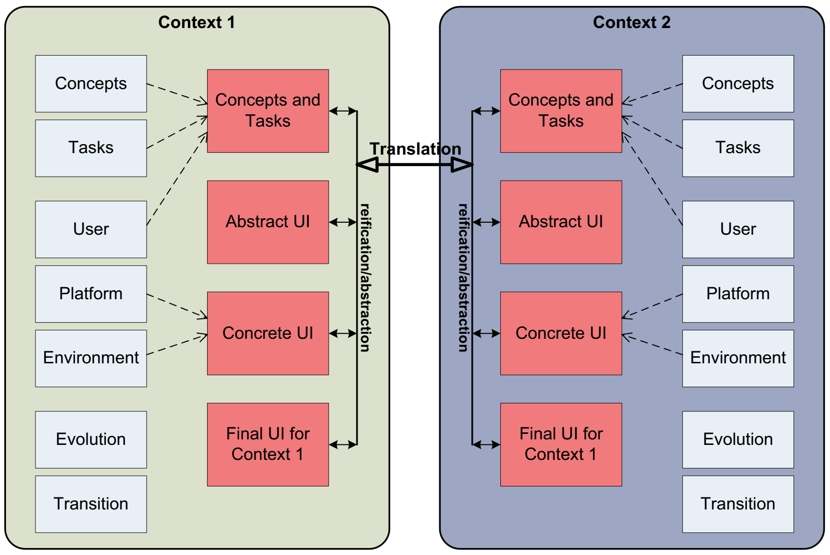


Figure 4.2: The CAMELEON reference framework according to [Calvary et al.02, Calvary et al.03]

different contexts. The vertical transformations specify the refinement of specifications during the design process. The reference framework explicitly allows starting the development process on any level of abstraction as well as a flexible order of transitions between levels of abstraction because many approaches are not strictly top-down or support reverse engineering steps. [Calvary et al.02] examines several existing approaches in terms of the framework. Furthermore, a general schema for the application's context-sensitive behavior at runtime is provided.

An advanced feature of user interfaces in ambient environments is the capability of flexible migration over different devices to utilize available devices as profitably as possible. For example, if the user gets to a room with a large display she wants for some information to be presented on the large display. In particular, it is sometimes desirable to allow not only migration of the whole user interface but rather partial migration: For example, when using a large display it might be useful to provide the visualization part (i.e. purely output) on the large display while the control part (buttons etc.) should be rendered on the user's personal mobile device as supported by [Bandelloni and Paternò04, Bandelloni et al.04] or [Braun and Mühlhäuser05]. A taxonomy and general discussions of user interface migration can be found in [Berti et al.05, Luyten and Coninx05].

Two concrete examples approaches are described in the next section: *UsiXML*, and XML-based approach which realizes the design-time part of the reference framework very closely, and *Dynamo-Aid* which provides an example for context-sensitivity at runtime.

A subclass of user interface modeling languages with increasing practical relevance are XML-based languages focusing only on concrete user interface specification. They allow a declarative and often platform-independent specification of the user interface but do not aim to support the development process or to provide abstract models. The user interface specifications can be executed for example by mappings onto specific platforms, like in *UIML* (see section 4.1.2), or by player components. Of course, the latter ones can only be executed on platforms for which a player component exists. Such languages can be found more and more even in large open-source projects or as part of commercial tools by large vendors. For example, *XUL* [XUL] is part of the *Mozilla Project* [Moz] and is interpreted by the *Mozilla Browser*. *MXML* is provided by *Adobe* [Ado] and enables declarative specification of user interfaces for the *Flex framework* [Kazoun and Lott07] and is compiled into Flash files to be executed in the Flash player (see section 2.3). *XAML* [XAM] is developed by *Microsoft* [Mic] and can be compiled into Windows .NET applications or be interpreted by XAML players.

Due to their declarative nature, their ability to run on several platforms, and their similarity with CUI models, it is often useful to consider XML-based user interface specification languages either as target format or directly as CUI model in model-based user interface development approaches.

4.1.2 Concrete Examples

This section describes some concrete exemplary approaches in this field. ConcurTaskTrees are one of the most important approaches for task modeling. UsiXML is a comprehensive approach closely compliant to the CAMELEON reference framework, enabling the developer to move between different levels of abstraction and surrounded by various tools. Dynamo-Aid supports development and a runtime environment for context-sensitive user interfaces. Finally, UIML is shown as an example for an platform-independent XML-based user interface specification language.

ConcurTaskTrees

One of the most popular approaches for task modeling is the *ConcurTaskTree* (*CTT*) notation by Paternó [Paternò et al.97, Paternò99] which partially bases on LOTOS [ISO88], a formal language for concurrent systems specification. It supports four kinds of tasks:

User Tasks are performed by the user. Usually they are cognitive activities relevant when using the application like making a decision.

Interaction Tasks require an interaction between the user and the system like pushing a button.

Application Tasks are performed by the system like searching in a database or calculations.

Abstract Tasks represent complex tasks which are further subdivided into several subtasks.

A CTT model consists of tasks hierarchically structured in a tree structure. If child nodes of a task are all of the same type, the parent node is of the same type as well. Otherwise the parent node is an abstract task. Temporal relationships between the tasks are specified using operators shown in figure 4.1.

Figure 4.3 provides an example for a task tree. It shows an extract of a music player application which can run on a PC and a mobile phone. It was developed in a diploma thesis supervised by the author where several user interface modeling approaches were compared [Wu06b].

It is possible to calculate *Enabled Task Sets* (*ETS*) from a task model. “An enabled task set is a set of tasks that are logically enabled to start their performance during the same period of time.” [Paternò99]. This means the tasks which must be accessible at the same time for the user according to the hierarchy and the temporal operators in the task model and should thus be available within the same presentation unit of a user interface. (It should be mentioned that it is not always possible to present the user all tasks from an ETS in parallel, e.g. when facing a very large task model or for mobile devices with a small display. Such cases require additional decisions which tasks to present.)

For the model in figure three Enabled Tasks Sets are calculated:

- *ETS 0*: *adjust volume*, *list songs*, *quit*
- *ETS 1*: *adjust volume*, *select song*, *quit*

Operator	Description
$T_1[]T_2$	Choice
$T_1\|=\|T_2$	Order Independence
$T_1\|\|\|T_2$	Independent Concurrency
$T_1\|[]\|T_2$	Concurrency with information exchange
$T_1[>T_2$	Disabling/Deactivation
$T_1\|>T_2$	Suspend resume
$T_1>>T_2$	Enabling
$T_1[]»T_2$	Enabling with information passing
T_1*	Iteration
$T_1(n)$	Finite Iteration
$([T_1])$	Optional Task

Table 4.1: Temporal operators in CTT on tasks T_1, T_2. The binary operators are listed with descending priority.

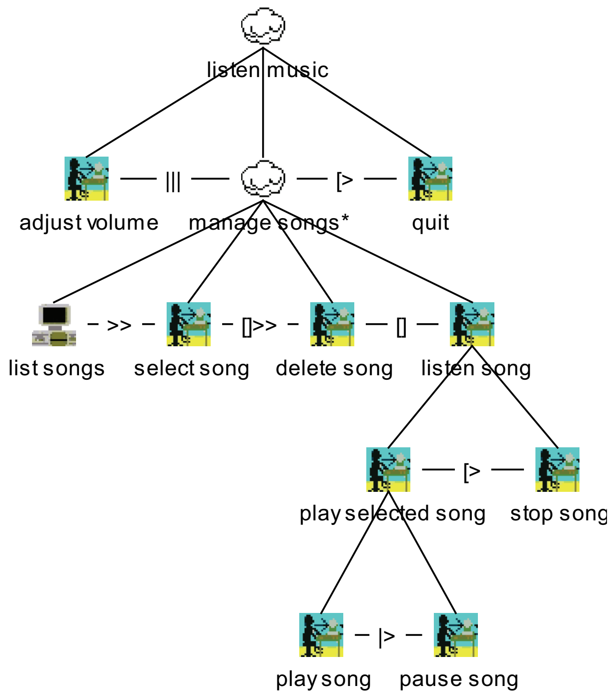


Figure 4.3: Example ConcurTaskTree Model from [Wu06b]

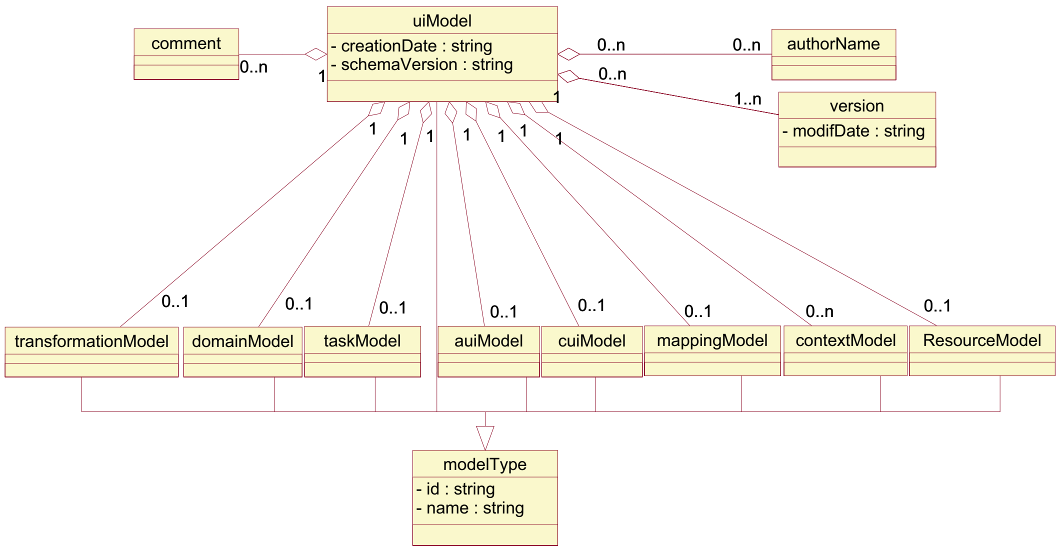


Figure 4.4: Extract from the UsiXML metamodel [Usi06] showing the UsiXML models.

- *ETS 2*: *adjust volume*, *delete song*, *play song*, *pause song*, *stop song*, *quit*

A development method based on CTT supporting the development of multimodal user interfaces for multiple target devices is presented in [Paternò and Santoro02, Mori et al.04]. It is supported by a modeling tool, *TERESA*, which is freely available [TERa].

UsiXML

UsiXML (USer Interface eXtensible Markup Language, [Limbourg et al.04, Usib] is a model-based approach close to the CAMELEON reference framework supporting user interfaces for multiple contexts. The language is based on XML and is defined in terms of a metamodel [Usi07]. Figure 4.4 shows the supported models. As their multiplicity is ‘0..1’ or ‘0..n’ it is possible to flexibly combine the models according to the modeler’s needs.

The *domain model* corresponds to UML class diagrams. The task model is based on CTT. The *abstract user interface model* is independent from platform and modality and specifies *abstract interaction objects* (*AIO*s) and relationships between them. An abstract interaction object (AIO) can be either a *container* or an individual AIO which consists of one or more of the four facets *input*, *output*, *navigation*, and *control*. The relationships between them specify for example decomposition, or spatio-temporal relations.

The *concrete user interface model* is modality-dependent but still platform-independent. It defines the user interface in terms of *concrete interaction objects* (*CIO*s) – platform-independent widgets like *Button*, *Label*, etc. in case of a graphical interface – and their layout. It currently supports graphical user interfaces and vocal interfaces. The CIOs can be associated with behavior like operation calls triggered by events.

The *context model* specifies the context of use including the *user*, the *platform* and the *environment*. Furthermore, it contains a set of *plasticity domains* which defines the specific contexts for which e.g. an AIO or CIO should be available. The *resource model* can be used to specify content for the interaction objects which depends on localization aspects, for example text in different languages or the reading path. The *transformation model* enables to define transformations on the models e.g. from

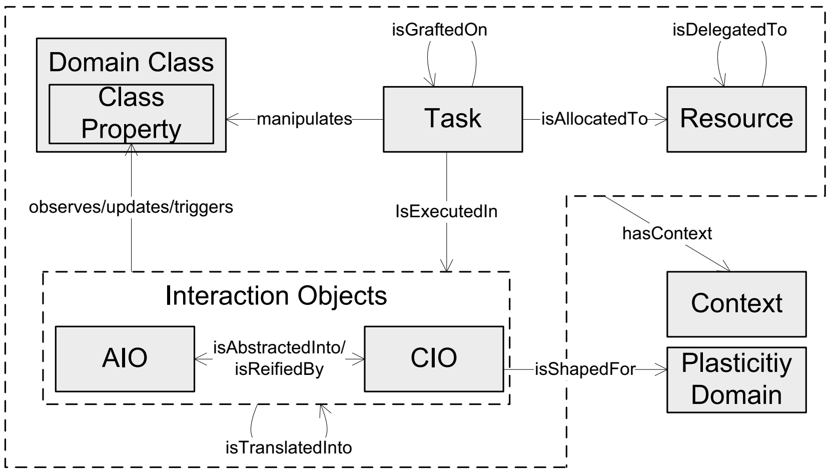


Figure 4.5: Relationships between different models in UsiXML

a task model elements to interaction objects or transitions of the interaction objects according to the context.

The *mapping model* is used to specify relationships between models (often called *inter-model relationships* in contrast to *intra-model relationships* which reside within a model). They are important parts of the user interface specification, as the mapping model connects the model elements from the other models and thus provides important information for the overall model. Figure 4.5 shows the relationships between model elements from different models in UsiXML. For purpose of illustration the figure depicts the relationships (metaclasses in UsiXML) as named relationships between models and/or model elements. Some models or model elements are clustered into groups. For example, the relationship *hasContext* can be specified between a context and any other model or model element. As UsiXML is close to the CAMELEON reference framework the UsiXML mapping model provides a good example for conventional relationships between user interface models in general.

A visual notation is not part of the UsiXML specification. The concrete syntax is XML but it is intended to provide various tools for convenient creation of the models. Task and Application Models can be created with IdealXML [Ide]. GraphiXML [Michotte and Vanderdonckt08, Gra] is a GUI builder for UsiXML and allows storing the results as AUI and as CUI. To enable a less formal development *SketchiXML* [Coyette et al.07, Ske] is a tool for user interface prototyping which allows export into UsiXML specifications. Besides, various other tools exist, like and a tool for defining and executing transformations based on graph grammars [Montero et al.05, Tra], code generators and interpreter for various platforms (e.g. Java, Xul, and XHTML [Usia]) or a tool for reverse engineering HTML pages into AUI and CUI models [Bouillon et al.04, Rev].

Dynamo-Aid

An approach supporting context adaptations at runtime is *Dynamo-Aid* (*Dynamic Model-Based User Interface Development*, [Clerckx et al.05a, Clerckx et al.05b, Clerckx and Coninx05] which is part of the framework *Dygimes* [Coninx et al.03, Luyten04]. Dynamo-Aid includes a modeling tool supporting the development process as well as a runtime architecture.

Dynamo-Aid supports the following models: Context-sensitive task model (also called Dynamic task model), Context Model, Dialog Model, Presentation Model, and Interface-Model.

The *context-sensitive task model* is an extension of CTT enabling context-sensitive task models.

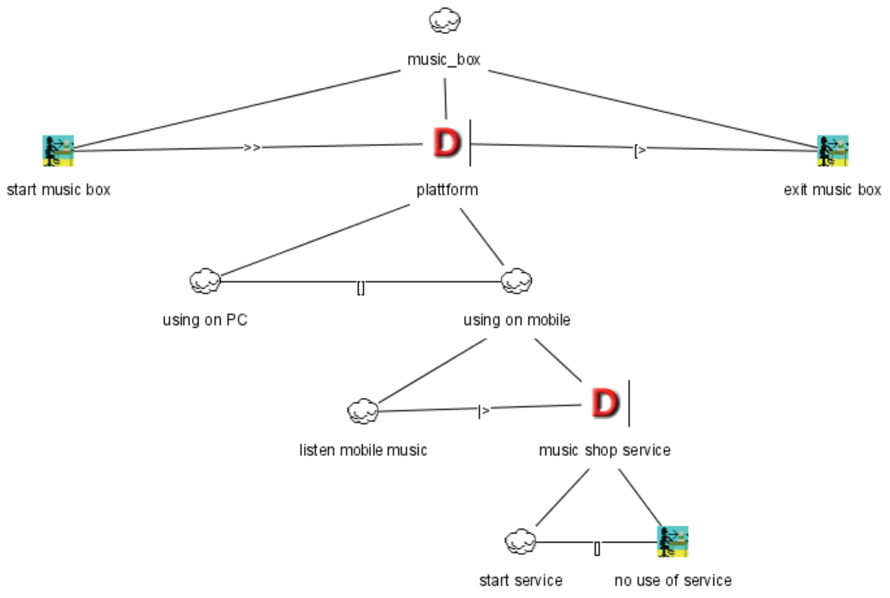


Figure 4.6: Example for a Context-sensitive task model in Dynamo-Aid.

Therefore, a it supports a new kind of task, the *decision task*. A decision task has several subtasks. At runtime the system decides according to the context which of them is active. Figure 4.6 shows the context-sensitive task model for the example music player application from [Wu06b]. Depending on the context, the application is either executed on the desktop PC or the mobile phone. In addition, a specific location-based service is available for the mobile phone version: When the user enters a music shop (supporting this service) the application offers the user to listen to some sample versions of new songs.

The *context model* consists of *Concrete Context Objects* (*CCO*s) and *Abstract Context Objects* (*ACO*s). A CCO represents low-level context information, like obtained by a sensor. An ACO is connected to CCOs and interprets their information to provide context information which is relevant for the application. The ACOs can be connected to decision tasks in the context-sensitive task model to specify the context information which determines the selection of tasks at runtime.

For each context of use a *context-specific dialog model* is calculated from the task model. It consists of states and transitions. A state corresponds to an enabled task set (see CTT in 4.1.2). A transition is associated with a task which triggers the transition. For the calculation of enabled task sets, Dynamo-Aid implements an algorithm presented in [Luyten et al.03] which bases on the algorithm from [Paternò99] and uses heuristics given in [Paternò and Santoro02]. Figure 4.7 shows the context-specific dialog model for the example application by [Wu06b] calculated from the task model from figure 4.3. The Enabled Task Sets correspond to those in section 4.1.2. *ETS-1* corresponds to a terminal state.

The modeler specifies the *context-sensitive dialog model* by defining transitions between the states of different context-specific dialog models. They have to be defined manually to avoid any context change which is not desired.

The *presentation model* is defined by selecting AIOs and assigning them to the tasks. Available AIOs are *Choice*, *Input*, *Static*, *Navigation Control*, *Hierarchy*, *Canvas*, *URI*, and *Group*. The Dynamo-Aid modeling tool supports hierarchically structuring the AIOs and provides support when

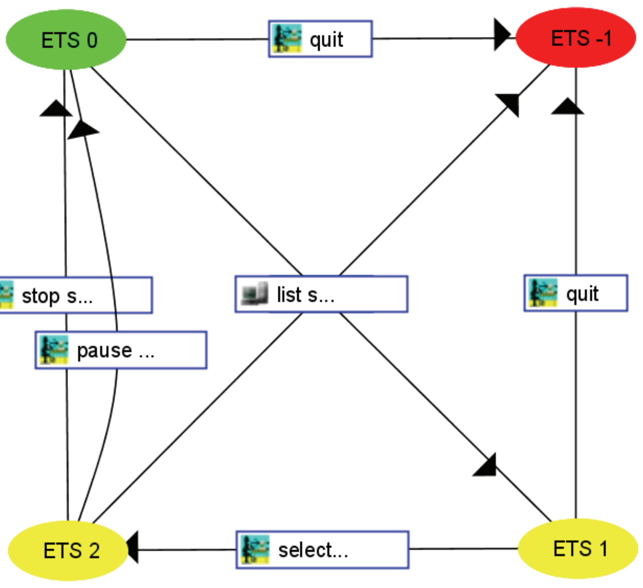


Figure 4.7: Example for a Context-specific dialog model calculated from task models in Dynamo-Aid.

assigning the associated tasks. However, the links between the AIOs and the tasks are not directly visible in the diagram.

The *context-sensitive interface model* contains the aggregation of all the models defined before and thus provides an overview of all relationships between them. As the number of relationships is usually very large, the tool supports to show or hide them and to mark them with different colors and provides semantic zooming.

UIML

The *User Interface Markup Language* (*UIML*, [Abrams et al.99, Phanouriou00, UIM]) allows to specify the concrete user interfaces independently from the platform. The language is based on XML. The current version is 3.1 [Abrams and Helms04].

UIML enables to specify user interfaces in a platform-independent way. This is realized by *vocabularies* for the different platforms. A vocabulary is a mapping from UIML specifications onto the corresponding implementation for a specific platform, e.g. onto Java Swing classes. When using UIML one can either use an existing vocabulary or create an own. According to the UIML website vocabularies are currently available or under development for the following target platforms: Java, J2EE, CORBA, C++, .NET, HTML, Symbian, QT, Visual Basic, VoiceML, and WML.

UIML aims to fully support all user interface elements and properties of the target platforms. On the other hand, it aims for the highest possible extensibility. For that reason, the available user interface elements in UIML are not defined by UIML itself but within the vocabularies. The structure of UIML documents is very modular and provides a strict separation between structure, content, layout, and behavior. A UIML document contains the following parts:

Structure: The structure of the user interface in terms of widget objects, like a panel or a button, and relationships between them. The relationships are usually spacial for graphical user interfaces and temporal for vocal ones. They are defined by specific widget classes like containers. All available widget classes are defined in the vocabulary. An application may have several different user interfaces structures for different purposes, e.g. a vocal and a graphical. It is also possible to define dynamic changes of the structure by restructuring.

Style: The style of user interface elements in terms of properties. For example, assigning a text to a button label or gray color to all buttons. Properties are defined in the vocabularies and assigned to widget objects from the structure section.

Content: Content on the user interface, for example strings to be used as text on the user interface or as label for a button. The content can be referenced by properties in the style section.

Behavior: Behavior of the user interface in terms of conditions and corresponding actions. The conditions are usually events from widget objects. Logical operations and other boolean expressions can be used for complex conditions. Actions allow either calling a method from the application logic or assigning a value to an object's properties. A value can be any of the following: a constant value, a reference to a constant, the value of a property or the return value of a method call. Available events are defined in the vocabulary. The application logic is defined in the logic section.

The sections above contain the actual interface definitions. As already mentioned, two additional sections may be necessary:

Logic: Objects representing the application logic together with mappings on the actual implementations. The application logic objects are just a collection of application methods and can be mapped to any kind of implementation, like a database, script snippets, or Java objects.

Vocabulary: The vocabulary defines all the elements used for the interface definitions which includes the widget classes to be instantiated in the structure section and associated class properties. In addition, events and listeners are specified as classes and used by the widget classes. For all classes and properties, mappings onto the target implementation platform must be specified.

The described mechanisms show that languages like UIML clearly aim for a pragmatic, detailed specification of the concrete user interface. Abstraction and platform-independence are hence limited compared to other approaches or depend on the vocabularies. The main contribution lies in the single declarative language which can be used for different platforms and the general framework for strict separation of different user interface aspects. These properties also make such languages a useful candidate as target language for transformations from more abstract models.

4.1.3 User Interface Modeling and Software Engineering Concepts

This section briefly discusses model-based user interface development approaches from the viewpoint of Software Engineering and MDE as relevant for this work. The relationship to Software Engineering concepts and standards (see section3.4) are indeed discussed and often mentioned as one of the important challenges, like in [Clerckx et al.04] or in group discussions like [Basnyat et al.05].

Compliance

Considering Software Engineering concepts can be performed on two levels: First, the user interface must be linked to the the application logic. Provided that the application logic is developed using models as well, it is useful to enable links between these models. Many approaches do already fulfill these requirement by including an application model into their approach. Of course, it is useful to enable compliance with application models in Software Engineering by either using UML, as the de-facto standard in Software Engineering, or by enabling flexible usage of any application model. Second, it can be useful to adhere to Software Engineering standards in general, as they are subject of intensive research and well established. This enables reuse of general concepts and tools and might also increase the general acceptance of an approach.

Several contributions discuss the integration of user interface modeling and UML. Standard UML does not explicitly support user interface modeling concepts. For example, there is no kind of abstract user interface element. Of course, on implementation level a widget is just an ordinary class and can thus be specified in an ordinary UML class diagram. However, its semantics then would not differ from any other application class, which would not be useful for meaningful user interface modeling. Modeling the dialogue is easier with UML as behavior diagrams like State Charts can be used for this purpose. Finally, the tasks, as one of the central user interface modeling concepts, are not explicitly supported in UML but it is subject of several discussions (e.g. [Trætteberg02, Paternò01]) whether they can be substituted by existing UML elements with similar semantics. On a first look, UML Use Cases seem similar to tasks as they specify the actions a system can perform when interacting with the user. However, Use Cases focus on system when interacting with one or multiple actors while task models focus on the individual user and his goals [Paternò99, Markopoulos and Marijnissen00, Constantine and Lockwood01]. Another possibility is to substitute tasks by actions from UML Activity Diagrams as discussed in [Van den Bergh06, Nóbrega et al.05]. While this is basically possible, authors agree that using standard Activity Diagrams would lead to a limited usability for the modeler and adaptations are desired.

In particular for transformations into other models or code, it is often necessary to specialize the semantics of UML even if the notation remains unchanged. For example, defining that in a State Chart each state represents a Presentation Unit is an extension of the the semantics of UML states. Thus, UML usually has to be extended anyway. Existing UML extensions for user interface modeling mainly use the stereotype mechanism (see section 3.4.3). Examples are the *Wisdom* approach [Nunes and Falcão e Cunha00, Nunes01] or UMLi [da Silva and Paton00, da Silva and Paton03] which support the basic user interface models like in figure 4.1. The *Context-sensitive User Interface Profile* (CUP, [Van den Bergh and Coninx05, Van den Bergh and Coninx06, Van den Bergh06] supports a similar approach like Dynamo-Aid (section 4.1.2). Besides, a few other approaches aim for a integration of the task concept with more general Software Engineering concepts: For example, [Sinnig et al.07] defines a common semantic domain for task models and Use Cases or [Bourguin et al.07] describes a component-based approach where each component represents a (generic) user task.

As the area of model-driven engineering is relatively young, currently only a few user interface modeling approaches comply to its concepts and upcoming standards. Basically, many user interface modeling approaches constitute themselves as “model-based” (instead of “model-driven”) but this is often not intended as a statement about the degree of automation. UsiXML provides explicit metamodels and supports transformations and adheres to many MDE concepts [Vanderdonckt05]. Some of the latest approaches explicitly adhere to MDE concepts and tools: [Botterweck06] addresses the development of user interfaces for multiple target devices. [Sottet et al.07b, Coutaz et al.07] focuses on development and runtime adaptation of user interfaces for ambient spaces. Both approaches provide EMF-compliant metamodels and ATL transformations between them.

Automation and Usability

An important general challenge lies in the degree of automation and, related with that, the usability of resulting user interfaces. Applying the idea of model-driven development, more concrete models would be automatically derived from abstract models by transformations and modified and completed by the developer. On the other hand, automation can easily lead to user interfaces whose provided degree of usability is not sufficient. An example for intensive automation is JANUS [Balzert95, Balzert et al.96] which is still available as a commercial tool [Otr]. It generates the user interfaces directly from the domain model. Although it provides various possibilities for the user to tune the transformation, the resulting user interfaces tend to reflect the application model instead of the user's conceptual model. Such user interfaces are useful to provide user access e.g. on database values but are often not sufficient to support a less experienced user through his/her tasks. Thus, such highly automated development approaches are usually considered as adequate only for very specific application domains [Puerta and Eisenstein99]. [Arens and Hovy95] is another example addressing in particular multimedia: It proposes an intelligent system (called *Cicero*) which aims to automatically select a media representation for a given piece of information and specified user.

The opposite alternative is to specify all models up to the final implementation manually – maybe even in parallel and by different persons – and finally to specify manually the relationships between them required to complete the overall specification – e.g. between task model or dialog model and interaction objects. In the literature the problem how to establish the relationship between the different models is referred to as *Mapping Problem* [Puerta and Eisenstein99] which is discussed in several contributions [Limbourg et al.00, Clerckx et al.04, Montero et al.05]. As pointed out by [Szekely96], any approach should give the possibility for manual post-editing to provide the interface designer the final control about the results and not to hinder him/her to revise the final results.

While manual specification of models and mappings aims to ensure usability by leaving the responsibility to the human developer, it remains still desirable to increase productivity by a possibly high amount of automation. One of the most important arguments for user interface modeling mentioned above – the possibility to generate user interfaces for different devices which might even be unknown at design time – would become quite weak if no (semi-)automatic transformation from platform-independent user interface models to platform-specific user interfaces exist. Moreover, even a non-automated but systematic approach should include as much knowledge as possible about how to systematically achieve appropriate usability. Thus, it is useful to formalize knowledge about usability, or at least make it explicit, as much as possible.

A common definition from the ISO standard on *Ergonomics of Human System Interaction* [ISO98] defines usability as:

> The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

From the viewpoint of Software Engineering usability is usually considered as one of the non-functional requirements. While some approaches for formalizing and modeling non-functional requirements already exist [Zschaler07], it currently seems not possible to adopt them for usability as there currently is not even a concept how usability could be measured at all [Seffah and Metzker04].

The HCI domain has provided several collections of usability guidelines and patterns. Guidelines are generic rules to be obeyed when designing a user interface. Some guidelines are often very concrete, as found in guidelines for operation systems like the *Apple Human Interface Guidelines* [App08] or the *GNOME Human Interface Guidelines* [GNO04] for GNOME desktops. They specify for instance the look of widget components and distances between them on the screen. Such guidelines can often be integrated into code generators easily so that the code generator helps to obey them. Other guidelines are very generic and qualitative rules like the design rules in [Shneiderman and Plaisant04, Nielsen93, Preece et al.94, Constantine and Lockwood99], for instance the rule that a user interface should be consistent. Some formal rules can be indirectly derived from them, e.g. by generating the same kind of widgets for similar user tasks. Model driven development can be valuable for realizing such rules as it allows to implement, manage, and maintain them by explicit declarative transformation rules, as sketched in first proposals by [Sottet et al.06, Zhao and Zou07].

However, the problem remains that usability guidelines are often contradictory, e.g. using the same kind of widgets for similar tasks can conflict with the rule that the user interface should prevent errors, for instance by using widgets which allows only valid inputs. A possible solution can be to treat this as an optimization problem with a cost function which is user specific and adapts according to the user's usage of the generated user interfaces like in SUPPLE [Gajos and Weld04].

Another promising solution is to build on existing established manually created building blocks, i.e. *user interface patterns*. Examples for existing user interface pattern collections are [Tidwell05, van Welie, Duyne et al.02]. An approach for integrating patterns into user interface generation is presented in [Radeke and Forbrig07, Radeke et al.06]. They provide the *User Interface Pattern Extensible Markup Language* (*UsiPXML*) to describe patterns and their implementations in machine-readable form. UsiPXML describes the patterns in terms of model fragments based on an adapted version of UsiXML. The authors propose a general framework how to integrate patterns into model-based user interface development and apply it as example to their development approach presented in [Wolff et al.05].

The model driven solution from Sottet et.al. aims to integrate such mechanisms. They propose a mix of automated, semi-automated and manually performed transformations. The approach aims to enable flexible integration of any usability framework into the transformation. The transformations are treated as models and, thus, usability guidelines can be managed and maintained in terms of models as well. In particular, a transformation can be associated with properties representing one or more usability rules (see [Sottet et al.07a]).

As follow up step on that base, it seems desirable to create customized user interfaces for the user interface designer which allow to manage the transformations – including guidelines and patterns – as proposed e.g. by [Sottet et al.06]. This idea was already supported to some extent by earlier tools like Mobi-D [Puerta and Eisenstein99]. Moreover, the end-user might also need a "Meta-UI" which allows to control and customize the user interface adaptations at runtime in ambient spaces as claimed by [Coutaz06].

Initiatives

In general, modeling is a highly active research area in Software Engineering (see section 3.4). Hence the interchange between user interface modeling area and Software Engineering area is important. User interface modeling can profit from the evolving concepts, tools, and standards from the MDE community. In turn, results from HCI are important for Software Engineering as the usability of applications is a key factor for its success. In addition, user interface modeling can be an important application area for applying and evaluating the theory of modeling as due to the large experience existing in this field. An increasing number of efforts investigates into this directions, e.g. conferences and workshops like the HCSE conference mentioned above or HCI-SE and books like [Seffah et al.05]. Another initiative is the workshop on *Model Driven Development of Advanced User Interfaces* (*MDDAUI*) co-organized by the author of this thesis and so far held three times on the MODELS conference (one of the main conferences on MDE) in 2005 [Pleuß et al.05b, Pleuß et al.05a], 2006 [Pleuß et al.06b, Pleuß et al.06a], and 2007 [Pleuß et al.07c, Pleuß et al.07a]

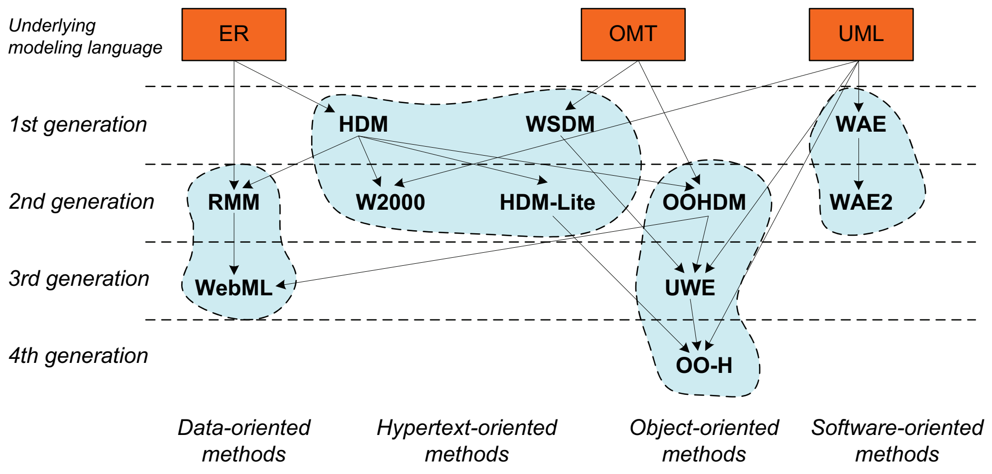


Figure 4.8: Web Engineering approaches according to [Schwinger and Koch03]

4.2 Web Engineering

As discussed in section 2.1.4 Web and multimedia applications have many commonalities. This is manifested in the term *hypermedia application* which refers to applications integrating both aspects (see sec. 2.1.4). While only few approaches exist which focus directly on multimedia applications there is a whole research community addressing systematic development of Web applications, called *Web Engineering*. Although several modeling approaches from Web Engineering use the term hypermedia application, this area clearly emphasizes on Web information systems. Nevertheless, due to their general relevance and adjacency to multimedia this section gives a short general overview on the typical concepts. Some exceptions which focus more on multimedia capabilities are provided in the next section, even if they arise from the Web Engineering community

The area of Web Engineering is part of the general research on the Web and its applications. The *World Wide Web Conference* (*WWW, [WWWb, Williamson et al.07]*) is one of the main conferences in this area. The systematic development of web applications is a specific sub-area within this community and referred to as *Web Engineering*. One of the first papers introducing this term was published in 1997 by Gellersen, Wicke and Gaedke [Gellersen et al.97]. First workshops followed in 1998 held on the *International Conference on Software Engineering* (*ICSE*, [ICS]) and the WWW conference. The area thus has also a strong background in Software Engineering. Web Engineering is devoted to the

> application of systematic, disciplined and quantifiable approaches to the cost-effective development and evolution of high-quality solutions in the World Wide Web. [Weba]

Modeling is one of the main topics in this area.

In contrast to user interface modeling, which has a much longer history, the number of existing approaches is more limited. Overview diagrams on the most established approaches can be found in [Schwinger and Koch03] and [Lang01b].

Figure 4.8 shows the overview from [Schwinger and Koch03]. Methods basing on Entity-Relationship Diagrams [Chen76] focus on database-oriented web applications. Examples are the *Relationship Management Methodology* (*RMM*, [Isakowitz et al.98]) and the *Web Modeling Language* (*WebML*, [Ceri et al.02]). Hypertext-oriented systems focus mainly on the hypertext character of web applications. Examples are the Web Site Design Method (*WSDM*, [Troyer and Decruyenaere00]) and the

Hypertext Design Model (*HDM*, [Garzotto et al.95]) which evolved to W2000 ([Baresi et al.01]) and HDM-lite ([Fraternali and Paolini98]). Object-Oriented methods base either on OMT [Rumbaugh et al.91] (one of the predecessors of UML) or UML. Examples are the Object-oriented Hypermedia Design Method (*OOHDM*, [Schwabe et al.02]), UML-based Web Engineering (*UWE*, [Koch et al.07]), and the Object-Oriented Hypermedia Method (*OO-H*, [Gómez et al.01]). Software-oriented methods treat web applications from the viewpoint of traditional Software Engineering, like the Web Application Extension (*WAE*, [Conallen00]) and its successor WAE2 [Conallen02].

The methods can be classified into four generations where each generation reused concepts from earlier approaches. Altogether, a convergence can be observed in this field and most approaches currently are either defined as UML extension or extended with a UML Profile to support compliance to UML [Schwinger and Koch03]. Moreover, the approaches are still enhanced and maintained and provide tool support, sometimes even as a commercial product (e.g. *WebRatio* [Webb, Acerbis et al.07], a tool for WebML).

Usually, a web application model specifies three different aspects: Content, Hypertext, and Presentation. Thereby, the models aim to address the specific characteristics of the web. The *Content model* corresponds to the application's domain model and can thus be modeled e.g. by UML class diagrams. Often, media types of documents are already considered here. Moreover, web applications often base on existing infrastructure, e.g. an existing database, which then has to be considered in the models. The *hypertext model*, often also referred to as *navigation model*, reflect the link and navigation structure of the application. It distinguishes between different kind of links like for internal navigation or for calling an external service. In addition, some approaches like OO-H support a pattern language for navigation patterns. The *Presentation Model* specifies the "Look and Feel" of the user interface and sometimes also its behavior.

Compared to user interface modeling approaches discussed in the section before, Web Engineering approaches are specific for Web applications. This means in particular, that in existing approaches the user interfaces are restricted to HTML which results in restricted interaction possibilities for the user interface. Moreover, they consider web-specific patterns which often are quite different from desktop applications. For example, many guidelines for web user interfaces regard the navigation bar, which does usually not exist in desktop applications. Due to this specific focus, most Web Engineering approaches aim to generate the complete application code from the models, except for the final user interface which is supposed to be implemented manually in some approaches.

Several approaches consider media components. Basically, proposals mainly enable to specify the media type of an element, like image or video to include a corresponding media object into the HTML user interface, like e.g. in [Hennicker and Koch01]. HDM, one of the earlier approaches (sometimes also cited in context of multimedia modeling [Zendler98]) allows defining different views on information artifacts, e.g. text and an alternative image. For description of media objects they use concepts from multimedia modeling for databases [Gibbs et al.94]. However, by their focus on HTML-based user interfaces the approaches support only a limited kind of user interaction and hence use media objects only as purely presentation elements. As examined in [Preciado et al.05], the multimedia support of these traditional Web Engineering approaches is limited.

Currently several research groups work toward extending the approaches for supporting also so-called *Rich Internet Applications* [Arteaga et al.06, Gruhn07], i.e. client-side web applications realized e.g. with AJAX, OpenLaslo or Flex. *RUX* [Linaje et al.07] aims to support the user interface aspect of RIAs and can be connected with existing Web Engineering approaches. A combination with WebML is presented in [Preciado et al.07]. The RIA modeling approach in [Martinez-Ruiz et al.06b] is based on UsiXML. Others mainly address certain aspects of RIAs, like the single-page paradigm and client-server communication, and apply them as extensions for WebML [Bozzon et al.06, Carughi et al.07] or OOHDM [Urbieta et al.07]. Although the emphasis certainly lies on widget-based user interfaces using Ajax and Flex and the specifics of Web applications, it still seems promising that such approaches will lead to an advanced multimedia support.

4.3 Multimedia Modelling

In the Multimedia Domain there is currently no established community which can be seen as equivalent of UI modeling and Web Engineering, focusing on systematic multimedia application development. Indeed, Multimedia as a whole has a well established community. One of the main events is the ACM Multimedia conference [Lienhart et al.07]. Most papers in multimedia community deal either with low-level techniques as base for multimedia applications, e.g. algorithms improving the performance of applications like a compression algorithm or techniques for semantic querying of multimedia data, or show new concrete applications. But they rarely address application development in terms of Software Engineering. Conferences focusing on "multimedia modeling" like the Multimedia Modeling Conference [MMM, Satoh et al.08] do not deal with modeling in terms of application development. Modeling here refers rather to modeling of concrete domain knowledge, like for example required for computer vision or semantic video concepts. In terms of model-driven development such kinds of models are domain models, but not suitable as metamodel for multimedia applications in general (which is of course even far away from their purpose).

Some increased research interest in systematic multimedia application development was postulated in the end of ninetees. Several researchers introduced the term *Multimedia Software Engineering* [Mühlhäuser96, Chang99], which refers to both: Using multimedia possibilities in Software Engineering, e.g. advanced code visualization techniques, as well as using Software Engineering Principles for multimedia development. This means that Multimedia Software Engineering claimed the general need for a better integration of multimedia and Software Engineering domain. However, these initiatives still have not really established yet.

Thus, still only very few approaches with similar goals like this work exist. As described in section 2.1, there is a number of contributions which provide multimedia modeling but are limited to a document-centric approach. A prominent examples is the Amsterdam Hypermedia Model (AHM, [Hardman et al.94, Hardman et al.97]) which revised the Dexter Hypertext Reference Model [Halasz and Schwartz94] and added multimedia properties like temporal and spatial layout. [Boll01] introduced context adaptivity to multimedia documents. An XML-based approach with similar features is Madeus [Villard et al.00]. Besides content adaptation, it also supports a very basic kind of interactivity by enabling to define "abstract devices" (e.g. abstraction of a mouse) which can trigger an event. [Tran-Thuong and Roisin03] provides a document model based on the MPEG-7 standard.

Beside the Hypermedia approaches which mainly focus on hypertext and Web application development, there are a few approaches which in fact provide multimedia support. The Hypermedia Modeling Technique (HMT, [Zoller01]) bases on concepts from RMM. It enables to define primitives like audio or slide shows and fine grained temporal relationships between them [Specht and Zoller00]. The example in figure 4.9 from [Specht and Zoller00] shows the HMT model of a webpage of an research association. The document homepage (in the center) shows the name, a logo an audio a header and a standard footer. A table of content leads to a page with information about associated research cooperations (left hand side). In addition, the names and logos of research cooperations are presented by a slide show. The temporal synchronization is specified in the lower part of the model: First the welcome-header, the slide-show and the audio are presented. Slide-show and audio are synchronized. After the slide show has finished, the remaining parts of the page (logo, name, footer, and list of

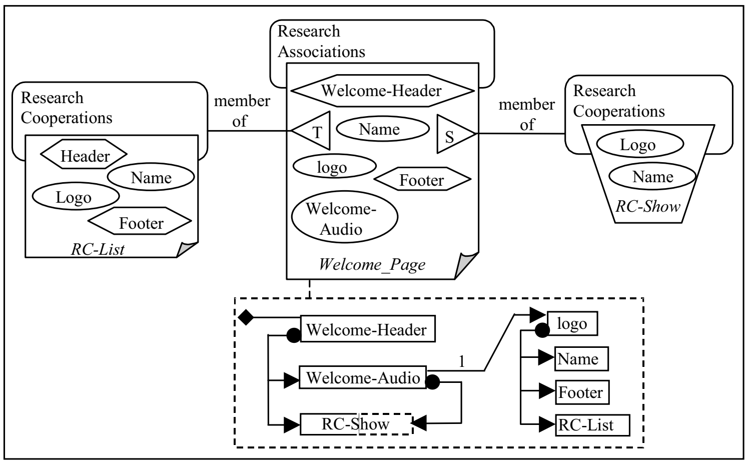


Figure 4.9: Example *Hypermedia Modeling Technique* (*HMT*) model from [Specht and Zoller00]

research cooperations) are presented.

4.3.1 HyDev

An approach even more considering specific features of multimedia applications is HyDev [Pauen et al.98a, Pauen and Voss98, Pauen et al.98b]. It aims to integrate both the document character and the software character of multimedia applications. HyDev is platform-independent and based on UML. It proposes three kinds of models: *domain model*, *instances model*, and *representation model*.

The domain model is used to model the application structure. It provides the conventional concepts of UML class diagrams. In addition, three multimedia-specific kinds of classes are available: *N-classes* used to model the application's narrative structure, *S-classes* which represent "‘spatial objects"’, and *A-classes* representing agents. All kinds of classes may own attributes and operations like conventional UML classes.

Figure 4.10 shows the domain model for an example application, a virtual museum. The application allows the user to take virtual tours on specific topics guided by a virtual museum guide.

The N-classes are used to model the application's narrative structure. An N-class represents a narrative unit, like scenes or episodes and is marked with an icon ≈. HyDev does not predefine kinds of narrative units , i.e. the modeler can define any kind of narrative unit required for the respective application. For this purpose several HyDev provides specific relationships between N-classes, like *sequence*, *simultaneity*, and *prerequisite-for*. For example in figure 4.10 the application has a narrative unit Tour which consists of a sequence of TourSegments which in turn consist of a sequence of Steps. Simultanously with each Step, CommentsOnExhibits are given.

The S-classes are marked by the icon ❐ and represent spatial objects. In HyDev, this means a 2D or 3D object, for example in a 3D virtual world. In one publication [Pauen et al.98a] they are also named as “physical objects” (but this does not mean real-world objects as used for augmented reality of tangible user interfaces). Specific kinds of relationships between them are *adjacent-relationship* and the *contained-in-relationship*. In the example, the virtual museum, its sections, and its rooms as well as the different kinds of exhibits are modeled as S-classes. A room has adjacent rooms and contains exhibits.

Furthermore, in addition to conventional class attributes and operations S-classes may own *behavior*, which means multimedia-related spatio-temporal behavior, like movements of an animation.

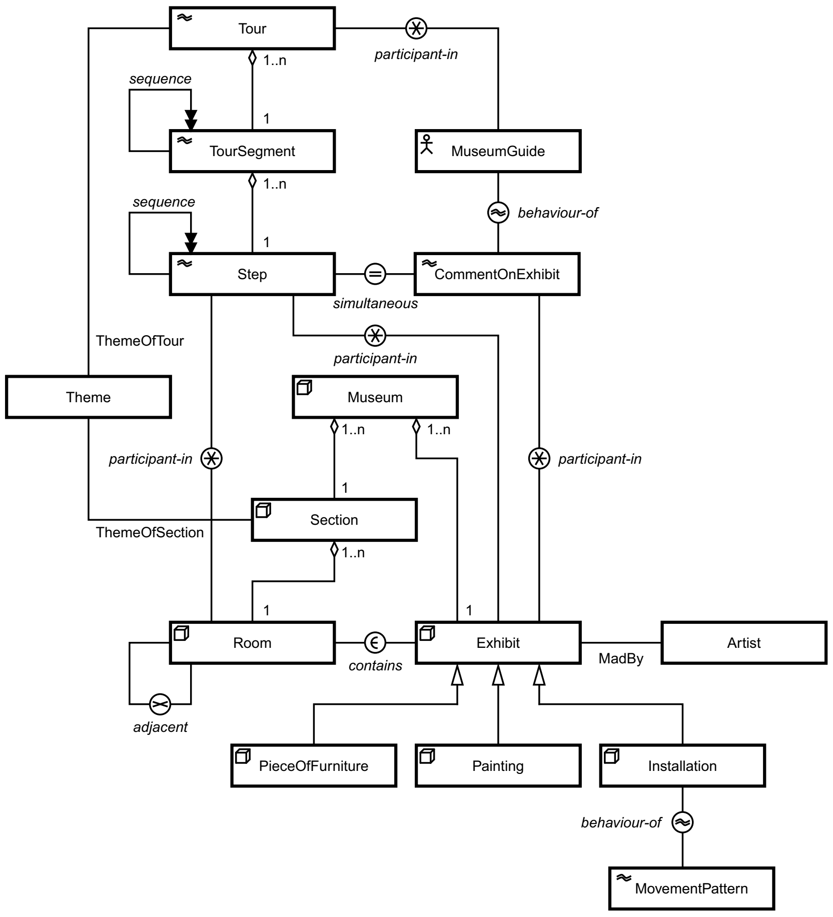


Figure 4.10: HyDev domain model for a virtual museum application from [Pauen et al.98a]

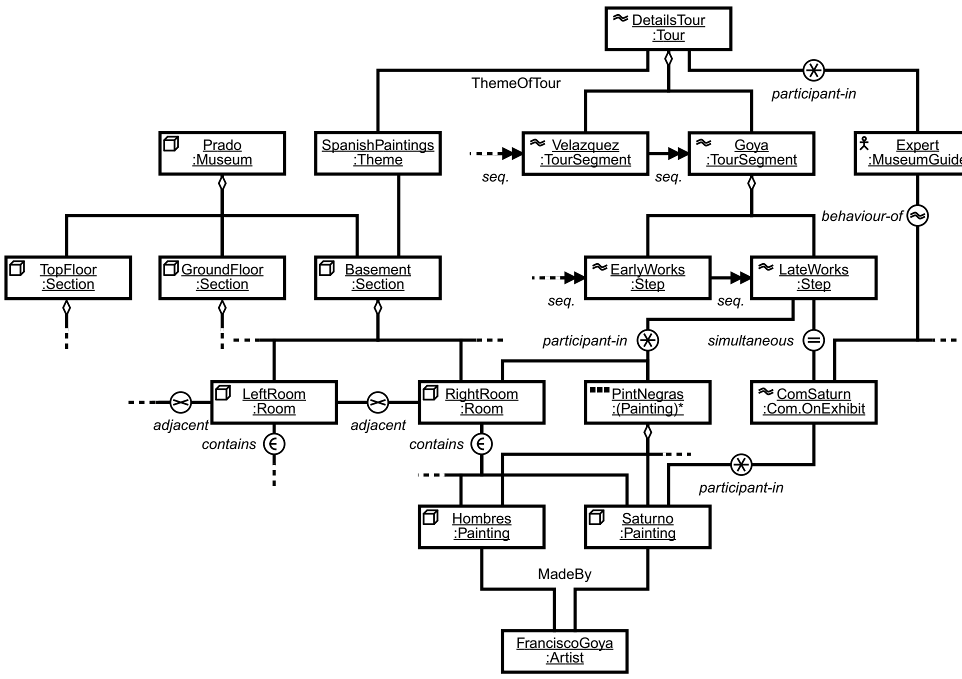


Figure 4.11: Extract from HyDev instance model for the virtual museum application taken from [Pauen et al.98a]

However, HyDev does not provide any further mechanisms to specify such kinds of behavior - it can only be specified informally using natural language [Pauen et al.98b].

The A-classes represent agents, which means elements "characterized by some kind of independence and autonomy." [Pauen et al.98a], like game characters or avatars. They always participate in narrative units. In the example application in figure 4.10 the museum guide is modeled as an A-class.

The instance model shows the instances of the running application. The authors argue that the concrete content of the application is an important information for the developer, e.g. which exhibits have to be developed for the virtual museum application. For this purpose, an object diagram is used, enhanced with the icons analogous to the domain model. An extract of the instance model for the virtual museum example is shown in figure 4.11. It is possible to aggregate several objects into collections, like for the series PintNegras in the example.

Finally, the representation model describes the object representation and the user interaction. It defines for the objects of the instance model how they should be represented on the user interface in terms of *representations*. Within an object's representation can be defined which attributes and relationships are represented by which media object. This includes the media type and a list of "'output media"', like audio channel, window, or external device. The representations can be nested which defines the overall structure of the user interface. Between media objects it is possible to define spatio-temporal relationships by qualitative statements like "'before"', "'after"' (temporal), or "'left"', "'right"' (spatial). Finally, the navigation between the representations is modeled by arrows annotated with events which trigger the navigation. Figure shows an extract of the representation model for the example application.

HyDev is a very interesting approach as it addresses many multimedia-specific properties of the application. Nevertheless, the choice of the models can be discussed. Modeling the application's objects in the instance model can often be a very tedious task. For example, if objects are taken

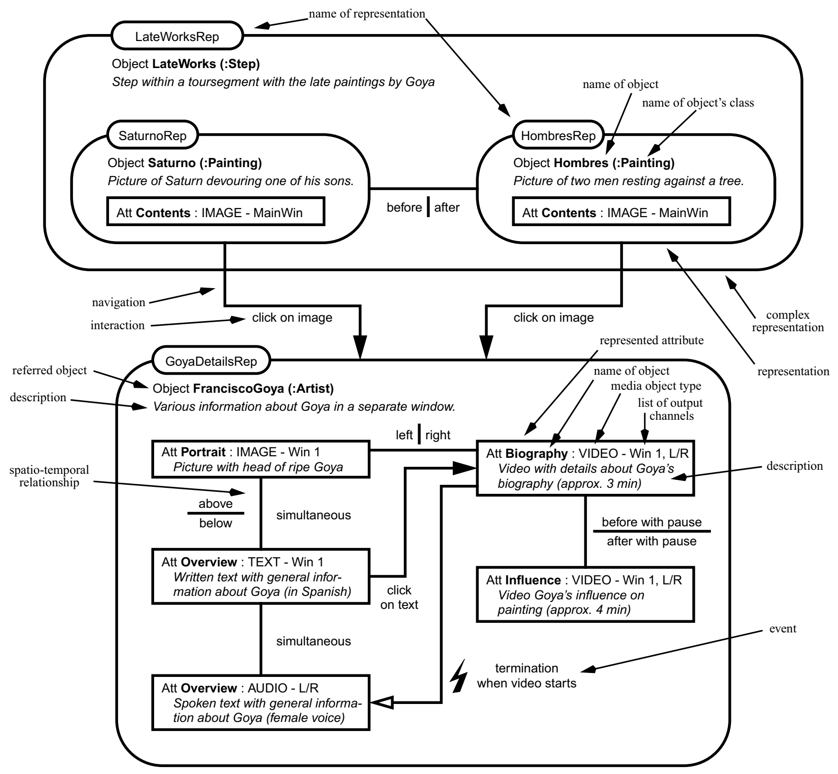

name of representation
LateWorksRep
Object LateWorks (:Step)
Step within a toursegment with the late paintings by Goya
name of object
name of object's class
SaturnoRep
Object Saturno (:Painting)
Picture of Saturn devouring one of his sons.
Att Contents : IMAGE - MainWin
before | after
HombresRep
Object Hombres (:Painting)
Picture of two men resting against a tree.
Att Contents : IMAGE - MainWin
navigation
interaction
click on image
click on image
complex representation
represented attribute
representation
GoyaDetailsRep
referred object
Object FranciscoGoya (:Artist)
description
Various information about Goya in a separate window.
name of object
media object type
list of output channels
left | right
Att Portrait : IMAGE - Win 1
Picture with head of ripe Goya
Att Biography : VIDEO - Win 1, L/R
Video with details about Goya's biography (approx. 3 min)
description
spatio-temporal relationship
above
below
simultaneous
before with pause
after with pause
Att Overview : TEXT - Win 1
Written text with general information about Goya (in Spanish)
click on text
Att Influence : VIDEO - Win 1, L/R
Video Goya's influence on painting (approx. 4 min)
event
simultaneous
termination when video starts
Att Overview : AUDIO - L/R
Spoken text with general information about Goya (female voice)

Figure 4.12: Extract from HyDev representation model for the virtual museum application taken from [Pauen et al.98a]

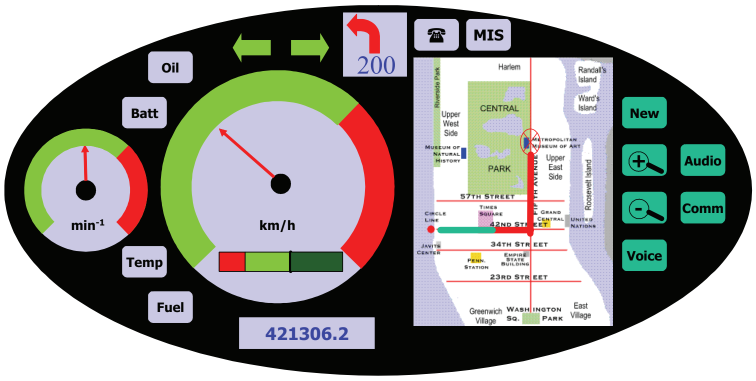


Figure 4.13: Screenshot from the automotive information system example for OMMMA [Engels and Sauer02]

from a database, creating the instance model can result in modeling whole database tables. On the other hand, the benefit of the instance model can often be limited, as the concrete object's names and attributes are not always meaningful in multimedia applications, as objects differ mainly in their visual representation. For example in a gaming application, different levels often have only the name "‘level 1"’, "‘level 2"’, etc. and differ only in the concrete spatial arrangement of objects on the user interface. On the other hand, the representation model tends to include too much information. It contains structure and behavior of the user interface and at different levels of granularity. Probably, when modeling a whole application, it can become very difficult to handle.

In its current form, code-generation is not supported by HyDev. Many of the language elements, e.g. agents and spatial objects, are not defined precisely enough. Others, e.g. behavior of S-classes and events in the representation model, are modeled just by textual descriptions. However, HyDev provides a worthwhile example in which direction multimedia-specific modeling should investigate.

4.3.2 OMMMA

Another platform-independent modeling language for multimedia applications is provided by the *Object Oriented Modeling of Multimedia Applications* (*OMMMA*, [Engels and Sauer02, Sauer and Engels01, Sauer and Engels99a]) approach by Sauer and Engels. It extends UML using Stereotypes and supports four kinds of diagrams: an extended class diagram for the *application and media structure*, an extended sequence diagram for *temporal behavior*, a presentation diagram for *spatial aspects of the presentation*, and a state chart diagram for the *interactive control*.

The OMMMA diagrams are explained in the following using an example application given in [Engels and Sauer02]: a (simulation) application of an automotive information system that provides the user control over the car's navigation and entertainment functionality via a multimedia user interface. It includes car audio, navigation and communication systems, travel or tourist information, and automotive system monitoring and control. Figure 4.13 shows a screenshot from its user interface.

The class diagram is used to model the application structure. Basically, it provides the conventional class diagram elements, like classes and relationships between them. It is divided into two parts: a hierarchy of media type definitions and a domain model describing the application logic. Figure 4.14 shows the class diagram for the example application. The media type hierarchy defines the media types to be used and is derived from existing multimedia standards and frameworks. In the example, it is located on the bottom part of the diagram. The upper part shows the domain model for

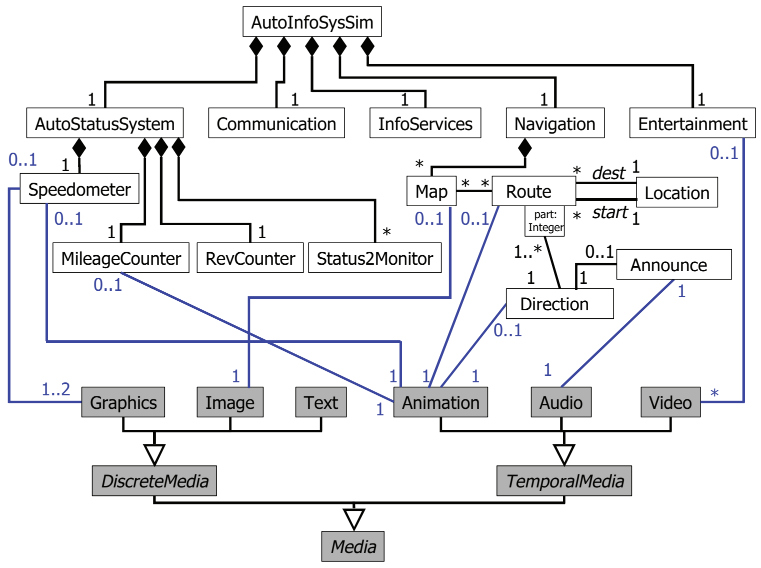


Figure 4.14: OMMMA class diagram from [Engels and Sauer02] for the example automotive information system

the example application. Here it is modeled as an composition of five subsystems for the different functionalities AutoStatusSystem, Communication, InfoServices, Navigation, and Entertainment. For some of them, some further example domain classes are shown.

The associations between elements from the two parts specify that a domain class shall be presented in the application by a respective media type. For example, the speedometer should be presented by an animation and one or two graphics, a map should be presented by an image, etc. In this way – using relationships between domain classes and media types instead of defining the domain class itself as media object – one domain class can be flexibly presented by multiple media objects.

The OMMMA class diagram in addition (not shown in the example) contains a *signal hierarchy* as base for the event-based interaction and, possibly in a separate package, *presentation classes* to specify the possible composition of user interfaces as a base for the presentation diagram.

Extended UML sequence diagrams are used in OMMMA to model the predefined temporal behavior. It should be mentioned that when OMMMA was published UML 1.3 was the current version of UML so it does not consider the additonal concepts of UML sequence diagrams in UML2.

The objects in the horizontal dimensions are instances from the domain model. Like in conventional UML sequence diagrams they have a lifeline in vertical direction (dashed lines). Horizontal arrows indicate a message sent between objects. An activation box on the lifeline indicates that the element has become active as result of a message.

For the purpose of modeling synchronization OMMMA provides several extensions. Figure 4.15 shows an example from the automotive system. In OMMMA a sequence diagram represents a scenario, which is specified by the initial message from an actor, e.g. a user interface component which triggers the behavior. The example shows the scenario that the navigation system presents the user a route from A to B. The initial message is showRoute(A, B). The lifelines in OMMMA represent local timelines and can be related to the actor's timeline which represent the global time. It is possible to define durations and points in time in several ways using time intervals and constraints.

Activations can be annotated with names of presentation elements, i.e. media objects (denoted with <>) or elements from the presentation diagram. In the example, the Navigation instance first

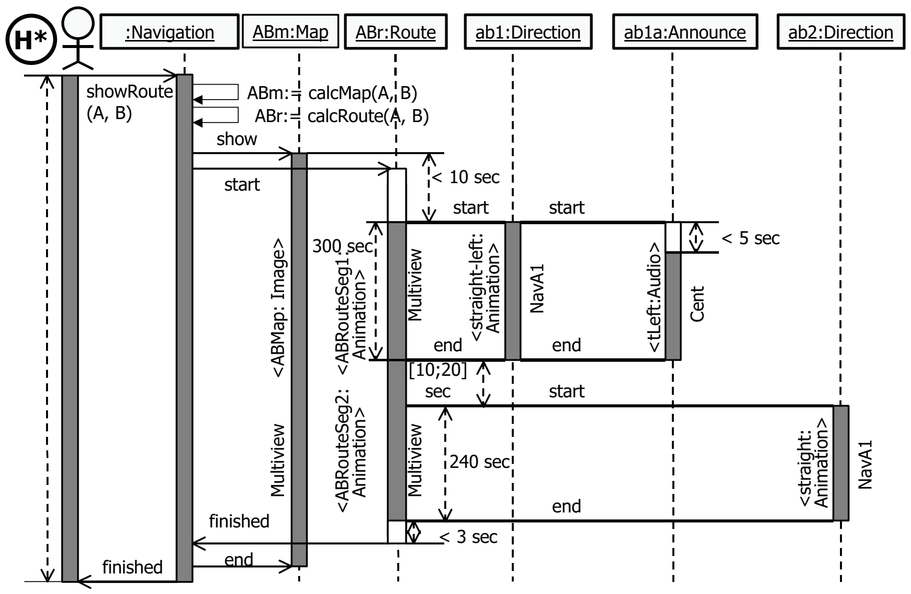


Figure 4.15: OMMMA sequence diagram from [Engels and Sauer02] for the example automotive information system

calculates the map and the route and then sends the message show to the Map instance. The map instance is presented by its associated Image object ABMap which is presented in the Multiview object from the presentation diagram.

Bold lines denote synchronization bars indicating that two or more objects should be synchronized. In the example, the Navigation instance then sends another message to the Route instance which should be synchronized with the Map presentation. Its activation box is not filled at the beginning which indicates an *activation delay* used to model tolerated variance of synchronization relations. The temporal constraint specifies that the presentation of the route must start at latest 10 seconds after the presentation of the Map instance.

It is also possible to specify *parallely composed activation* of media objects to model the simultaneous presentation using different or presentation elements. In the example, the Route instance is presented by its associated animation and in parallel the direction is presented and an announcement is performed. Sequentially composed activations is used to specify that after the ABRouteSeg1 another animation ABRouteSeg2 is shown. Finally, it is possible to overlay an activation with *media filters* which are temporal functions, e.g. the increase of the audio level over the time.

For each sequence diagram the *History* concept from UML statecharts can be used to specify to which extent it is possible to resume a behavior after an interruption. Deep history, denoted with H^*, means that the behavior can be resumed exactly in the same state before the interruption occurred. Shallow history (H) specifies that returning is only possible on the top-level.

The presentation diagram specifies the spatial structure of the user interface. It shows the elements on the user interface in terms of bounding boxes. The diagram visually defines their size and layout according to a specified coordinate system. There are two kinds of user interface elements in OMMMA: *visualization objects* are passive output objects which present some information to the user while *interaction objects* allow user interaction and trigger events. The latter ones are denoted by bold boxes.

Presentation diagrams can be split into different areas representing different hardware devices, like

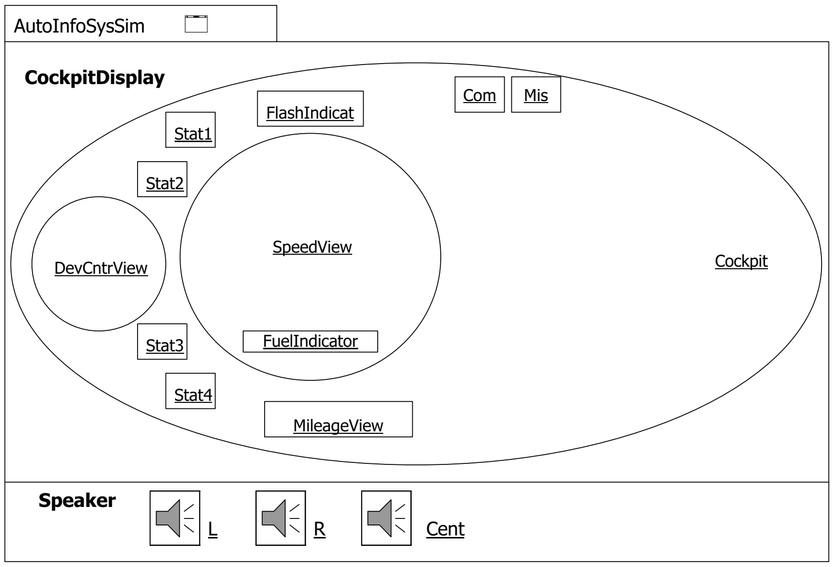


Figure 4.16: OMMMA presentation diagram from [Engels and Sauer02] for the top level view.

a screen and audio speakers. Figure 4.16 shows the presentation diagram for the example application's top level view (AutoInfoSysSim). The bottom area represents audio speakers (Speaker) which can be referred e.g. in the sequence diagrams. The top area presents the display (CockpitDisplay) containing bounding boxes for the visual user interface elements. In the example the display contains only visualization objects as the user input is performed via specific hardware devices like knobs. Those could be specified in an additional diagram compartment analogously to the speaker.

The user interface can be composed of different views which can be placed on different layers on the virtual area. For example figure 4.17 shows the content of the Cockpit element from figure Figure 4.16.

The statechart diagram in OMMMA describes the interactive control and the dynamic behavior of the system. Therefore it specifies the different states of the systems and the transitions between them which are triggered by events. It uses the standard UML statechart constructs. The events which trigger the transitions correspond to the signal defined in the signal hierarchy in the class diagram. As the class diagram is defined by the modeler, any kind of events are supported, including user events, system events, or timer events.

Figure 4.18 shows the statechart diagram for the top level of the example application. It uses advanced UML statechart concepts like composite states and submachine states. Complex composite states, like Navigation, InfoServices, and Entertainment are specified in additional diagrams.

When the system enters a (simple) state it executes associated predefined behavior specified in the sequence diagrams. For this purpose the initial message of the corresponding sequence diagrams is specified in the Do-section of the state. For example, one of the substates of the Navigation state (not shown in the diagram) performs the message showRoute(A,B) which triggers the predefined behavior specified in figure 4.15.

Altogether, OMMMA seems to be the most elaborated approach for modeling interactive multimedia applications. It covers the different aspects of multimedia applications and integrates them into a consistent approach. It thus provides an important contribution for all further research in this area. Nevertheless, a more in-depth analysis shows that there are still various aspects which are not covered by OMMMA yet. Also, the modeling concepts in OMMMA are not sufficient to fulfill the goals iden-

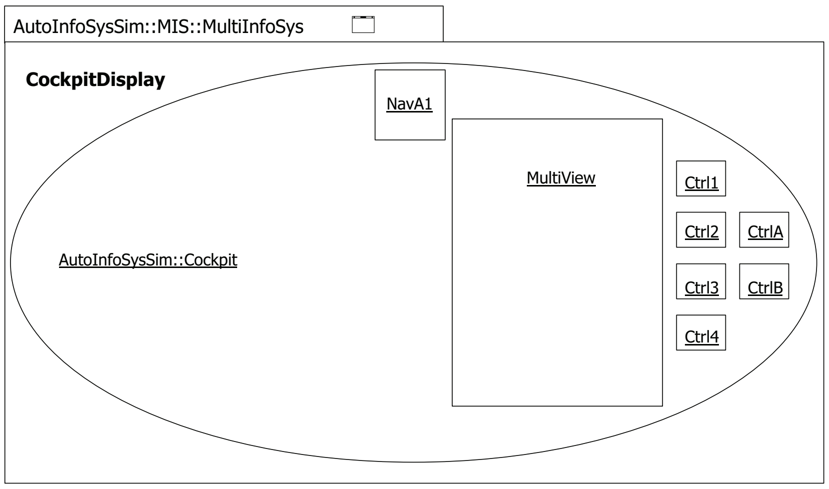


Figure 4.17: OMMMA presentation diagram from [Engels and Sauer02] for the cockpit view.

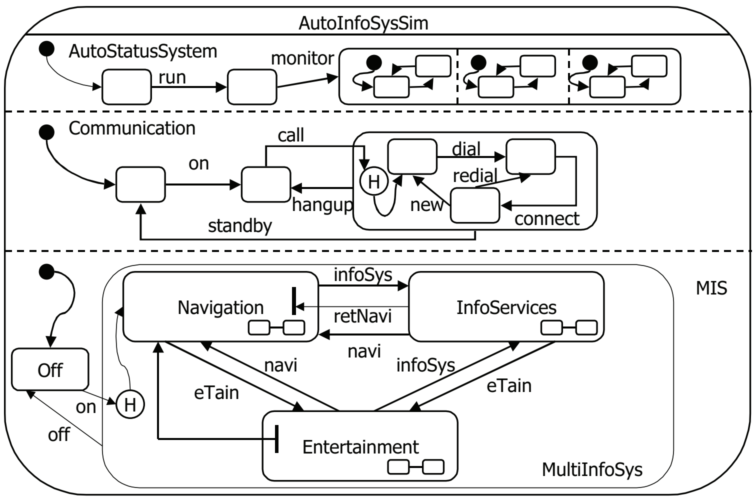


Figure 4.18: OMMMA statechart diagram from [Engels and Sauer02] for the example automotive information system

tified in this work, like an easy usable and model-driven approach. Some important shortcomings are discussed in the following:

As a first issue, several parts of the OMMMA language are too generic to enable a clear model-driven process. Several elements, like user interface elements and signals for interaction, have to be defined in the class diagram by the modeler. Predefining them would provide better support for the modeler and would also also be necessary for code generation. In its current form, there is also poor support for the modeler how to structure the overall model in a consistent way. For example, which classes from the class diagram correspond to the top-level of the statechart diagram? Some of the initial contributions [Bertram et al.99, Sauer and Engels99b] propose to explicitly specify one class as the application's top-level class labeled with the stereotype Multimedia Application which is composed of several scene classes labeled with the stereotype Scene (or scenario in [Sauer and Engels01]). A scene then represents an independent part of the application associated with domain classes, a presentation and a top-level state in the statechart diagram. In figure 4.14 the class AutoInfoSysSim would correspond to the Multimedia Application and the classes AutoStatusSystem, Communication, InfoServices, Navigation, and Entertainment to Scenes. However, currently this is not further defined. Besides, as OMMMA aims to be specified as a UML profile, such relationships between the model elements (e.g. also between thestatecharts and sequence diagrams) currently have mainly the character of conventions and can not be directly supported by modeling tools.

A second issue is the usability of the diagrams for the modeler. OMMMA emphasizes modeling the application's predefined behavior. Therefore it uses UML sequence diagrams and various extensions. However, as section 6.5 will discuss in more detail, it is questionable whether such a fine grained definition of durations and time values is frequently necessary in an abstract model for multimedia applications. In OMMMA, a sequence diagram specifies only one predefined behavior without any interaction. Often the contained information will be of limited value for the modeler as the predefined behavior is on the one hand trivial and on the other hand specification of exact time values is not required during application design. In turn, the statechart diagram contains a very high amount of information as it contains the application's complete navigation and interaction. In interactive applications the statecharts becomes very complex as already indicated by the extract shown in figure 4.18.

As third issue, OMMMA covers the user interface design only partially. The presentation diagram focuses on the concrete spatial layout. There are neither predefined user interface elements nor is there a notation to visually distinguish between different custom elements. The purpose of elements can often be derived only by analyzing the statechart and the sequence diagrams. Moreover, it seems not to be intended that media objects act as interactive elements as well.

Finally, the media objects in OMMMA are very simple model elements without any properties. Thus, the information about the media objects which can be expressed by the models is very limited. Let's consider for example figure 4.14:

- *Speedometer* has a relationship with *Graphics* with the multiplicity '1..2'. What is the purpose of this graphics and why are two graphics required? The textual description in [Engels and Sauer02] explains that the graphics are two alternative background graphics "e.g. to enable a day and night design of the background". However, this information is not reflected in the model. MML solves this issue by modeling each media object as an individual, named model element like NightBackgound and DaylightBackground (section 5.2.2).
- According to the textual description in [Engels and Sauer02] the speedometer consists of two graphics for day and night design of the background and an animated indicator for the actual speed. This means that the background and the indicator animation must fit together and build

the overall speedometer representation. MML solves this by providing support to model the inner structure of media objects. For instance, it is possible to model an animation Speedometer which contains as inner objects a graphic background and an animation speedIndicator (section 5.2.8).

- The Entertainment class in figure 4.14 is related to videos. Obviously, the user must be able to play, pause, stop, etc. the videos. Is such functionality already part of a video object (and if so, what kind of functionality) or must it be specified by the modeler? MML solves this issue by the concept of Media Components (section 5.2.2).
- Some media objects must be created by the media designer, like the speedometer. Others, like the videos for the Entertainment class, are possibly loaded dynamically into the application, e.g. from the user's personal video collection, and need not to be created by the media designer. There is no way to distinguish such different cases in OMMMA. MML solves this by the possibility to specify concrete artifacts of Media Components and by additional keywords (section 5.2.5 and 5.2.6).

These are just some examples showing that the simple way to model media objects in OMMMA is not sufficient to capture all information necessary for the media designer. Useful code generation from the models would require even more detailed information. A detailed discussion on issues and solutions for modeling media objects in interactive applications is given in chapter 5.2.

OMMMA does currently not provide a modeling process or code generation. However, there are several contributions into this direction regarding the extended sequence diagrams. [Engels et al.00] specifies an approach for formal specification of behavioral UML diagrams using collaboration diagrams which are interpreted as graph grammar rules. On that base, [Hausmann et al.01] shows how this approach can be extended to UML extensions like UML profiles and applies it as example to extensions for UML sequence diagrams. [Hausmann et al.04] then extends this example for specifying temporal properties for UML sequence diagrams, as used in OMMMA, and provides an interpreter to analyze or test such models.

4.3.3 Code Generation

This section discusses existing proposals directed towards automatic or manual derivation of code from design models. The majority of them addresses non-interactive multimedia presentations (e.g. [André95]) and they mostly address specific domains. An example is the discourse driven approach in [Morris and Finkelstein96] which aims to generate multimedia documents for Software Engineering tool demonstrations. An overview which classifies approaches according to their target domain – like report generation, route directions or education – can be found in [André00].

Code generation for more complex, interactive applications is supported only by approaches from User Interface Modeling or Web Engineering as explained in the foregoing sections, which provide only limited multimedia support and focus on conventional widget-based user interfaces. In particular, the idea of combining a systematic modeling approach and multimedia authoring tools has rarely been investigated yet. This section describes two approaches which can be considered as very first steps into this direction.

Boles, Dawabi, and Schlattmann [Boles et al.98] introduce such an approach for the domain of virtual labs. Their example is a virtual genetic engineering lab application which shows how to set up and conduct different experiments. As modeling language they use plain UML. UML Class diagrams specify the application's domain classes. A sequence diagram describes the experiment in terms of messages between domain classes. Finally, simple UML statecharts specify the domain classes' different states.

The authoring tool *Director* is used for the implementation. The authors provide a proposal how to implement the UML design models within the tool. The class diagram is mapped to Lingo class constructs. The statecharts are implemented in the class methods by setting attribute values. Moreover, they sketch how to structure the remaining application parts in terms of the Model-View-Controller pattern. Automatic code generation or a generalization of the approach beyond the scope of virtual labs seems not to be intended by the authors.

A second approach using UML models and the multimedia authoring tool *Director* is described by Depke, Engels, Mehner, Sauer, and Wagner [Depke et al.99], the same research group which investigated in the OMMMA approach described above. They use several different UML class and object diagrams to support the development process from platform-independent design toward the platform-specific implementation in the authoring tool. For the platform-independent modeling they provide a general *application model* in terms of a class diagram. It shows the general structure of a multimedia learning application including classes for the media types, presentation, control, and a starting point for the application logic. The application logic initially contains only a simple basic structure for learning applications, i.e. learning units and relationships between them. When developing a concrete application the application logic has to be extended to reflect the logic of the concrete application in terms of the concrete learning content. The other parts of the application model (media types, presentation, control) remain usually unchanged, i.e. these parts are a kind of general framework to be used in multiple projects.

For the authoring tool, they provide an the authoring tool model in terms of a class diagram. It defines the general structure of *Director* applications. It has to be defined only once as well. In addition, the authors provide mapping rules for mapping instances of the application model classes onto instances onto authoring tool model classes.

When developing a concrete application the developer first extends the application model with application classes specific for the current application. Then the developer creates an object diagram of the application by instantiating the classes from the application model. By application of the mapping rules an object diagram is derived which contains instances of the authoring tool model. It can finally be implemented in the authoring tool.

Interestingly, the proposal has many parallels with a model-driven development process (figure 4.19). However, it resides one meta-level below: instead of meta-models it uses class diagrams for the general concepts which are instantiated for the concrete application. As a consequence, concepts on class level (as required for example for the domain classes) must be specified by extending the generic application model, e.g. by specifying subclasses.

As the authors state, the purpose of this article is not to provide complete generic models and mappings but rather to demonstrate the process. They also restrict themselves to the static application structure and conventional UML class and object diagrams. The mappings are intended to be performed manually. Nevertheless, the proposal can be interpreted as a first systematic integration of modeling and authoring tools and, moreover, includes ideas of model-driven development.

4.4 Discussion

In summary, a large variety of approaches related to multimedia modeling exists. But none of them can provide sufficient support for highly interactive applications using non-standard user interface elements.

The area of user interface modeling has a long history. The number of approaches is thus very large and proposals are very sophisticated. However, the area addresses user interfaces in general

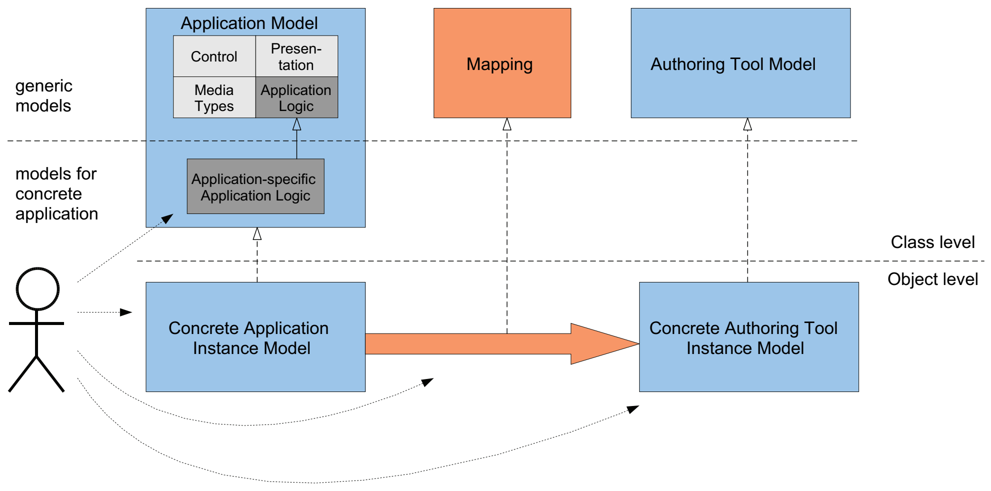

Application Model
Control
Presen-tation
Media Types
Application Logic
Mapping
Authoring Tool Model
generic models
models for concrete application
Application-specific Application Logic
Class level
Object level
Concrete Application Instance Model
Concrete Authoring Tool Instance Model

Figure 4.19: Approach from [Depke et al.99] interpreted as kind of MDE approach.

which results in standard widget-based user interfaces. Multimedia aspects are not further considered. However, many of the established concepts from this area can also be applied to multimedia applications and are thus carefully considered in this work.

The area of Web Engineering provides approaches which allow a high degree of automation and can be applied in commercial projects. However, this is caused by their restriction to common web-specific applications and standard HTML-based user interfaces. Research on Rich Internet applications, providing a higher degree of interaction, has just started in the last few years. However, these latest efforts show that modeling multimedia applications is becoming more and more relevant.

Finally, the research area of Multimedia itself hardly targets the application development aspects. Various existing proposals cover multimedia data and multimedia documents very well but they are not extended towards interactive applications. From the few remaining approaches, OMMMA clearly seems to be the most elaborated one. As discussed above, various issues in OMMMA do not satisfy the goals for the current work. For example, it does not consider concepts from user interface modeling and also does not provide any manual or automatic transformations towards implementation yet. Nevertheless, it provides several important basic contributions and is carefully considered in this work.

The approach presented in the following integrates the relevant existing concepts from multimedia modeling and user interface modeling. As general foundation it uses the state-of-the-art concepts from model-driven engineering. To gain feedback from these three important areas, MML has been presented on conferences in all three communities: General foundations of MML and the overall framework [Pleuß05b] have been presented on the MODELS conference. The MML modeling language [Pleuß05a] has been presented on the International Symposium of Multimedia. A summary on MML and the integration of authoring tools [Pleuß and Hußmann07] has been presented on the special session on "Patterns and Models for the Development of Interactive Systems" at the HCI International conference.

Chapter 5 will identify several important features of highly interactive multimedia applications which can not be modeled with the existing work so far. Later on, section 8.3.3 will summarize these new features and use them as base for a detailed comparison of selected approaches and MML.

Finally, the approach presented here proposes and implements the integration of an existing professional multimedia authoring tool and a model-driven approach. Such an integration has not really been considered in existing work yet. As discussed in section 9, generalizing this idea might contribute to a better integration of systematic Software Engineering concepts and creative design in general.

Chapter 5

Modeling Multimedia – MML Core Principles

The following three chapters present the *Multimedia Modeling Language* (*MML*) as main contribution of this thesis. The current chapter discusses basic decisions on language definition and introduces core modeling concepts in detail. Afterwards chapter 6 provides a more compact overview on the overall modeling language, its different diagrams, and the process to create MML models. Finally, chapter 7 shows by the example of Flash how MML models are mapped to code and how visual multimedia authoring tools are tightly integrated into development with MML.

The current chapter is structured as follows: The first section explains basic issues on modeling language definition and resulting decisions made for the definition of MML. this includes a short excursion on icon design for modeling languages which is necessary to understand how icons for MML model elements have been developed. On that base the second chapter introduces in detail the concept of *Media Component* as core concept for modeling multimedia and discusses it by various examples.

5.1 Basic Decisions and Language Engineering

In chapter 3 the overall goals for the modeling language have been elaborated. It should be easy to use, support a design phase and transformation into code for multimedia authoring tools, and integrate multimedia design, software design, and user interface design. Realizing such a modeling language requires some more detailed decisions about language design and definition. Currently, there is little literature on such practical issues like systematic design of abstract and concrete syntax of a modeling language or patterns to be used in metamodels. Latest books like [Kelly and Tolvanen08] address such issues as well as, in a general scope, emerging initiatives on *Language Engineering* like the *ATEM* workshop in 2007 [ATE].

The following sections do not claim to provide a generic overview on language design but discuss such issues as far as they are important for the design of MML. This includes basic decisions about MML like the language scope, UML-compliant definition, and some (very basic) applied metamodeling principles. In particular it provides a short excursion about systematic icon design for visual modeling languages which has only marginally been addressed by literature yet and is necessary to understand how the icons for MML model elements have been developed. In this way the section provides a summary of basic language engineering issues which turned out during the development of MML.

5.1.1 Scope

MML should support multimedia applications, as defined in section 2.5. It should address such applications in general, i.e. not devoted to a specific application domain. This is reasonable as concepts of multimedia applications (section 2.5) and implementation technologies (section 2.3) can basically be applied in any kind of application and any domain (see 2.4). From that point of view MML can be judged as a General Purpose Language.

For the definition of Domain Specific Languages it is often useful to use terms and modeling concepts tailored to the specific audience. This can even be a single company or development team. In contrast, a more general language like MML must be accessible for a broader audience. Thus, it is useful to reuse as much as possible existing common modeling concepts developers might already be familiar with; i.e. mainly the de-facto standard UML. Reusing established modeling concepts also ensures the quality of the resulting language.

Two aspects of multimedia applications are already supported by existing modeling languages: the application logic and the general aspects of the user interface. For the application logic, UML is already established as a standard. It seems useful to use it for MML as well. In the area of user interface modeling there is currently no such standard but many general concepts exist (sec. 4.1) which should be reused in MML.

These two existing areas (modeling of application logic and user interface modeling) provide in addition some advanced approaches for modeling specific features. For example, several approaches from the user interface modeling area support context-sensitive user interfaces (section 4.1) or physical user interface objects [Gauffre et al.07]. UML-based approaches support various application features like databases or real-time constraints. As MML aims to support applications in general, the question arises whether such aspects must be integrated into MML as well. Of course, a multimedia application can be context-sensitive and use physical user interface objects, include a database, and have real-time features. Furthermore, the area of web applications provides for example concepts for modeling client server communication which can also be relevant for multimedia applications. However, it seems very hard to combine all such aspects into a single modeling language. The resulting language then would become very large and hard to handle. It would also require to select the "best of" the existing proposals. Such a unified language is specified better by a consortium like the OMG than by single research efforts. Instead it is much more promising to focus on the core concepts for multimedia applications and define them in a way that still allows to extend or combine them with other approaches later if needed (see also discussion in sec. 10.3).

Consequently, MML focuses only on the core concepts of multimedia applications and is designed in a way that it can easily combined or extended with other modeling approaches. The core concepts are those required to model a multimedia application in general – without any specific features beside "multimedia" itself – so that it is possible to generate useful code for multimedia-specific implementation technologies like Flash.

5.1.2 Language Definition

As discussed above MML should be compliant to the UML. Section 3.4 describes three different alternatives how to define a standard compliant modeling language: as independent metamodel, as metamodel extending the UML metamodel or as UML Profile.

The application logic in MML can be modeled using UML class diagrams. Other aspects of multimedia applications, like media objects or the user interface, are not supported by UML. However, some of them can be denoted using one of the UML general modeling concepts, like state charts used for modeling the dialogue in user interface modeling. It is valuable to prefer established existing concepts but only as long as they can be applied properly. Thus MML aims to reuse UML concepts where this does not lead to any drawbacks, and introduces new customized concepts otherwise.

Based on these considerations, it seems useful to define MML either as UML Profile or at least based on the UML metamodel. However, MML is supposed to act as conceptual model for multimedia applications. Furthermore, potential reuse of UML concepts should not at all prevent the design from selecting the optimal solution for each concept in MML. Thus, MML is defined as an own metamodel which partially reuses the UML metamodel but defines a customized and independent conceptual model of multimedia applications. In a second step, it might then – if required – still possible to additionally define it as UML Profile later, e.g. to reuse a UML modeling tool as a solution for modeling tool support (see section 6.7).

5.1.3 Notation

As a consequence from the decisions above, the language's notation partially reuses the UML notations. Model elements which are directly adopted from UML should obviously keep their UML notation. For other elements different possibilities exist: 1) just to apply an existing UML notation for them, 2) to mark them by an additional keyword (which is in UML denoted in guillemets «» analogous to stereotypes, see section 3.4) or 3) to define a completely new notation for them, either using icons (analogous to stereotype icons) or even by more complex graphical elements.

The advantage of the first case is that visual primitives and components already established in modeling can be reused. As UML is a large modeling language, many preferable notations are already used by UML model elements. Reusing them can lead to diagrams with easy to use and already approved notations. The disadvantage is that such elements can be misinterpreted as UML elements. Thus, reuse of notations is only useful when either the custom element is a specialization of the UML element and no distinction is required (e.g. because the UML element is not part of the custom language) and/or if the difference becomes clear from the context where the notation can be used. An example is reusing the notation for UML states in a diagram modeling the user interface dialogue.

In particular, for the notation of relationships only a limited number of possible notations exists as the main difference between notations results only from line style and line ends. When using too many different notations for relationships they can easily become difficult to distinguish or when using too complex solutions, difficult to draw (ideally, it should also be possible to sketch diagrams by hand). On the other hand, relationships can often be understood just by their context, i.e. the kind of elements they connect. Thus, for relationships in many cases the UML notation can be reused.

The advantage of the second case is that adding a keyword to the notation unambiguously defines the kind of element. The disadvantage is that when using too many elements with keywords the visual character of the notation can get lost and the modeler has to read all the keywords in a diagram to distinguish the elements.

The third possibility allows to create a meaningful notation for custom elements which allows a clear distinction from UML elements. But it can be difficult to find a notation which is easy to handle and can be easily understood and recognized by different people.

MML uses all three possibilities according to their advantages and disadvantages. The notation of model elements reused from UML remains unchanged. Elements which can be seen as specialization of UML elements are denoted using the respective UML notation as well. Ideally, modeling tools should allow to optionally show and suppress additional keywords. Relationships are denoted using the UML notation as well and can be identified due to their context. For important elements not supported by UML, like abstract user interface elements and media components, a custom visual notation is provided.

5.1.4 Custom Icon Design

As custom icon design is not a trivial task and as no common way for systematic icon design in the modeling community exists, this section provides a short excursion on this topic and finally explains how the custom icons for MML (shown during the introduction of MML modeling elements in the following sections) have been developed.

The goal of a custom notation is usually to enhance the usability of the modeling language, e.g. to allow easier learning, recognizing, and understanding of modeling elements and thus finally increase the efficiency when using the language (see e.g. [Moyes and Jordan93] for advantages and properties of icons). However, developing new icons can be a difficult task, in particular as software engineers often have little knowledge in graphic design. This section shows user testing methods for icons and how they were applied for MML.

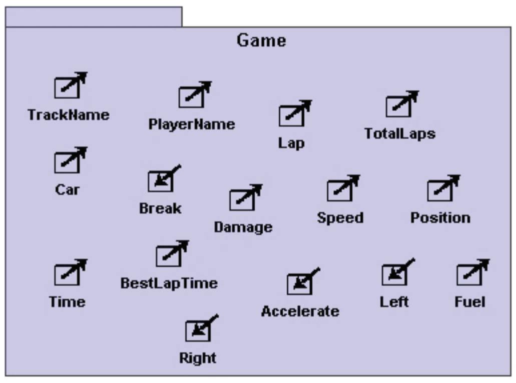
Kelly and Tolvanen provide several basic guidelines for symbol definition ([Kelly and Tolvanen08], pp.259). However, it remains a problem how to find appropriate visual representations and the detailed visual design. For these questions design principles have to be considered. The design handbook [Stankowski and Duschek94] addresses the design of pictograms where icons are usually seen as a subclass from. Pictograms represent an object, concept or function by an visual representation. [Stankowski and Duschek94] describes pictogram design as a process with various steps which includes to identify possible representations, simplify and objectify them, and finally humanize them again so that the representation is not too abstract and still likeable by the user. A typical problem is finding a representation for abstract concepts where no direct visual representation exists. Beside the representation and the proper level of abstraction, the shape, color and layout details are important just as well.

A specific property of icons is that they mostly occur as a set of related icons which results in additional challenges. It should be apparent for the user that the icons of a set belong together. They should be consistent, logically related and perhaps even allow to compose more complex icons from the basic ones. On the other hand the contrast between them must be large enough to easily recognize and identify different icons. This trade-off is illustrated in figure 5.1: It shows two alternative notations for modeling an abstract user interface (see sec. 4.1.1) containing several Output Components and a few Input Components.

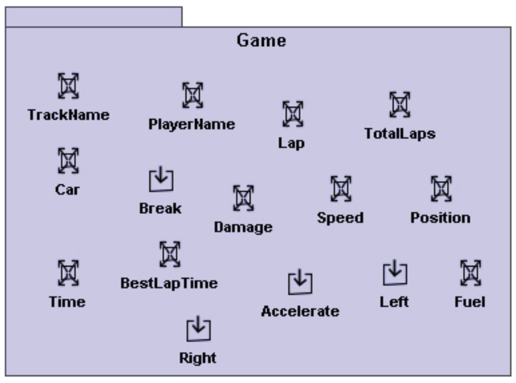
Figure 5.1a shows a notation oriented at [Van den Bergh and Coninx05] where the set of icons is very consistent and logically related. However, it is difficult to quickly find the Input Components within the diagram (for example if a developer wants to look up how many Input Components to implement). The alternative notation in figure 5.1b is less consistent but provides higher contrast enabling to identify the Input Components at first look. (This notation has been elaborated for MML by the user tests described below.)

The preferable way in such a situation is, according to the principles of human-computer interaction, to perform user tests with different icon sets to find out the preferred solution and ensure quality of designed icons. [Misanchuk et al.00] propose four kinds of tests specifically useful for icons:

Appropriateness test "is conducted to determine which icons from a number of variations (typically three) are perceived by users to be most appropriate for communicating their intended meanings. The test is conducted by showing a single participant each of the variants for an icon depicted on an individual card. The participant is told the context in which the icon would appear, and the intended meaning for the icon. Then the participant is asked to rank order the supplied



(a) High consistency



(b) High contrast

Figure 5.1: The trade-off in icon design illustrated by two alternative notations.

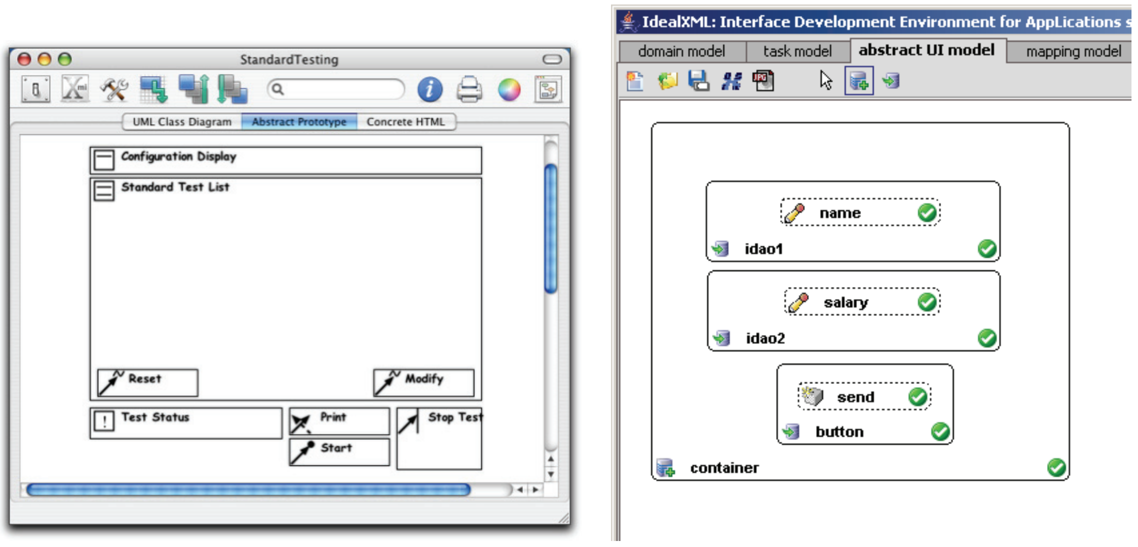
variants according to how appropriate they seem as representations of the intended meaning." [Misanchuk et al.00]

Comprehension test "is conducted by telling the participant the context in which icons will appear, but not the intended meanings for the icons. Then the participant is shown individual icons that have been created as a result from the prior appropriateness tests. Participants are asked to name the function represented by each icon. The designs are not accompanied by the labels they will have on screen." [Misanchuk et al.00]

Matching test "is conducted to determine how well an entire set of icons works. Each participant is shown the entire icon set and given one functional description to match with an icon out of the set. In order to avoid a situation in which participants choose icons based on a process of elimination, each participant should only be given one function for which to identify the correct icon." [Misanchuk et al.00]

Perceptibility test "is also conducted to determine how well an entire set of icons works. Each participant is shown a screen representation from the product under development, including the icons that would appear on that screen. Participants are given one task description at a time and asked to identify the icon that should be used to complete or begin the task. Each participant completes an entire list of tasks that covers the functions of all the icons, and each function appears in more than one task so that the participant does not simply use the process of elimination to guess the correct icons for tasks late in the test." [Misanchuk et al.00]

In [Finkenzeller08], a project thesis supervised by the author, icons have been developed for the most important MML model elements without an established notation: Abstract Interaction Objects (see sec. 6.4) and Media Components (see sec. 5.2). The project started with a brainstorming under consideration of some existing alternatives. For AIOs the notation from CUP [Van den Bergh and Coninx05] shown above and from *Canonical Abstract Prototypes* (*CAP*) [Constantine03] have been considered whereby the latter one could not be taken directly as it uses different AIO model elements than MML. Figure 5.2a shows the usage of CAP in *CanonSketch* [Can], a visual editor based on the Wisdom approach ([Nunes and Falcão e Cunha00, Nunes01], see also sec. 4.1.3). Figure 5.2b shows a faceted notation in *IdealXML* [Ide], an editor for UsiXML (see sec. 4.1.2). It is an extension of CAP,



(a) Usage of CAP in CanonSketch [Can]

(b) Extension of CAP with different icons and a faceted notation in IdealXML [Ide]

Figure 5.2: Notations for Abstract Interaction Objects based on Canonical Abstract Prototypes (CAP).

using different icons and, in particular, a faceted notation: Each AIO supports the four facets *input*, *output*, *navigation*, and *control* (see [Montero05]).

For Media Components common icons from operating systems or media player software have been considered. The test were conducted in three iterations where each iteration included different combinations of the four kinds of user tests described above (see [Finkenzeller08]). Altogether 18 participants took part. The small number of participants does not allow very general statements about the icons although for some icons the trends shown in the test were very clear. Nevertheless, the work ensures the quality of icons at least to a certain degree and exemplifies a possible way for more systematic icon design in modeling language development.

The resulting icons are depicted later in figure 5.5, figure 5.13, and figure 6.12.

5.1.5 Basic Technical Metamodeling Principles

This section describes some basic rules for structuring and denoting the metamodels. They are common to many other metamodels, including the UML specification, and apply to all metamodels presented in the following chapters of this thesis.

Containment Hierarchy Metamodels are usually built up in a containment hierarchy, i.e. a hierarchical tree structure resulting from the containment relationships between the model elements. For example in UML a package may own classes and a class may own operations. Each model element is owned by exactly one parent element. The top-most model element in a UML model is usually an instance of the metaclass Model which is a specific kind of Package. The containment relationship in the metamodel is denoted like a composite relationship in UML. The containment hierarchy allows for instance assigning each model element to exactly one namespace or mapping the model into formats like XML. Also it ensures that, if a model element is deleted, all child elements are deleted as well. Some metaclasses have multiple containment relationships; for example a Property in UML can be

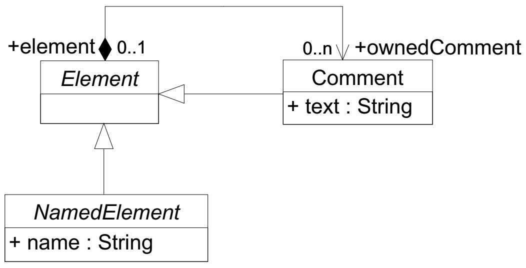


Figure 5.3: MML metamodel - Root Elements

owned by a class *or* by an association. In this case the “or” has to be interpreted as “exclusive-or”, i.e. when the model element is instanciated it always has only one parent.

In MML the metaclass MultimediaApplication represents the top-most root element. In the MML metamodel all model elements have at least one containment relationship either to MultimediaApplication directly or to one of its children.

Generalization Hierarchy Analogous to the containment hierarchy metamodels usually have a generalization hierarchy as well. This means that there is a most general metaclass where all other metaclasses are directly or indirectly subclasses from. In UML the top-most metaclass is Element. It is an abstract metaclass and has attached properties and constraints which should hold for any model element. An important direct subclass of Element is NamedElement which is used for any model element which has a unique name (like classes and most other model elements).

The same mechanism is used in MML. Figure 5.3 shows the corresponding part of the MML metamodel. All MML elements which have a name are subclasses (directly or indirectly) of NamedElement. In the following the generalization relationships to Element and NamedElement are not shown explicitly in the metamodel figures to reduce their complexity.

Advanced Structuring Mechanisms A basic principle or kind of “pattern” in the UML metamodel is to initially separate different aspects by different (abstract) metaclasses and combine them later again by multiple inheritance if required. For example, UML uses an abstract metaclass Namespace to generally define the concept of namespaces and a metaclass Type to generally define the concept of types. For instance a UML Classifier fulfills various different roles, including that it can be used as namespace and also as a type, and thus inherits from both Namespace and Type.

Moreover, metaclass properties can be refined by subclasses. For example, one can define that the abstract metaclass Namespace owns other model elements by an association ownedElement. However, the subclass Package owns PackagableElements which is specified by an association packagedElements. To specify that packages can *only* own PackagableElements as indicated by packagedElements (and not any model element as indicated by the association ownedElement inherited from Namespace) some additional definitions are required. UML2 offers new features to model such constraints: *subset* properties, (strict) *union* properties, and property *redefinition* (see e.g. [Alanen and Porres08, Amelunxen and Schürr06] for discussion). In the example, the property packagedElements can be marked as redefining ownedElement by denoting “redefines ownedElement” at the corresponding association end.

These mechanisms are heavily used in the UML2 metamodels. The advantage is that metamodels can be defined in a very modular way and relationships between properties of superclasses and subclasses can be specified more precisely. Moreover, those relationships are visible now directly in the metamodel diagram. The disadvantage is that the overall metamodel can become more complex. Furthermore they must be considered for the metamodel implementation (see [Amelunxen et al.04]). Usually it is possible to construct the metamodel without these constraints like in previous versions of UML. Often the property in an abstract superclass can be just omitted when it is redefined by all its subclasses. Also, the relationships between properties can be defined by conventional OCL constraints. As tool support for metamodel implementation did not support those constraints when the MML metamodel development started they are used very sparsely in the MML metamodel.

Conventions for Presentation For the presentation of the metamodel in terms of diagrams the same rules hold like described in the UML specification ([Obj07d], 6.4.2). These are the most important (additional) conventions used in the metamodel diagrams in this thesis:

- Abstract metaclasses are denoted with the metaclass name in italics.
- Metaclasses are often used in multiple diagrams. For example the metaclass Class is first defined a metamodel diagram showing class diagram elements and but later used in other metamodel diagrams. At its first occurrence – i.e. in its initial context – the metaclass is denoted with an attribute compartment showing the metaclass' attributes. Denoting a metaclass with suppressed attribute compartment indicates that the metaclass has already been defined in another diagram.
- Colors in metamodel diagrams are used to increase the diagram's readability and are not associated with additional semantics.

5.2 Modeling Media Components

This section introduces the basic concept of *Media Component* and specifies it in terms of a metamodel. Each subsection discusses a specific aspect of Media Components and shows a brief example using the racing game application from section 3.5.

At the end of each section there are optional paragraphs about Notation, Tool Support, and Code Generation for the discussed model elements, if required. Code generation at this point is only sketched on a platform-independent level to illustrate the model element's semantics. A concrete example for code generation for a specific multimedia authoring tool is given later in chapter 7.

Based on the concepts in this section the subsequent chapter 6 presents the overall modeling language.

5.2.1 Rationale

The core characteristic of a multimedia application is the integration of different media types. The choice of a media type determines the perception channels used to presented information to the user and how the user can interact with it. Thus, it is a fundamental decision which media type to use for an optimal presentation of a given piece of information to the user. Often the choice of media type is determined a priori by the customer's basic requirements or visions. For example, the learning application later in figure 8.7 must contain video with synchronized text. In addition, in some cases the media types can also be determined a priori by the application purpose, the target platforms or target devices, e.g. when developing infotainment systems.

Furthermore, the production of media content can take much effort and time and requires specific experts and tools. (Re-)using already existing media objects can require tasks like selection, post-processing, copyright management, etc. Creation of new media objects is either performed within a multimedia authoring tool (like animations in Flash) or in a separate process (like video production).

For these reasons MML takes the position (in contrast to HyDev, sec. 4.3.1) that the developers should be enabled to specify the media objects as early as possible in the development process. Due to their possible high complexity and relevance for the development process they are considered as first class entities and as part of the overall application structure (like in OMMMA, sec. 4.3.2).

5.2.2 Media Types and Media Components

The MML metamodel defines the different media types as base for the code generation. A general classification of media types can be derived from multimedia standards and existing research approaches for multimedia documents (see sections 2.1 and 4.3). Here in context of this thesis, the purpose of the model is application development. Thus, it is useful to distinguish between those media types which need different development support. For example, animations must be further separated into 2D and 3D animations, as they are developed by significantly different experts, tools, and implementation code. On the other hand, some literature introduces new media types where currently no established knowledge about their development exists. For example, the [] specifies E-Ink as an additional media type. But currently only little knowledge exists about the way E-Ink is used and implemented different than conventional text or graphics. There is also no specific implementation support yet. Similarly, media types related to other human senses – touch, smell, taste – are not relevant here. For example, haptic output in shown in various research examples and even commercial devices exist [Hayward et al.04]. In the form of "force-feedback" it is already common in commercial input devices for computer games and is supported by the DirectX API [Microsoftb]. However, there is currently no specific format for haptic information and it can not be handled and designed in the sense of a media object. The purpose of modeling approaches like MML is to make established and platform-independent knowledge explicit and support it by code-generation. Modeling approaches can not predict the usage of new technologies or invent them; they have to be established by research and implementation support by the respective experts first. Thus, it is not useful to include such non-established media types into MML.

Figure 5.4 shows the resulting metamodel for media types. The media types are classified into temporal and discrete media as it is relevant for the possible behavior in an application. Temporal media types are *Audio*, *Video*, *Animation2D* and *Animation3D*[1]. Discrete media types are *Image*, *Graphics* and *Text*.

The terms graphics and image are used to distinguish between synthesized and captured media like in fig. 2.2 in chapter 2. The term image (instead of photo, like in fig. 2.2) is used to refer to any kind of bitmap (raster graphics) which must not necessarily result from a photo. "Graphics" refers to vector graphics. However, basically a vector graphics can also contain a bitmap as part of the graphic (e.g. a texture).

An animation is basically defined as a visual change over the time [Steinmetz00]. More specifically it consists of graphics or images which change over the time. A "change" means here the change of a parameter with impact on the visual appearance. These can be parameters on the image/graphic as a whole – like its position on the screen or its orientation – as well as the content itself like colors or shapes. In this understanding it is not important whether the animation's behavior is predefined or not nor whether the animation changes itself or is just changed as a whole – i.e. a graphic which is moved over the screen is interpreted as animation as well. Complex animations usually base on vector

[1]To allow direct implementation in Java all metaclass names in MML start with an alphabetic character

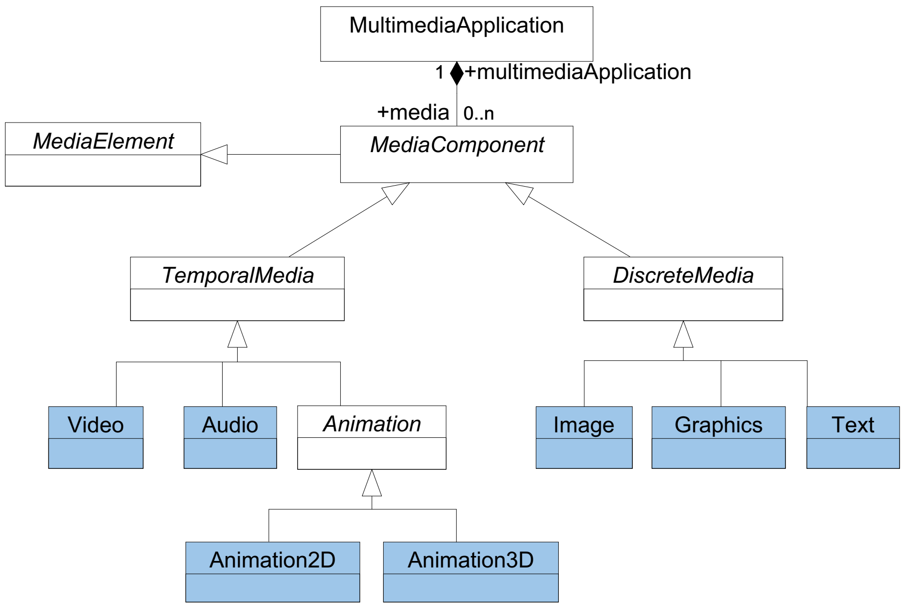


Figure 5.4: MML metamodel - Media Types

graphics, like Movie Clips in Flash. An example for simple animations based on bitmaps are animated GIF images. From the viewpoint of development it is also possible to classify them as images as they are usually developed and handled similar to conventional still images. In 3D space there is no need to distinguish between animated and non-animated 3D graphics as both are implemented and rendered with 3D authoring tools and 3D rendering software.

A video is a sequence of images and can include synchronized audio. Audio usually refers to captured sound but can also be synthesized using e.g. the Midi file format. However, there is usually no significant difference in development. Text refers to any kind of text which can include formatting and hyperlinks. Basically, any synthesized media object can be transformed into captured media either using software which supports the export or just by capturing it, e.g. by taking a screenshot or a screen video.

It is important to understand that in a MML model a media object should only be specified explicitly when it is relevant for the application development. Usually this means that the creation of the media object requires some effort and/or the media object must be integrated with other parts of the application. For instance, adding a conventional text label or an adorning image to a user interface usually requires no specific development support and thus such media objects need not to be defined in the model.

On a first look, media objects are purely documents, i.e. "‘content"’. However, in context of a multimedia application the usage of a media object obviously implies the ability to present this content to the user. Hence, the media objects must be encoded and rendered or played. Furthermore, there are some standard operations on the media elements. Some of them are often available at the user-interface, like to pause and resume a video or to mute audio. Usually this is implemented by a player component which presents the media content. In addition, standard operations on media elements must be available internally for the application logic to control the media objects, e.g. to start an animation or set the location of an image. As this standard rendering functionality and standard operations is in context of application development always part of media usage it would not be useful if the modeler has to model it everytime. Thus, MML provides Media Components instead of purely

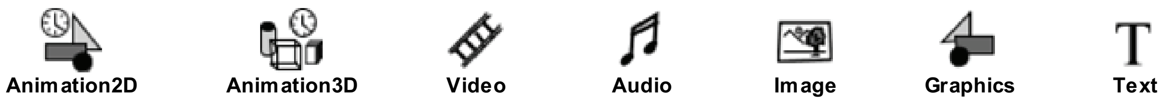


Figure 5.5: Icons for MML Media Components.

media types. A *Media Component* encapsulates media content and standard player or rendering functionality and provides some standard operations depending on the media type. Thus, in figure 5.4 all media types are defined as subclass of *MediaComponent*. The abstract metaclass MediaElement represents any structural part of a Media Component including Media Components themselves.

Notation Basically, the notation of Media Components corresponds to classifiers in UML, i.e. a solid-outline rectangle containing the Media Component's name. Optionally it is shown with a compartment containing properties the Media Component's inner structure (section 5.2.7).

MML provides icons to denote the different types of Media Components. Like in UML, the modeler can optionally use the icon, a text label or both to denote the Media Component's media type (see example on UML stereotypes notation in fig. 3.6). Figure 5.5 shows the icons for Media Components. They were developed based on user tests as explained in section 5.1.4.

Code Generation Media Components are mapped to placeholders for the actual media content and an implementation for the player and rendering functionality. A placeholder is simple dummy content corresponding to the component's media type, e.g. a short dummy video, audio, animation, image, graphic or text. The placeholders are then filled out or replaced by the media designer. The (file) format for the placeholder depends on the implementation platform. For example, for the Flash platform a video can be mapped to a dummy video in FLV file format while for the Java platform it is mapped e.g. to AVI. A 2D animation in Flash is usually a Movie-Clip within a FLA file while in Java it might be a Java class. In XML-based formats like SVG the media components are usually implemented by corresponding XML tags; probably associated with some additional program code like Java Script. For the player and rendering functionality the APIs of the target platform are used, e.g. the MediaPlayer component in Flash.

5.2.3 Interfaces

As explained in the last section a Media Component provides standard operations. MML supports by default the basic operations usually supported by any platform. They can be supported by automatic code generation in straightforward manner. Moreover, it should be possible to specify additional custom operations on Media Components. This includes operations realizing filters and transitions. Filters and transitions are usually realized in implementation frameworks by classes or components of their own [Gibbs and Tsichritzis95]. However, on platform-independent level they are considered to be too implementation-specific. Thus, in MML they are considered just as operations (i.e. operation signatures) which can then be implemented using the respective classes and algorithms of the implementation platform.

Consequently, different sets of operations are required for Media Components: Standard operations resulting from its media type and additional custom operations. The standard operations are the same for each media component with the same media type. In addition, there are sets of operations which apply to e.g. all temporal media types or all visual media types. It should also be possible to

Interface Name	Media Types	Standard Operations

TemporalDefault	Video, Audio, Animation2D, Animation3D	play()
		pause()
		stop()
		gotoAndPlay(cuePoint)
		gotoAndStop(cuePoint)

VisualDefault	Video, Animation2D, Animation3D, Image, Graphics, Text	getWidth()
		getHeight()
		setWidth(i:Integer)
		setHeight(i:Integer)
		getX():Integer
		getY():Integer
		setX(I:Integer)
		setY(i:Integer)
		setVisible()
		setInvisible()

AuditiveDefault	Video, Audio	setVolume(percent:Integer)

Table 5.1: Media Component Standard Interfaces

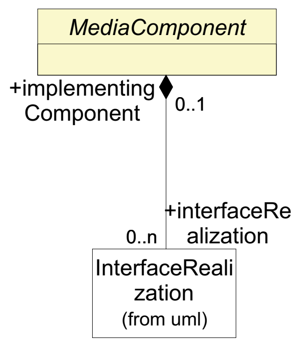
define a custom operation only once and assign it to multiple Media Components, e.g. if multiple Media Components should use the same kind of filter.

A useful modeling concept for this purpose are *Interfaces* as used in UML. The standard operations are defined once within standard interfaces. They can be classified into operations for temporal media, visual media, and auditive media resulting in three standard interfaces TemporalDefault, VisualDefault, and AuditiveDefault. Custom operations can just be modeled as – and if necessary grouped by – additional interfaces which can be provided by one or more Media Components.

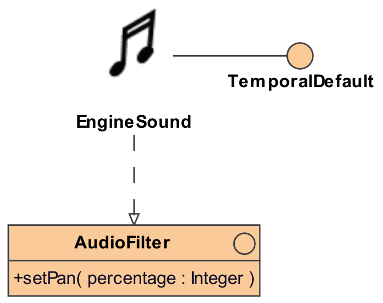
Table 5.1 shows the operations defined in the standard interfaces and the media type they are assigned to. They are not defined directly in the MML metamodel as they are considered as instances of the UML metaclass Operation and thus reside on model level (see [Atkinson and Kühne02], pp.8–12). The standard interfaces are thus defined as model library elements. MML modeling tools must automatically create them and assign them to the Media Components. Figure 5.6 shows the metamodel extract connecting Media Components with interfaces. The reused parts of the UML metamodel defining interfaces, operations, etc. is shown later in figure 6.5 and 6.4.

Figure 5.6b shows an example for an Audio Component EngineSound realizing the standard interface TemporalDefault and a custom interface AudioFilter. Each MML Media Component realizes by default the standard interfaces from Table 5.1 according to its media type. The modeler needs not to specify them explicitly in the MML models.

Tool support The default interfaces are created automatically by the modeling tool. When a Media Component is created the modeling tool automatically adds interface realization relationships to the corresponding default interface.



(a) MML metamodel extract.



(b) Example for specifying interfaces for Media Components.

Figure 5.6: Interfaces for Media Components.

5.2.4 Media Representations

As already observed by OMMMA (see sec. 4.3.2), Media Components mostly represent concepts of the application domain. The domain concepts are modeled as conventional domain classes and specified in terms of a conventional class diagram as shown in section 6.2. In a racing game application there is for example a domain class Car. It might be represented by an animation and sound. Other examples are a video representing a learning unit, graphics representing a part of a map, or a 3D graphics, an image, and a text representing a product in an online shop, etc. As a domain class can be represented by multiple media components – possibly of the same media type, e.g. multiple images for each product – it is useful to model this by relationships between the domain class and the respective media components. In MML this kind of relationship is called *Media Representation*.

In MML each media component has an identifying name. Media components are considered as first class elements, similar to classes. This is necessary to distinguish them from each other when they should be referenced from other parts of the MML model as well as for code generation. In addition, it should be possible to model individual properties for each media component, like its inner structure or custom behavior (see below).

Often a Media Component represents only some specific properties or operations of a domain class. For example, an animation Speedometer represents the property speed of the class Car. A sound SkidSound might be played when the car breaks, i.e. it in particular represents the operation break() of the class Car. MML allows to optionally annotate the Media Representation relationship with the names of the properties or operations to be represented. It is also possible to annotate the Media Representation with names of multiple properties or operations. Properties might also be association ends.

In figure 5.7 the domain class Car is represented by two animations CarAnimation and Speedometer and three audio components SkidSound, Horn and EngineSound. In the figure, Speedometer represents the property speed and SkidSound represents the operation break(). For the other Media Components no specific property or operation is specified.

Figure 5.8 shows the metamodel for Media Representations. An MMA_Class is an abstract metaclass for any kind of class in a multimedia application modeled with MML.

Code Generation A Media Representation is mapped to a link between the domain class and the Media Component. It can be realized for instance by a variable or association in the domain class

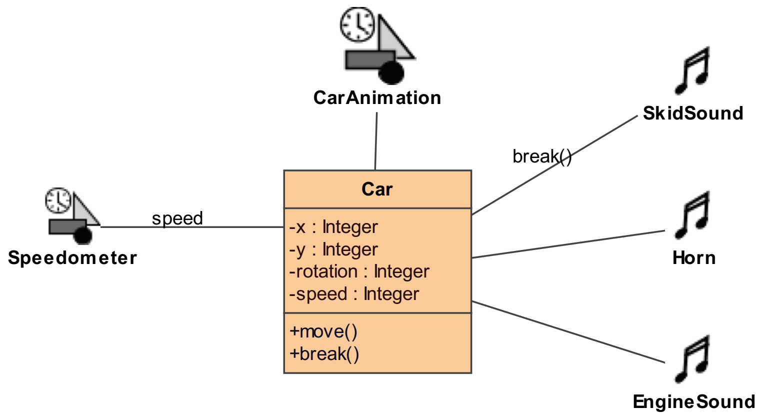


Figure 5.7: Example for Media Representation.

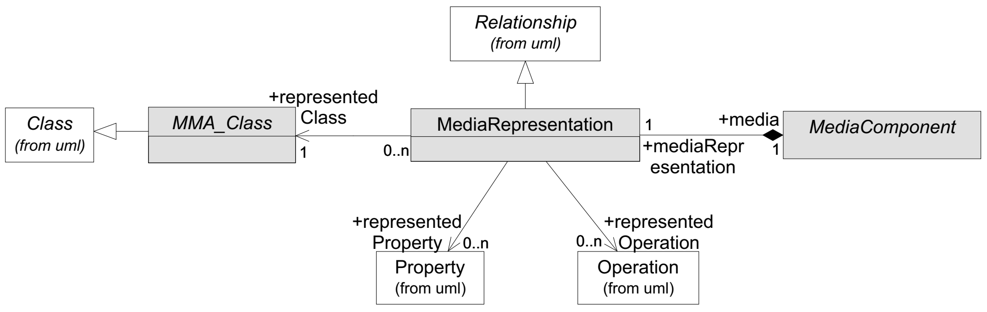
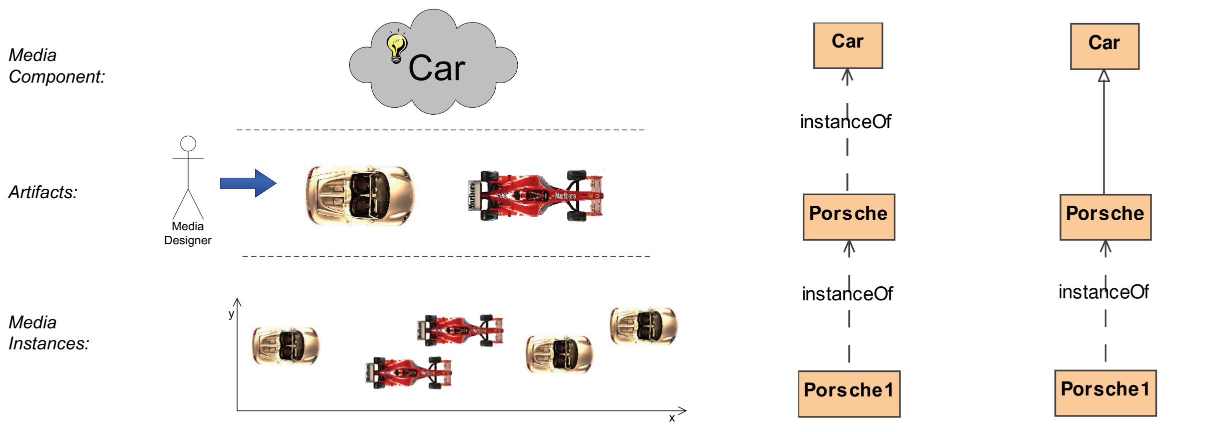


Figure 5.8: MML metamodel for Media Representation.

or by the design pattern *Observer* [Gamma et al.95]. If the media Representation is annotated with a class property or operation then a property change or operation call potentially causes an update of the Media Component. When using the *Observer* pattern, this can be implemented by notifying the observer in the property's setter operation (setSpeed() in the example) or at the end of the specified operation (break() in the example). However, the details of the manipulation of the Media component by the domain class have to be implemented manually.

5.2.5 Artifacts and Instances

On a first look the Media Components seem to be relatively simple constructs – a media object encapsulated by a player or renderer. However, considering all possible cases in multimedia development the situation becomes more complex: For example the racing game application might provide the user with different kinds of cars to choose from, say Porsche and Ferrari. In the early stage of application development it is sometimes not definitely decided how many cars or which kind of cars the application will provide – the developers just start with some example cars. Moreover, a common way of implementation is to keep the concrete cars modular and to load them dynamically into the application at runtime. Dynamic loading of Media Components is an important technique in larger multimedia applications and must be considered by the modeling approach. Moreover, sometimes the user even might create his own custom cars at runtime using a specific editor provided by the application. Other typical examples for dynamic loading of Media Components are levels in games, exhibits in a virtual museum or learning units in education software. In these examples Car, Level, Exhibit, or Learning



(a) Car Example

(b) Interpreted as instantiation

(c) Interpreted as generalization

Figure 5.9: Illustrating example of abstraction layers of Media Components

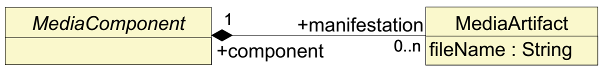
Unit are kind of abstractions of the (probably unknown) concrete media components.

Moreover, an application can contain multiple instances of the same concrete media component. For example, in the racing game application multiple Porsche cars might be visible on the screen. Each Porsche can be represented by the same animation, i.e. the same media component, but has a different location on the screen and is related to a different domain object (e.g. associated with a different player). It is also possible that some visual properties vary over the different instances, e.g. the different Porsche cars have different colors. This phenomenon is not restricted to a specific media type: There can be multiple instances of the same image at different locations on screen (e.g. in different scales) or multiple instances of the same video (having different states). Depending on the technology this is usually implemented either by multiple references to the same media object (e.g. to the same video) or by creating multiple copies (e.g. when placing a Flash MovieClip from the library on the screen). From the viewpoint of object-oriented programming the latter mechanism can be considered as similar to prototypes in “prototype-based” (or “object-based”, see e.g. [Noble et al.99]) programming languages. Anyway, the media components (e.g. Porsche and Ferrari) have to be designed only once by the media designer. Figure 5.9a illustrates the observed different abstractions.

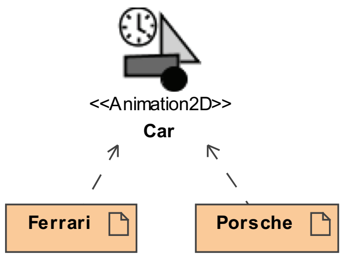
Despite of the possible implementations in different platforms an abstract modeling language should support these different conceptual views on media components. There are different ways which might be used to model the observed abstractions. One can interpret them as kind of instantiation: the Porsche is an instance of Car and can itself be instantiated multiple times on the screen (fig. 5.9b). Another possibility is using inheritance and interpreting Car as an (abstract) superclass of Porsche (fig. 5.9c). However, both interpretations raise the problem that media components are basically no classes and concepts like inheritance and instantiation can not be directly applied to them without further definitions. For instance, it would be necessary to define the impact of inheritance as MML supports to define the inner structure of media components (see below).

A beneficial concept to model these relations can be taken from components in UML.[2] In UML, a component can be manifested by one or more artifacts. “An artifact is the specification of a physical

[2]Please note that although Media Components reuse some selected concepts of UML components they are still an independent concept. There are significant differences between components in UML and Media Components.



(a) MML metamodel for Media Artifacts.



(b) Example for Media Artifacts.

Figure 5.10: Media Artifacts.

piece of information that is used or produced by a software development process, or by deployment and operation of a system. Examples of artifacts include model files, source files, scripts, and binary executable files, a table in a database system, a development deliverable, or a word-processing document, a mail message. [...] Artifacts can be instantiated to represent detailed copy semantics, where different instances of the same Artifact be deployed to various Node instances (and each may have separate property values, e.g., for a 'time-stamp' property)." [Obj07d]. Examples are an artifact Order.jar manifesting the component Order or an artifact Oracle manifesting the component Database Management System [Jeckle et al.04]. In the UML metamodel the Manifestation relationship is defined as a kind of abstraction. This concept fits very well as archetype for media components:

> A *Media Component* (e.g. Car) can be *manifested* by one or more *Media Artifacts* (e.g. Porsche) which can be instantiated (probably by copy) multiple times where different instances may differ in some properties (e.g. their location on the screen).

Specifying media artifacts provides an important information to the media designer about which and how many media artifacts must be designed. On the other hand it is not always possible to specify the complete set of media artifacts as it might be not decided yet or it should be extended at runtime. Thus, it is not expected that all media artifacts are defined in the MML model but it is recommended to state at least some example artifacts.

Figure 5.10a shows the metamodel for Media Artifacts. In MML, each Media Artifact can manifest only one Media Component. The manifestation relationship has no further properties and thus is not specified as a metaclass of its own.

Notation The media artifacts are denoted similarly to artifacts in UML. They are optionally marked by an icon and are connected to a media component by a manifestation relationship. Figure 5.10b shows an example from the racing game application.

Code Generation For each Media Artifact one placeholder is generated as described in section 5.2.2 for Media Components. Each placeholder is named like the artifact in the model. If no Media Artifacts are specified for a Media Component then a single placeholder is generated which is named like the Media Component itself.

5.2.6 Keywords for Media Representations

An important information about associations in UML class diagrams is given by the multiplicities at association ends specifying the numerical relation between instances of two associated classes. This section briefly discusses whether there is an analogy to that for Media Representations. As discussed in the previous section one can distinguish between artifacts and instances of Media Components. Consequently, the situation becomes more complex and two different relations can be identified: First, the number of media instances related to a single domain object. This relationship can be defined when specifying the concrete instances on the user interface in the MML Presentation Diagram described in section 6.4. Second, the number of media artifacts which can represent a single domain object and vice-versa. There are four basic cases which can be distinguished:

1. The most simple case is that there is only one media artifact which represents all domain objects. For example, in the racing game there might be only one sound artifact Horn which is the same for all car objects. It can be interpreted as n:1 relation between the domain object and the intended number of possible representations (i.e. *not* the number of media instances but the number of media artifacts which could represent the domain object). On implementation level this means that there is only one media artifact and thus it can be statically assigned to all domain objects.

2. The second case is that each domain object is associated with its own representation. A typical example is a museum application where usually each exhibit has a representation of its own. In the racing game, for instance each track might have a visual representation of its own. It can be interpreted as 1:1 relation. On implementation level it means that there needs to be e.g. a list or table or mapping rule which assigns the correct representation to each domain object (e.g. assign “MonaLisa.jpg” to the exhibit “Mona Lisa” and “VenusdeMilo.jpg” to the exhibit “Venus de Milo”, etc.).

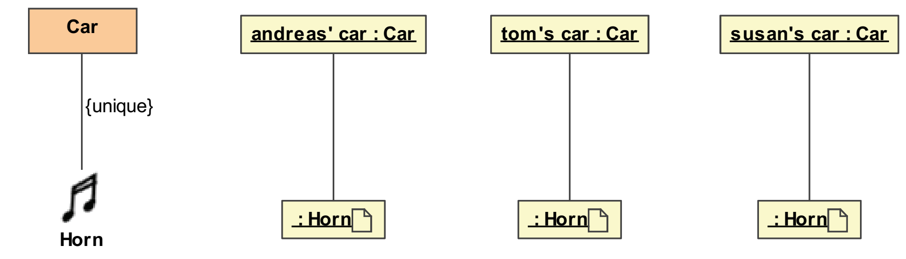
3. A domain object also can be represented by multiple media artifacts but a media artifact also represents multiple domain objects. This is the usual case at racing games where each player selects a car type but several players might select the same one. This can be interpreted as n:n relation. On implementation level it requires to assign the media artifacts dynamically to domain objects, e.g. depending on the user’s selection of a car type.

Basically it would be possible to use multiplicities at the Media-Representation relationship for its specification but this would easily lead to misunderstandings as multiplicities are commonly used at associations to denote the relation between instances. Thus, the three cases identified above are denoted in MML by the keywords *unique* (first case), *one-to-one* (second case), and *dynamic* which denotes all other cases as the generated code in dynamic cases is always the same. The keywords are denoted as text in curly braces placed close to the Media-Representation relationship they belong. Figure 5.11 shows examples for the three identified cases. The MML models are shown on the figure’s left hand side while the right hand side provides a possible corresponding situation at runtime.

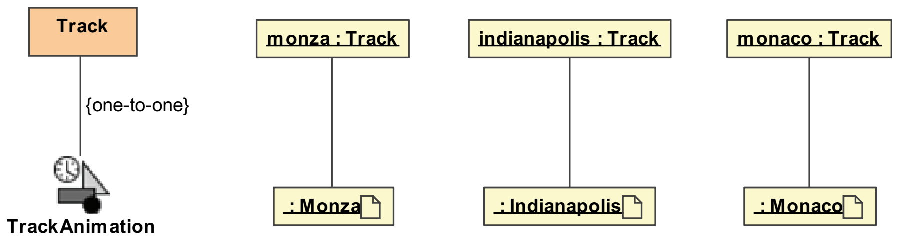
Additional constraints for the third case, like that each Media Component can only represent one car at the same time (i.e. each user is represented by a different car type), are not considered by MML. Such constraints are better realized during the final implementation by manually specified code.

5.2.7 Media Parts

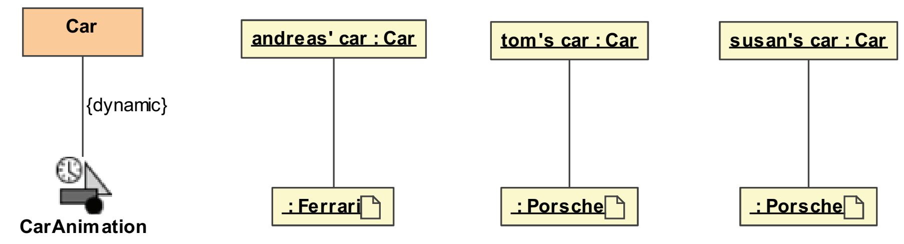
As explained in section 3.1.1 (see also example in fig. 3.8) the inner structure of media components is important in interactive applications. It must be possible to specify inner parts of a media component if they should be accessible for other parts of the application, e.g. to associate them with some event handling code or to control their behavior by program code. For this purpose the modeling language



(a) Keyword *unique*: All domain objects are represented by the same artifact.



(b) Keyword *one-to-one*: Each domain objects is represented by an artifact of its own.



(c) Keyword *dynamic*: No a priori information about the assignment of artifacts to domain objects.

Figure 5.11: Examples for Keywords for Media Representations: The left hand side shows the MML example models, the right hand side corresponding example situations at runtime.

should support to define the inner structure in an abstract and simple way. It is not the intention of MML to define complete media objects in the models. Rather, in the viewpoint of MML creative media design should be performed using the established authoring tools for the respective media types. Thus, the definition in the MML model should only define an agreement on abstract level between the media designers and the software developers.

A Media Component can consist of different kinds of parts depending on the media type. Synthesized media usually consists of primitives having some kind of semantics, e.g. shapes and lines inside of vector graphics. Based on the primitives, more abstract parts can be identified, e.g. a wheel within a graphic representing a car. In contrast, media captured by sensors can only be decomposed by its natural dimensions, e.g. a specific region in a bitmap image is identified by its spatial x- and y-coordinates. Temporal media always has a temporal dimension which can be used for its decomposition, e.g. specific points of time in a video.

3D animation – as synthesized media type with the largest number of dimensions – has the most complex inner structure. Relevant types of inner parts can be identified based on an existing abstract modeling language for 3D applications called *SSIML* [Vitzthum and Pleuß05]. SSIML specifies the concepts common in 3D standard formats (like *VRML* [ISO97a] and *X3D* [ISO04]) and 3D authoring tools and is used as base for a model driven development approach for Virtual Reality and Augmented

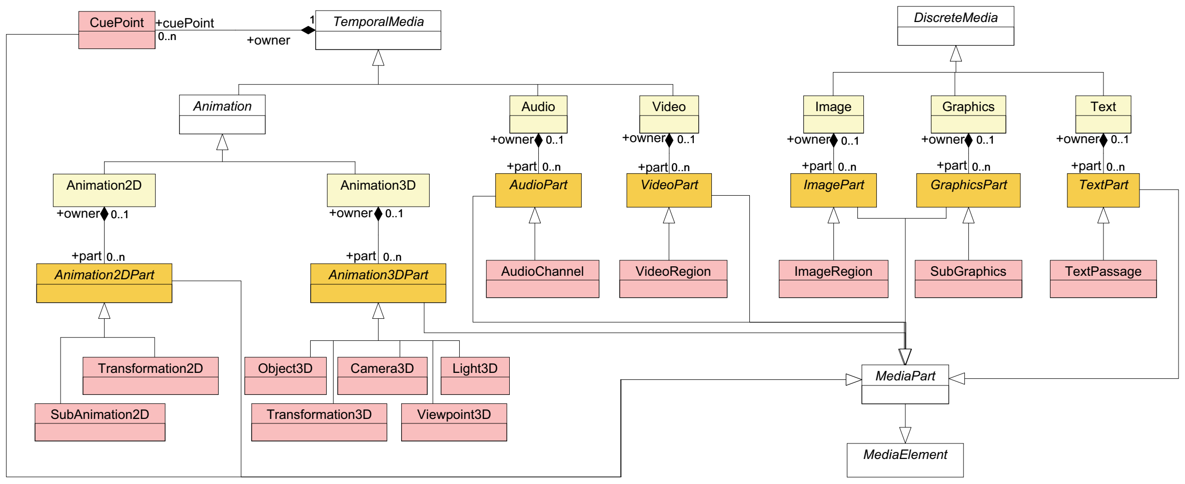


Figure 5.12: Metamodel for Media Parts.

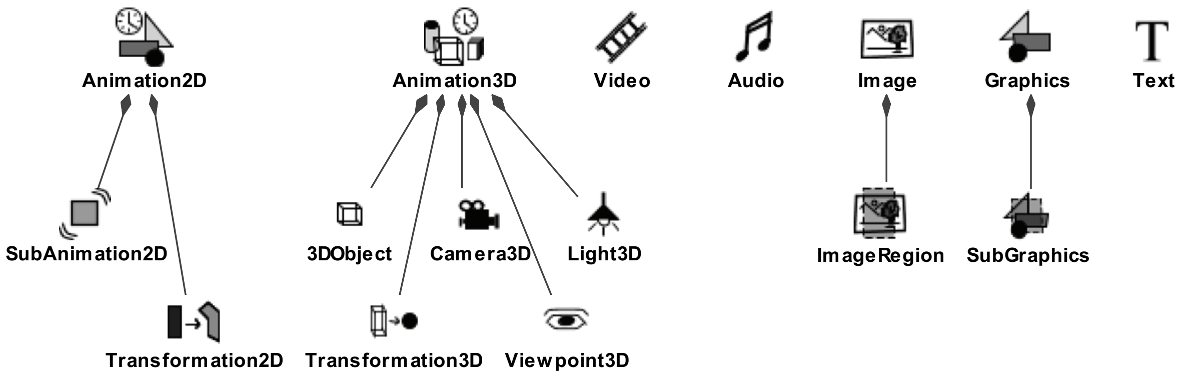


Figure 5.13: Icons for MML Media Components and Media Parts (some Media Parts do not have an icon representation yet).

Reality applications [Vitzthum05, Vitzthum06, Vitzthum and Hussmann06].

In the 3D domain the structure of a 3D scene (i.e. the virtual world) is often denoted by a scene graph. Implementation formats usually use the same kind of hierarchical structure. A scene graph is a directed acyclic graph (DAG) containing the content of the scenes – geometric objects, lights, viewpoints, etc. – as its nodes. Positioning and orientation of objects is usually defined by transformation nodes. A transformation is applied to all child nodes. This enables the positioning of groups of objects relative to other objects. For example, the wheels of a car are positioned by a transformation relative to the car's body and the whole car is positioned within the whole 3D scene by another transformation. The resulting hierarchical structure is very similar to a tree structure. However, it is possible that a node has several parents to enable reuse of objects for increased efficiency of the implementation. For instance, the four wheels of a car can be implemented using four references to the same wheel object and positioning them at the four sides of the car by four transformations. Also, in many 3D formats, only some kinds of nodes – e.g. transformation nodes and group nodes – can have children while the others may only act as leaf nodes in the scene graph.

MML abstracts from these concepts supporting a simplified tree structure for inner parts of media components. Any kind of node may have children, independently of whether it is further decomposed internally using transformation nodes or group nodes. Also the structure is always a simple tree structure independently of whether it is implemented internally by reusing a node multiple times for efficiency reasons. The nodes are called *Media Parts*, reusing concepts from Parts in UML Composite Structure Diagrams which describe e.g. the inner structure of UML components (see e.g. [Hitz et al.05], chap. 3.5). A Media Part represents a specific structural part of a media component. For example for 3D animations the possible Media Parts derived from SSIML are Object3D which refers to an arbitrary complex geometrical object, Transformation3D, Camera3D, Viewpoint3D, and Light3D[3]. Only those Media Parts are explicitly modeled in MML which are relevant for other parts of the application, e.g. as they should be manipulated by the application logic. Other Media Parts which are only required for the concrete realization of the Media component are omitted. A parent-child relationship between two nodes specifies that the child node's position is relative to it's parent node. An implementation for instance in VRML requires additional transformations between the elements which are though not modeled explicitly in MML.

The metamodel in figure 5.12 shows the different kinds of Media Parts available in MML for the different Media Types. Each temporal media can be structured by *Cue Points* which are abstract points of time within a temporal Media Component's timeline. A Cue Point can represent for instance the beginning of a scene in a video or a specific point of time in an audio. The Cue Points are abstract as well, i.e. they are described by a meaningful name while their concrete time values needs not to be specified until the final implementation.

2D animations adopt those elements from 3D animations which make sense in 2D space, i.e. objects (SubAnimation2D) and transformations (Transformation2D). Audio can be decomposed in different channels which represents on abstract level any decomposition beside CuePoints, for instance a stereo channel or a MIDI channel. Video can be decomposed into regions like an image. Image, Graphics and Text can be decomposed in the spatial dimension only. An ImageRegion represents a spatial area within an image. A SubGraphics represents a part within a graphic. A TextPassage represents a part within a text.

In addition, some types of Media Components can contain other Media Components. For example, a video can contain sound, graphics can contain bitmaps, and animations often can contain bitmaps, sound, or even videos. Such containment can be modeled by references to other Media Components as shown in the next section 5.2.8.

For the Media Parts custom icons have been developed based on user tests as described in section 5.1.4. They are shown in figure 5.13 A few Media Parts currently have not been considered yet: Channel, VideoRegion, TextPassage, and CuePoint.

5.2.8 Inner Properties

As discussed above, Media Components can be manifested by different Media Artifacts which are instantiated by Media Instances. Analogously, a Media Part can be manifested by different *Part Artifacts* which are instantiated by *Part Instances* (fig. 5.14). The Media Part initially is a kind of abstract concept, e.g. Wheel. Each instance of the Media Component Car can contain several instances of Wheel, like wheel_left and wheel_right. wheel_left and wheel_right are thus placeholders for concrete instances – analogous to properties in (domain) classes. They are thus called *Inner Properties*[4]. An Inner Property enables access to the Part Instances for a given Media Instance. Thus it is mandatory

[3]MML benefits from the concepts discussed in SSIML but still provides a slightly higher level of abstraction. Many concepts are similar but must be adapted to the different purpose of MML as Media Components in MML can be manifested by different artifacts and instantiated multiple times while the main focus of SSIML is a single large 3D scene.

[4]The term "Inner" is used to distinguish the model element from other general properties of Media Components

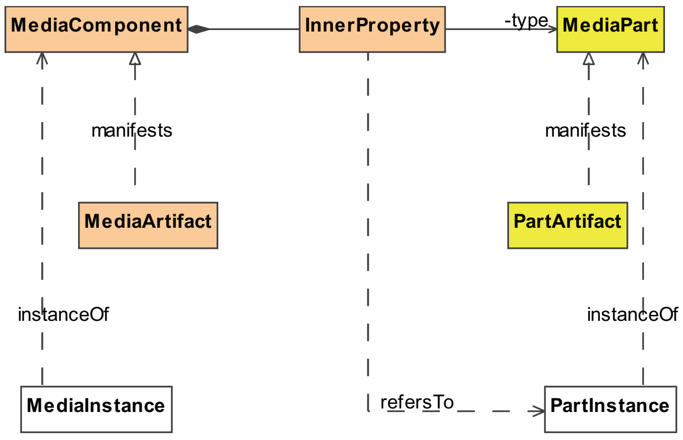


Figure 5.14: Illustration of different abstraction layers of media and their inner structure. In MML structure diagrams the white colored model elements need not to be specified in the model. The light yellow colored model elements are modeled only optionally.

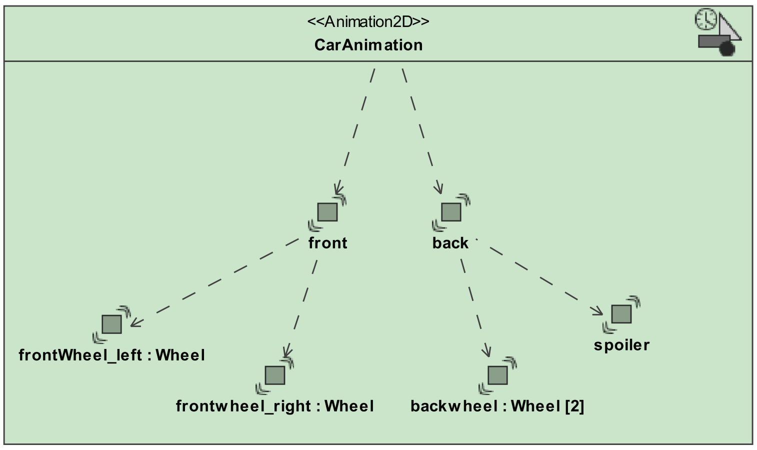


Figure 5.15: MML Example: Media Parts and Inner Properties.

to specify them in the MML model. Optionally, the name of the Media Part can be specified after the Inner Property name separated by a ':', e.g. wheel_left:Wheel. Media Part names need only to be specified explicitly if multiple Inner Properties refer to the same kind of Media Part, like wheel_left and wheel_right which both refer to Wheel. Often a Media Property refers to a Media Part of its own (e.g. a car contains only one hood). In that case the Media Part name can be omitted in the MML model and is by default the same name as the Inner Property name (but starting with an uppercase letter). For brevity it is also possible to specify a multiplicity for a Media Property to denote multiple instances of the same type. This enables modeling in a compact way for instance multiple wheels of a car or a large number of spectators at the stand. Specifying a property with the name n and the multiplicity m corresponds to m properties of the same type named with the concatenation of n and a consecutive number, i.e. n1, n2, n3, etc. Specifying a multiplicity range means that the concrete number of properties is either decided during implementation or calculated at runtime.

Figure 5.15 shows as example a Media Component Car containing different Media Properties. There are different Inner Properties referring to the Media Part Wheel. All Media Parts in this example are from type SubAnimation. The frontwheels, fronwheel_left and frontwheel_right are both modeled explicitly, while the two backwheels are modeled by a multiplicity. As explained above one can reference them by the automatically generated names backwheel1 and backwheel2. For the other Inner Properties in the example no Media Part names are specified which implies that front, back and spoiler refer to Media Parts Front, Back and Spoiler. Furthermore the hierarchy of the Media Properties specifies that frontwheel_left and frontwheel_right are located relative to front while backwheel and spoiler are located relative to back.

Media Properties can not only refer to MediaParts but they may also refer to other Media Components. This can be used to specify that a Media Component contains an instance of another one. For example, in the racing game the track is represented by an animation TrackAnimation which contains several instances of CarAnimation. The car animation instances are thus located inside the track animation and can be referred from it. Of course, the car animations still can be used at other places in the application as well, for instance in the game's menu when the user has to select a car. The metamodel for Inner Properties is shown in the next section 5.2.8 when Part Instances have been introduced.

5.2.9 Part Artifacts

Until now, the Part Artifacts have not been mentioned. Figure 5.15 can be interpreted in different ways: Either each wheel is manifested by an artifact of its own or some wheels are manifested by the same artifact (see examples in fig. 5.16)[5]. Part Artifacts need not to be specified explicitly in MML. If omitted then the Media Property is manifested by an implicit Part Artifact which has by definition the same name as the Media Property itself. For example the model in figure 5.16a does not specify explicitly any Part Artifacts. Thus, by definition each wheel is manifested by an individual Part Artifact of its own (named like the respective Media Property, i.e. frontwheel_left, frontwheel_right, backwheel1, and backwheel2). However, it is then still up to the media designer whether to create really different wheels or e.g. to copy and paste a created wheel multiple times into the generated placeholders.

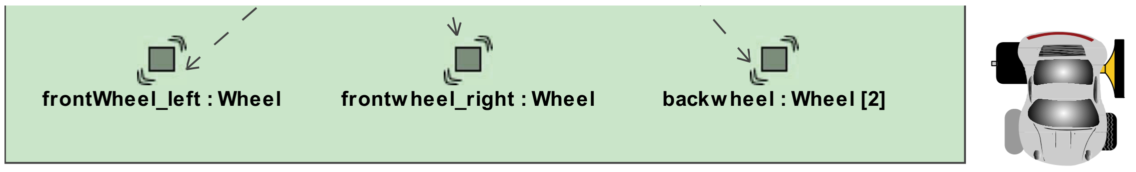
Optionally, a Part Artifact can be specified explicitly at each property (denoted like a Tagged Value in UML)[6]. This can be used to specify that multiple Media Properties are manifested by the same artifact. In fig. 5.16b the two frontwheels are instances of the Part Artifact frontwheel while the two backwheels are instances of the Part Artifact backwheel. In figure 5.16c all four wheels are instances of the same Part Artifact. If, for instance, in fig. 5.16b no Part Artifact would be specified for the backwheels then there would be three Part Artifacts: one for the frontwheels (frontwheel) and two for the backwheels (backwheel1 and backwheel2).

Another aspect is the relation between Part Artifacts and the different Media Artifacts, i.e. whether different cars have different wheels or not. The default interpretation in MML is that there exist individual Part Artifacts for each Media Artifact, i.e. different kind of wheels for each car (figure 5.17a). Alternatively, there might be only one unique kind of wheel which is used by all cars (figure 5.17b). This means that the Media Part (wheel) is manifested by only one unique Part Artifact in all Media Artifacts (the cars). This case is specified in MML by the keyword unique at the respective Media Part. (If other Media Parts refer to the same Part Artifacts they are thus unique as well.)

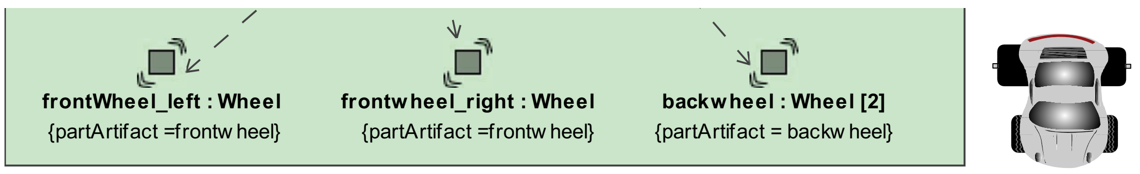
Figure 5.18 shows the metamodel for Inner Properties and Part Artifacts. A InnerProperty may refer as its type to any MediaElement, i.e. either a MediaPart or a MediaComponent. An optionally specified Artifact must correspond to the specified type.

[5]The general term "artifact" can be used when it is clear by the context whether it refers to a MediaArtifact or a PartArtifact

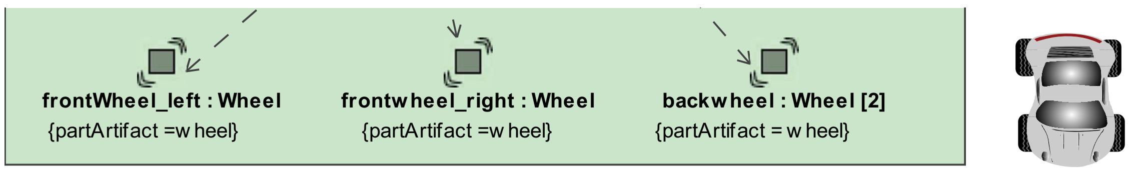
[6]The Part Artifacts in MML do not have any additional properties besides a name



(a) Each Media Part is possibly manifested by an artifact of its own (default case)


frontWheel_left : Wheel
{partArtifact =frontw heel}
frontwheel_right : Wheel
{partArtifact =frontw heel}
backwheel : Wheel [2]
{partArtifact = backw heel}

(b) Two artifacts specified for frontwheels and backwheels



(c) The same artifact specified for all four wheels

Figure 5.16: Specifying Part Artifacts.

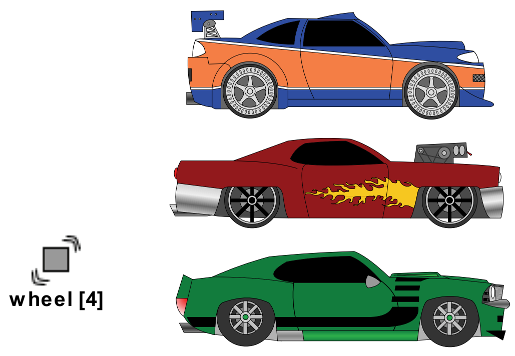
Tool Support The Part Artifacts need not to be specified as model elements of its own in the MML diagrams. They are just used to be referenced by Media Properties. Of course, a modeling tools manages them and shows them in the containment tree view which is usually part of every modeling tool. It is also a common standard functionality that if a modeler has to refer a Part Artifact the modeler can either create a new one or choose an existing one from a selection provided by the tool.

5.2.10 Variations

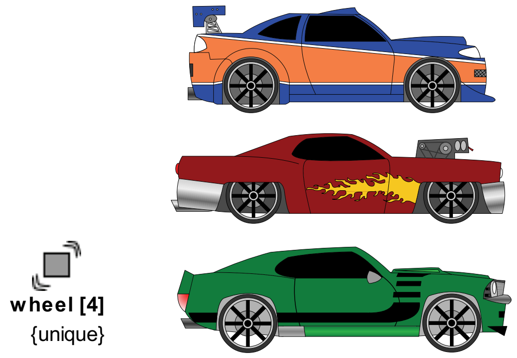
Sometimes content must be produced in different variants. Typical examples are different qualities of a video or text in different languages. For this purpose it is possible to specify *Variation Types* in MML. A Variation Type consists of a descriptive name and a list of possible values (literals). MML allows defining any custom Variation Type. A Media Component refers to a Variation Type to specify that the Media Component varys according to this type. This means that there is one variant for each value of the Variation Type. If a Media Component refers to multiple Variation Types then it has a variant for each possible value combination. Figure 5.20 shows the corresponding metamodel.

In the example in figure 5.20 a Variation Type Quality has been defined with the possible values low, medium, and high and a second Variation Type Language with the values english and german. The text HelpText refers to Language, i.e. it must be created in the two different languages English and German. The video IntroVideo should be created for the two different languages and the three different qualities, which means altogether in six different variants.

At this point MML could be extended with concepts from modeling approaches for context-sensitive applications like UsiXML (sec. 4.1.2) or Dynamo-Aid (sec. 4.1.2). Then it would be possible to describe in more detail e.g. hardware properties like screen size or different user groups. These approaches also allow to specify how the user interface changes according to the context (see sec. 4).



(a) Different cars (Media Artifacts) with different wheels (Part Artifacts)



(b) Different Cars (Media Artifacts) using a unique wheel (Part Artifacts)

Figure 5.17: Example: Individual vs. unique Part Artifacts.

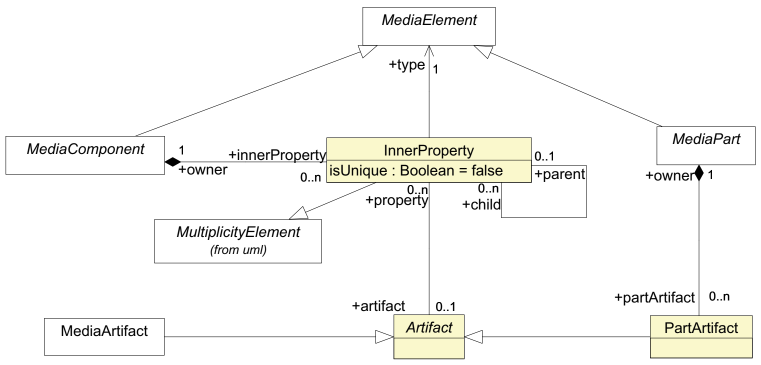


Figure 5.18: Metamodel for Inner Properties and Part Artifacts.

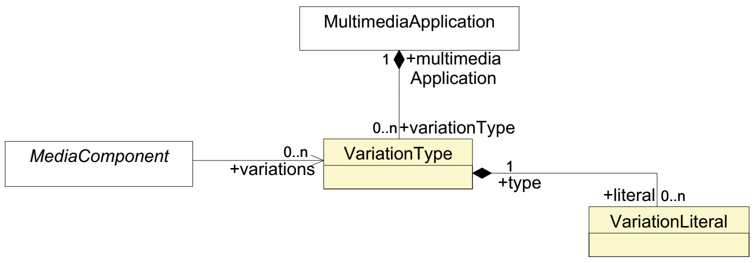


Figure 5.19: Metamodel for Variations.

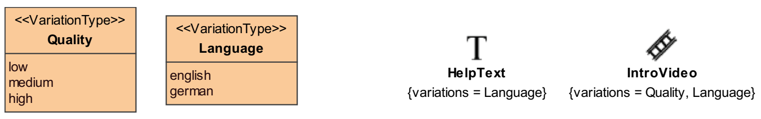


Figure 5.20: Example: Variations.

Chapter 6

MML – A Modeling Language for Interactive Multimedia Applications

This chapter gives an overview on the *Multimedia Modeling Language* (*MML*) supporting model driven development of multimedia applications. The language design follows the basic decisions and the multimedia modeling concepts introduced in chapter 5. This chapter provides a more straightforward overall view on the whole language including the modeling process and modeling tool support.

The sections on the different models as well as the subsections on the contained model elements are basically structured as follows: Rationale, Semantics, Notation , Example, Abstract Syntax, Tool Support, and Modeling Hints. To ensure readability and keep the descriptions as compact as possible, redundant or trivial passages are omitted or described only once for multiple elements. A more formal specification of the modeling language's semantics is given by the model transformation into code in chapter 7.

6.1 MML – Task Model

Most existing approaches in multimedia modeling area (see sec. 4.3) model the user interface mainly in terms of media objects. Conventional standard user interface elements, like widgets, are not considered very well. However, an interactive multimedia application usually indeed uses standard widgets – not specific media objects only. For example in the racing game, conventional standard widgets might be used to e.g. display the player's name and score and to steer the car (e.g. just by keyboard keys). Thus, it is necessary that a modeling approach supports both kinds of elements on the user interface, media objects and standard widgets.

In general, the user interface in multimedia applications is especially important – a high quality of the user interface is usually the reason for usage of different media (see sec. 2.1.3). Thus, it is essential to ensure a good usability of the user interfaces to be developed. As described in section 4.1, usability is addressed by the modeling approaches from user interface modeling area. A common concept there are Task Models which model the application from the view point of user tasks. From the Task Models it is possible to derive the required user interface elements. Thus, Task Models are supported by MML to integrate systematic user interface design into the models.

The Task Model in MML does not differ from those commonly used in user interface modeling approaches. It uses the well-known CTT notation as introduced in section 4.1.2. Figure 6.1 shows a simplified Task Diagram for the racing game example.

Originally, CTT is specified in terms of a Document Type Definition (DTD) for the XML-based

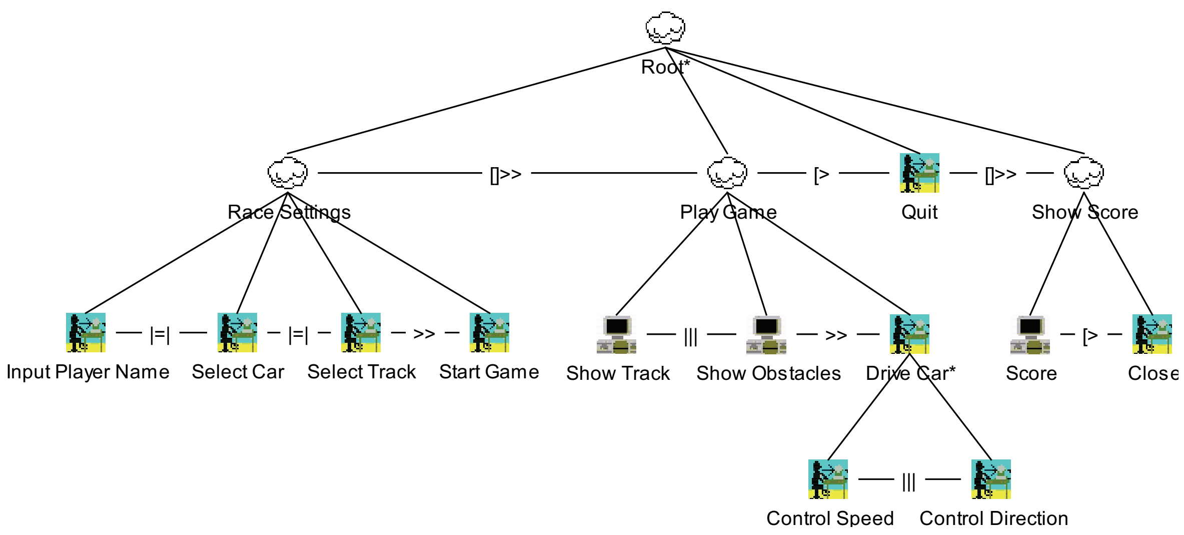


Figure 6.1: Task Diagram for the Racing Game Example.

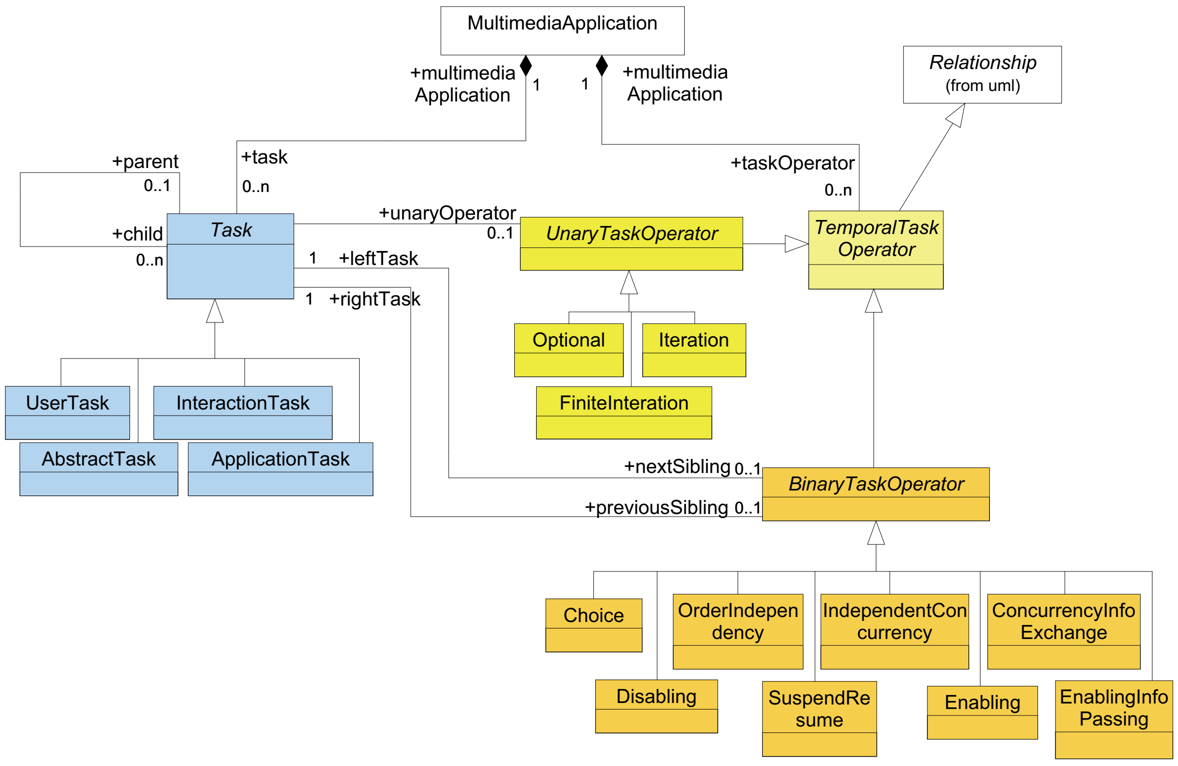


Figure 6.2: MML metamodel - Tasks

language which can be found at [TERb]. Metamodels for Task Models exist for instance in [Bastide and Basnyat06, Usi07]. The MML metamodel for Task Models shown in figure 6.2 has been made compliant to the other parts of the MML metamodel. For example, relationships have been defined as a subclass of the metaclass Relationship reused from UML as Relationship is already existing in other parts of the MML metamodel.

6.2 MML – Structural Model

This section introduces the Structural Model which defines the static application structure. In includes a Domain Model (like in most other existing approaches, see sec. 4) and in addition the Media Components as introduced in section 5.2.

Domain Model The Domain Model is specified in terms of conventional UML class diagrams. The classes usually correspond to domain concepts. To distinguish them from other kinds of classes in MML they are referred to as *Domain Classes*.

All conventional model elements from UML 2 class diagrams are supported, like *Property* and *Operation* including *Visibility* and *Parameters*, the relationships *Association* and *Generalization*, and *Primitive Types* and *Interface*.

The domain model can be mapped to program code analogously to common mappings from UML class diagrams to e.g. Java code. For the class operations only the operation signatures are generated. It is not intended in MML to define the operation bodies (i.e. their behavior) within a model for the following reasons: The domain class operations are expected to contain the application logic which can be very complex behavior. Modeling them in such a way that code can be generated would mean a kind of visual programming which can quickly lead to very large models. Furthermore, the implementation of complex effects combined with optimal performance often requires platform-specific code. In particular, the operations in multimedia applications often should cause a specific effect on the user interface. For example, in a racing game the car should move in such a way that it appears realistic to the user. It requires much fine-tuning to provide the optimal level of difficulty for the user so that the application is really appealing, challenging, and entertaining. It is often very difficult to specify such behavior in advance within a model – in contrary, often the optimal algorithms and parameters have to be found out incrementally by multiple test runs. Thus, the operation bodies are preferably implemented directly within an appropriate development environment (like the Flash authoring tool) where they can be directly executed and tested.

The domain model is denoted like conventional UML class diagrams. Optionally the domain classes can be marked with a keyword DomainClass but this is usually not necessary as all classes without specific notation or keyword are interpreted as domain classes. Figure 6.3 shows as example the domain classes for the Racing Game application.

Figure 6.4 and 6.5 show the corresponding parts of the metamodel. They are extracts from the UML 2 metamodel, restricted to those elements supported by MML. Thereby the original structure has not been changed to enable compliance with other approaches and tools, e.g. UML modeling tools.

Media In addition, the Media Components are defined in the Structural Model. All model elements elaborated in section 5.2 can be used, i.e. *MediaComponent* (sec. 5.2.2), *MediaArtifact* (sec. 5.2.5), *MediaPart* (sec. 5.2.7), *InnerProperty* (sec. 5.2.8), *PartArtifact* (sec. 5.2.9), *VariationType* and *VariationLiteral* (sec. 5.2.10). A Media Component can refer to an *Interface* by an *InterfaceRealization*

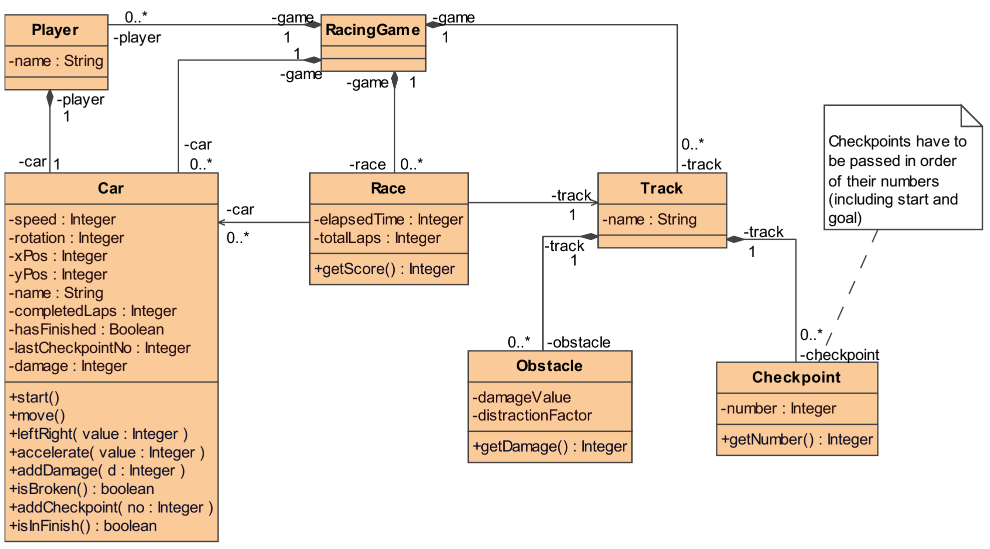


Figure 6.3: Domain Classes for the Racing Game example.

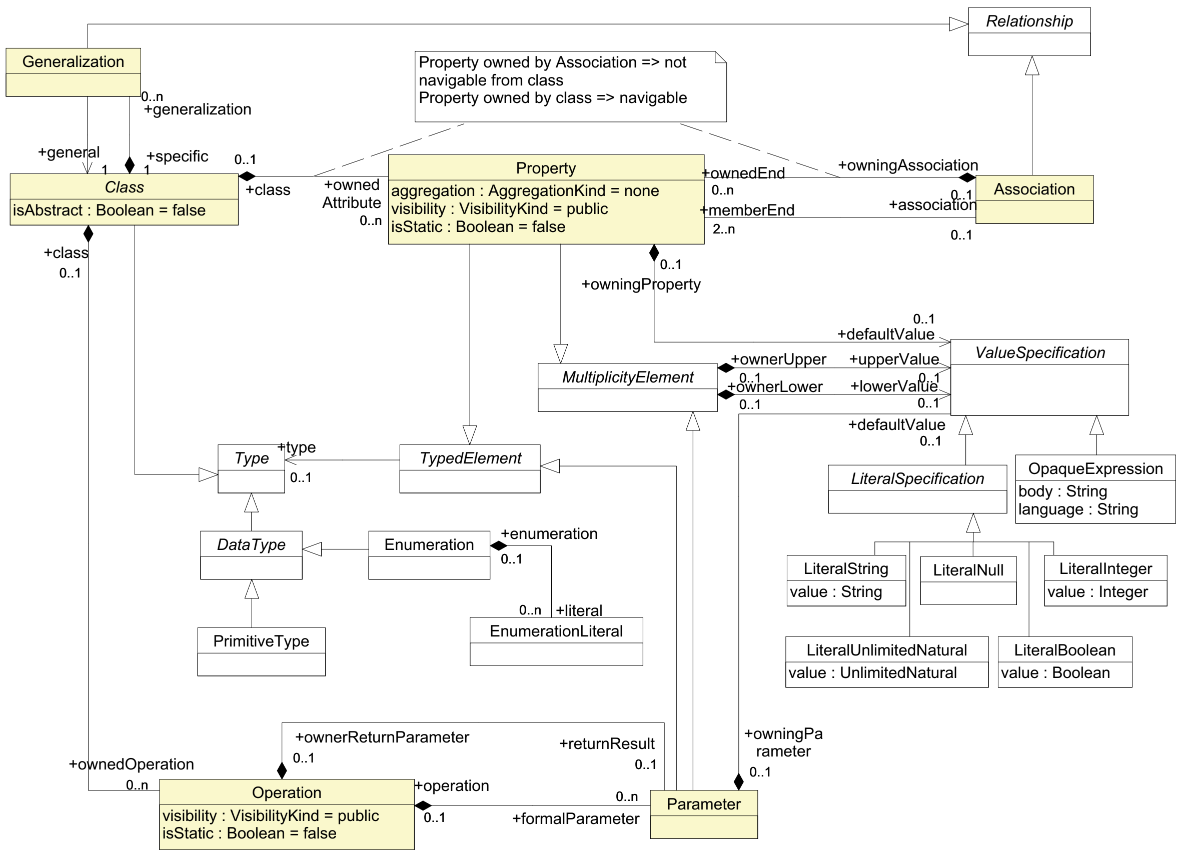


Figure 6.4: MML metamodel for classes reused from UML.

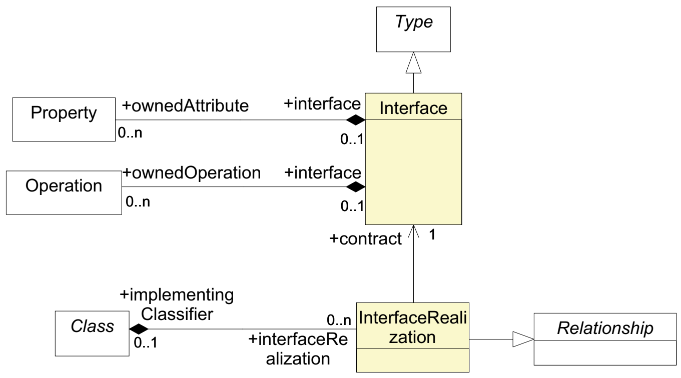


Figure 6.5: MML metamodel for interfaces reused from UML.

relationship (sec. 5.2.3). The Media Components represent the Domain Classes which is specified by the *MediaRepresentation* relationship (sec. 5.2.4). It is possible to assign one of keywords defined in section 5.2.6 to a Media Representation.

The example in figure 6.6 shows the complete Structural Model for the racing game application, i.e. the domain model from fig. 6.3 enhanced with Media Components. In the example, the domain class Car is represented by an animation CarAnimation and by EngineSound. The EngineSound in particular represents the car's property speed. The Keyword unique at the Media Representation between EngineSound and Car specifies that the same audio artifact is used for all cars. The Media Component EngineSound also provides an additional interface containing an operation setPitch. An inner structure is defined for CarAnimation and TrackAnimation. Thereby, inner properties of TrackAnimation refer to the Media Components ObstacleAnimation and CheckpointAnimation. TrackAnimation is manifested by three Media Artifacts Monza, Indianapolis, and Monaco. CarAnimation is manifested by Porsche and Ferrari. All animations should be created in two different qualities low and high as specified by the Variation Type Quality.

Figure 6.7 shows the part of the MML metamodel defining the overall structure of the MML Structural Model. As explained in section 5.1.5 the top-most model element in the containment hierarchy is MultimediaApplication. DomainClass is defined as a subclass of UML class. They are connected to Media Components via MediaRepresentation relationships like introduced in section 5.2.4.

Figure 6.8 shows the metamodel part which further defines Media Components. It is the composition of the different metamodel extracts elaborated in chapter 5.2. In addition, the metamodel in figure 5.12 defines the subclasses for the abstract metaclass *MediaPart*. Figure 5.12 can be added as it is and is thus not depicted here again.

6.3 MML – Scene Model

The Scene Model specifies the application's behavior in the large in terms of Scenes. The term *Scene* originates from multimedia domain and refers to a state of the application associated with a specific Presentation Unit. The Scene Model is similar to the Navigation Model in Web Engineering (sec. 4.2) or some (coarse-grained) Dialog Models in user interface modeling approaches (see sec. 4.1). The Scene Diagrams specifies the application's Scenes and the transitions between them. Therefore MML uses the concepts from State Charts, similar to OMMMA (sec. 4.3.2) and many user interface modeling approaches, e.g. Dynamo-Aid (sec. 4.1.2). A multimedia-specific difference to other approaches is the dynamic character of Scenes. Thus, Scenes are more generic than conventional Presentation

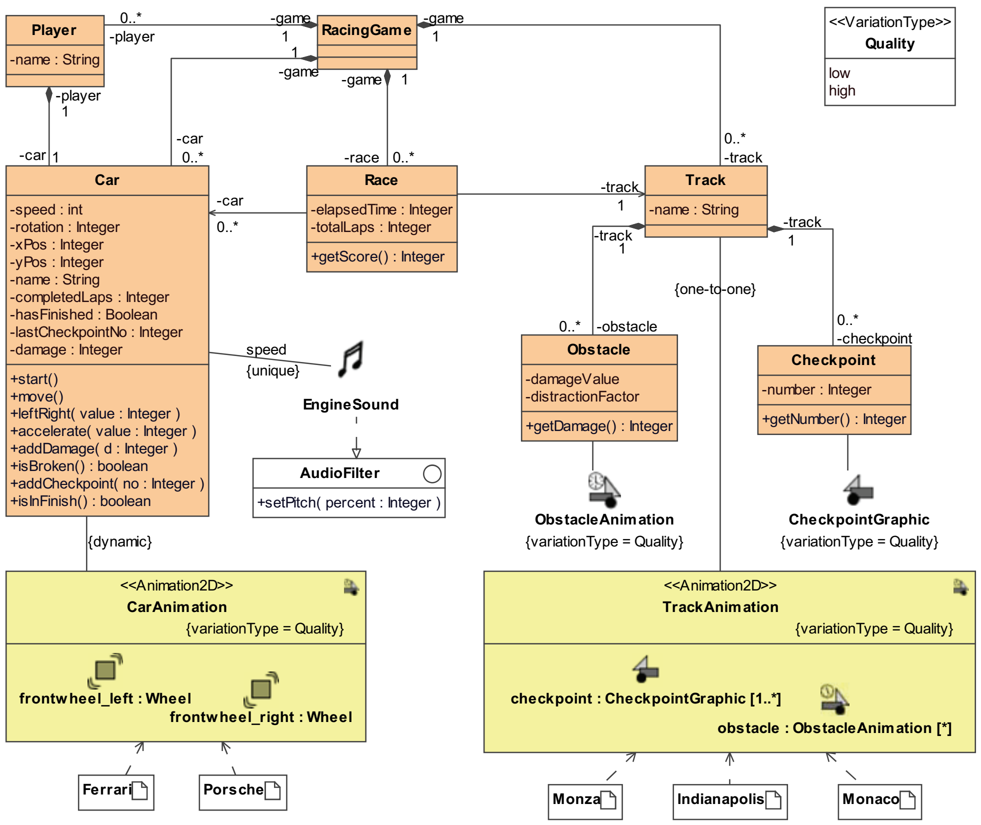


Figure 6.6: Complete Structure Diagram for the Racing Game example

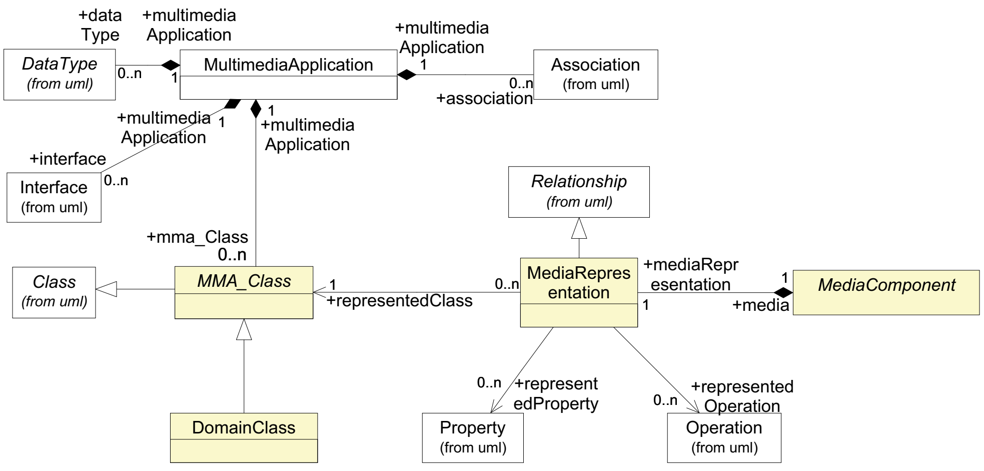


Figure 6.7: MML metamodel for MML Structure Diagram Elements.

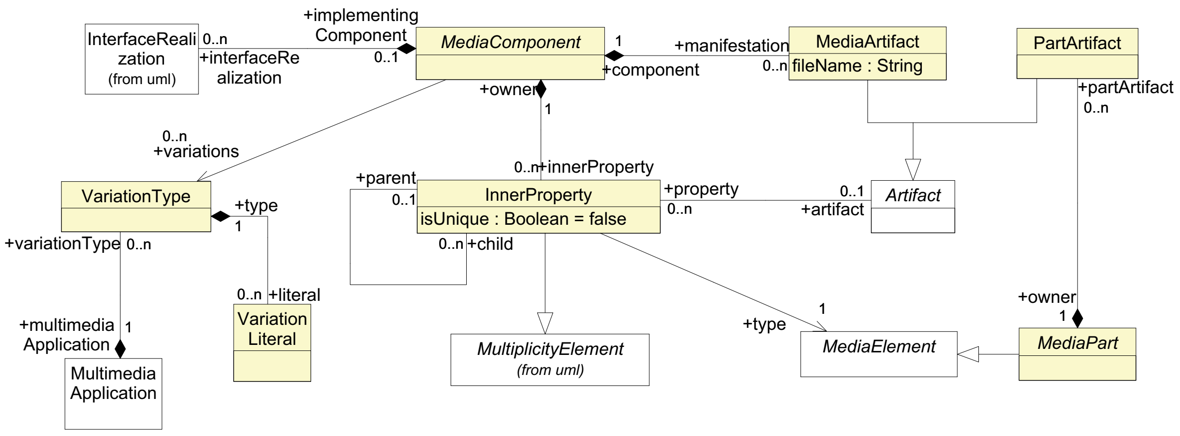


Figure 6.8: MML metamodel for Media Components in the Structure Diagram.

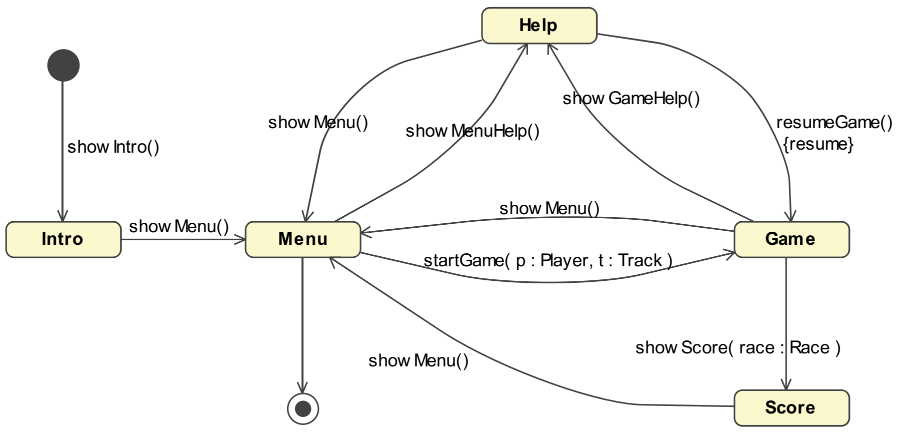


Figure 6.9: MML Scene Diagram.

Units and can receive parameter values at runtime.

Scene A Scene represents a specific state of the application. It is associated with a Presentation Unit. However, an important multimedia-specific aspect of Scenes is their highly dynamic character. This is caused on the one hand by the time-dependent behavior of temporal media instances in a Scene. On the other hand, often the number, the position, and the presented artifacts of Media Components is calculated at runtime. For example in a racing game the number of cars, the car types and the track might be selected by the user at runtime. Often it is not possible or useful to specify all possible configurations by a Scene of its own. Instead, a Scene is more generic and can receive parameter values.

Every Scene is associated with exactly one (probably generic) Presentation Unit specified in the MML Presentation Model (sec. 6.4). In contrast to OMMMA (sec. 4.3.2) the Scene Model specifies only the application's top-level behavior while the internal behavior of Scenes is specified in Interaction Diagrams (sec. 6.5).

Scenes are denoted like a states in UML State Charts. Figure 6.9 shows an example Scene Diagram for the racing game application. It contains the Scenes Intro, Menu, Help, Score, and Game.

In this example the Scene Game is generic and represents all possible races or levels the user can play. In particular, when the different tracks, cars and configurations for a single race are loaded dynamically it would not be useful to model each configuration as a scene of its own. Nevertheless, it is of course still possible to model a scene of its own for each race if desired; for instance, if the developers want to specify that there is always a fixed order of tracks (e.g. the user always has to play the track Monza first, then the track Indianapolis, and finally Monaco).

Entry Operations, Exit Operations, and Transitions Due to its generic nature a Scene provides *Entry Operations*. They can be used to initialize the Scene and pass parameters to it. During the final implementation (after code generation), Entry Operations can be used in particular to initialize the Scene's associated Presentation Unit. This might include setting up media instances as well as establishing event listeners and connections between domain classes and user interface elements. The Entry Operations also trigger associated behavior (according to the Interaction Diagrams in section 6.5), e.g. invoke predefined behavior of contained Media Components.

Furthermore, each Scene has *Exit Operations* which can be used to clean up the Scene (e.g. to free resources), call an Entry Operation of the next Scene, and pass parameters to it.

Transitions are defined between a source Scene and a target Scene analogous to transitions between states in UML State Charts. Basically, executing a transition means in MML that the target Scene becomes active and its associated Presentation Unit is displayed while the source Scene becomes inactive and its Presentation Unit is no longer visible. Moreover, a Transition corresponds to the execution of an Exit Operation in the source Scene followed by the execution of the Entry Operation in the target scene. Each Exit Operation is associated with exactly one Transition while an Entry Operation can be targeted by multiple Transitions.

In the Scene Diagram a Transition is denoted as directed arrow leading from a source Scene to a target Scene. A Transition is annotated with the name of the Entry Operation it addresses. The Exit Operations need not to be specified explicitly in the diagram as a Scene has an Exit Operation for each outgoing transition. As long as no name has been specified explicitly the Entry Operations can be referenced by definition by a default name: The name of an Exit Operation is composed of the prefix exitTo, the name of the target scene, and the name of the target Entry Operation, separated by '_', e.g. exitTo_Menu_showMenu().

In the example in figure 6.9, the Transitions specify that first the Scene Intro is shown followed by the Scene Menu. From the Menu it is possible to proceed either to the Help or to the Game. From the Game it is possible to call the Help as well. When the Game is finished the Score is shown and afterwards the application returns to the Menu.

In the example the Game's Entry Operation start is used to pass parameters from the Menu to the Game Scene. The Scene Help provides two different Entry Operations showGameHelp and showMenuHelp which can be used to show different help content and to return to the foregoing Scene afterwards. According to the naming convention for instance the Exit Operation which belongs to the transition from Intro to Menu is named as exitTo_Menu_showMenu().

Start State, End State A Scene Model has exactly one *StartState* and one *EndState* specifying the application's start and its termination. They are denoted like initial states and final states in UML providing specific kinds of Entry/Exit Operations. By definition, the Start State owns the operation *ApplicationStart* which is automatically called when the application starts and triggers the Entry Operation of a Scene as specified by the transitions in the Scene Model. The End State owns an operation *ApplicationExit* which causes the application to terminate.

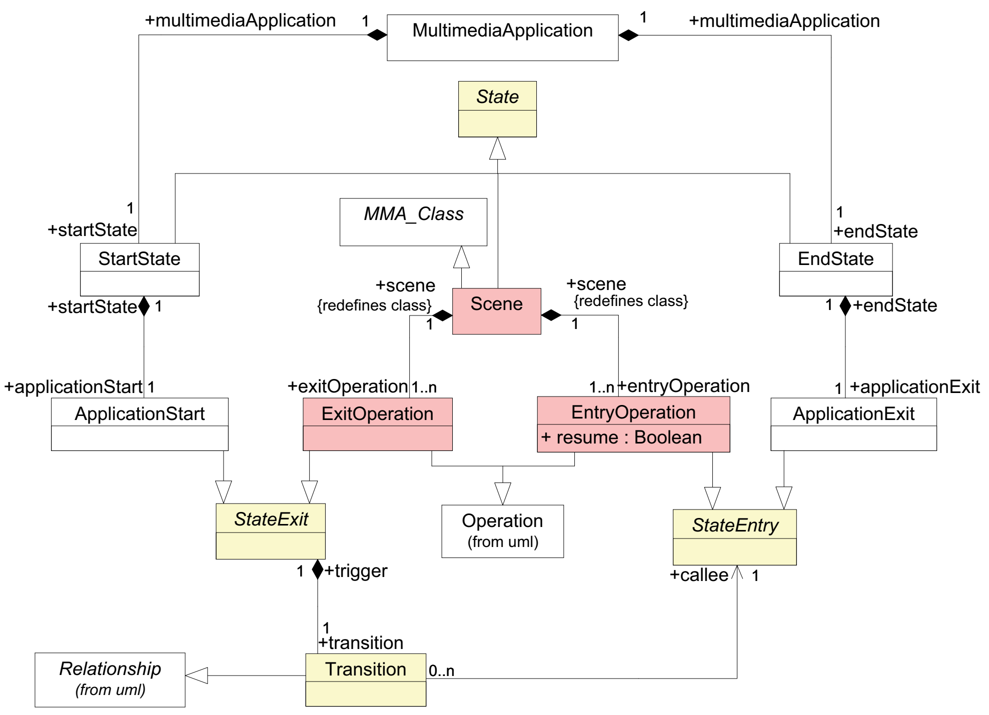


Figure 6.10: Metamodel for MML Scene Diagram elements.

In the example in figure 6.9 the Start State calls the Entry Operation showIntro of Intro. From the Menu it is also possible to exit the application.

Resuming Scenes As proposed by OMMMA (sec. 4.3.2) the Scene's dynamic properties require to distinguish between two different cases when entering a Scene: When a Scene has already been active before it is possible to either initialize the Scene again or to return to its previous state. The former case is the default case. The latter case is useful when a Scene has an inner state (e.g. resulting from usage of temporal media) which should be resumed later. For example in the racing game the Scene Game can be interrupted by calling the application's Help and should be resumed afterwards.

Thus, it is possible to specify for an Entry Operation whether it resumes to the Scene's previous state or not. If a Scene is left by an Exit Operation which might (according to the Transitions in the Scene Model) lead to an Entry Operation with resume set to true then the Scene instance is stored. When the Scene is then entered again by an Entry Operation with resume set to true then the stored previous instance is used instead of creating a new instance. If no previous instance of the Scene exists then a new instance is created, i.e. the value of resume has no effect.

In the diagram the value of resume is specified as true by the attaching the keyword *resume* to the corresponding Entry Operation[1]. It should be mentioned that such an example is modeled in the original statechart formalism by a *throughout* mechanism ([Harel87], p.26) which is a possible alternative for future versions of MML.

In the example in fig. 6.9 the Game's Entry Operation resumeGame() is marked with the keyword resume. In contrast, the Game's Entry Operation startGame(p:Player, t:Track) creates a new game and initializes it with (new) parameters.

[1]OMMMA uses History States for this purpose as OMMMA restricts to existing UML State Chart elements.

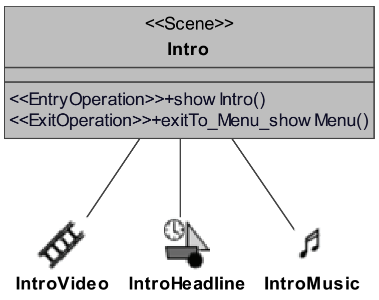

<<Scene>>
Intro
<<EntryOperation>>+show Intro()
<<ExitOperation>>+exitTo_Menu_show Menu()
IntroVideo
IntroHeadline
IntroMusic

Figure 6.11: Class representation of Scene Intro.

Figure 6.10 shows the resulting metamodel for the model elements in Scene Models. MML Scene Models have a more restricted structure than UML State Charts and contain only MML-specific model elements. Thus, none of the UML metaclasses is reused here. The basic concept defined by abstract metaclasses is that a State owns a StateEntry and a StateExit. A StateExit owns a Transformation which refers another StateEntry. In case of a Scene, the StateExits must be ExitOperations and the StateEntrys must be EntryOperations. The metaclass EntryOperation has a boolean attribute resume.

Class Character of Scenes Media Components usually represent Domain Classes. But in some cases a Media Component has no relationship with a Domain Class but rather represents a specific Scene. A typical example is help content like a text document. Although some parts of it certainly refer to Domain Classes the document as a whole only represents the Scene Help itself. Likewise, a Video for the application's intro usually represents not a specific Domain Class but the Scene Intro. Thus, a Scene can be represented by a Media Component just like a Domain Class.

Moreover, a Scene can own properties and operations like a class:

- As described above, a Scene owns Entry Operations and Exit Operations.
- It is possible to define additional operations for a Scene encapsulating some additional behavior (see sec. 6.4 and sec. 6.5).
- In context of the Presentation Model properties representing Domain Objects and Sensors are assigned to Scenes (see sec. 6.4).

Thus, the character of a Scene is two-fold: on the one hand it can be interpreted as an application's State (regarding the application's coarse-grained behavior defined in the Scene Model), on the other hand it can be interpreted as a class owning properties and operations. During code generation a Scene is mapped to a class associated with a user interface. Properties and operations owned by the Scene are mapped to class properties and class operations.

To gain an overview on all a Scene's properties and operations it is possible to denote it as a specific kind of class in the Structural Model. A Scene can be denoted in the Structural Model like a conventional class. It is marked with the keyword Scene to distinguish it from other kinds of classes in MML. Entry Operations and Exit Operations are then also marked with keywords EntryOperation and ExitOperation to distinguish them from other (conventional) operations a Scene may own. All properties are denoted like class attributes.

Figure 6.11 shows an example: The Scene Intro is represented by three Media Components: IntroVideo, IntroHeadline, and IntroMusic. According to the Scene Model from figure 6.9 it owns an EntryOperation show Intro() and an Exit Operation exitTo_Menu_showMenu().

In the metamodel in figure 6.10 the metaclass Scene is defined as subclass of MMA_Class and thus inherits its properties. Additional properties and operations for Scenes are defined in Presentation Model and the Interaction Model explained in the following sections.

Modeling tools usually also provide a containment tree view listing all model elements in a tree structure. Of course, this view can also be used to look up all properties of a Scene. Usually, it is also possible to drag model elements from the containment tree into a diagram. A modeling tool should thus support that Scenes can be dragged into an MML Structure Diagram where they are visualized like in figure 6.11.

Composite Scenes It is also possible to define Composite Scenes in MML analogous to Composite States in UML State Charts. This is mainly useful to specify that some parts of the user interface elements should be available throughout multiple scenes (like in frame-based HTML pages). In that case the user interface results from the unification of the Presentation Unit of the Sub-Scene and the Presentation Unit of the Composite Scene. It is also possible to specify regions to specify that two Scenes are active concurrently. This corresponds to e.g. two independent windows in a graphical user interface. However, it is intended to use Composite Scenes sparsely to keep the models simple. Also nested frames and multi-window applications are currently not that common in many multimedia application domains and only little supported by authoring tools like Flash. For that reasons Composite Scenes are not discussed here further in detail.

Future Extensions As mentioned in section 4.1 it is possible to calculate Enabled Task Sets (ETS) from the task models which help to identify the Presentation Units. All tasks within an ETS must be provided within the same Presentation Unit. [Luyten04] for example shows a concrete algorithm to calculate the ETS and to derive a State Transition Network with ETS as nodes, transitions between them as well as initial and finishing states. It then still has to be decided which ETSs to put together in a single Presentation Unit (as a Presentation Unit often contains multiple ETS) and how to map the tasks to user interface elements. Nevertheless, automatic derivation can provide a useful starting point for the developer and also helps to validate the created results. Analogously such a transformation could be applied to MML to derive a starting point for the Scene Model from the Task Model. Integrating such a transformation for would thus be a possible enhancement for future version of MML but has not been implemented yet.

6.4 MML – Presentation Model

The MML Presentation Model specifies the user interface of multimedia applications. The basic concept is the Presentation Unit which represents an abstraction of a top-level user interface container, e.g. a window. As described in the foregoing section (sec. 6.3) there is one Presentation Unit for each Scene.

The Presentation Model integrates systematic user interface design with the media design. As discussed in section 6.1 a modeling language for multimedia applications needs to consider the concepts from user interface modeling area for systematic user interface design. For this reason MML supports Task Models from which an Abstract User Interface Model can be derived. The Abstract User Interface Model specifies Presentation Units containing Abstract Interaction Objects. The Abstract Interaction Objects adhere to the user tasks to be supported by the application and reflect the user interface design. They are platform- and modality-independent. The MML model elements for this Abstract User Interface Model are presented in section 6.4.1.

In a second step, the user interface design and the Media Components have to be integrated. Section 6.4.2 shows how this is supported in MML by UI Realization relationships to instances of Media Components. At this point the user interface model is no longer independent from modality as the modality of each media component is already determined by its media type. However, as pointed out in section 6.1 the user interface of a multimedia application usually contains standard user interface elements as well – not media objects only. For those standard elements it is still possible to select between different modalities. For example, it is possible to present the user some piece of information by audio instead of by a visual component, e.g. the lap time after each lap in the racing game.

In a third step, *Sensors* are added to the Presentation Model. They represent specific events resulting from temporal media objects. These model elements are introduced in section 6.4.3.

Finally, it is possible to refine the Presentation Model in an additional step in terms of a Concrete User Interface Model, as common in user interface modeling. It specifies the user interface in terms of Concrete Interaction Objects for a specific modality. The Concrete Interaction Objects are concrete (but usually still platform-independent) widgets derived from the Abstract Interaction Objects.

The current version of MML concentrates here on the scope defined in section 5.1.1. Thus, it restricts on some basic concepts from user interface modeling to clearly demonstrate the integration with media objects and enable code generation. The transformation from Task Models into the Abstract User Interface as well as the Concrete User Interface model are not further investigated here.

6.4.1 Abstract User Interface

The first step when modeling the Presentation Model is to define the Abstract User Interface to support a systematic user interface design. It specifies the Presentation Units which are – according to section 6.3 – associated with Scenes.

In the diagram the abstract user interface is denoted within a top-level container representing the Scene. It contains another container for the Presentation Unit which in turn contains the AIOs (see figure 6.14).

Abstract Interaction Objects: Types An *Abstract Interaction Object* (*AIO*) is a platform- and modality-independent abstraction of user interface widgets like a button, a text field, etc. The set of AIOs provided by existing user interface modeling approaches varies. Many of them, like [da Silva and Paton03, Van den Bergh06, Görlich and Breiner07], provide a small set of simple, very abstract AIOs like Input Component, Output Component, etc. Another possibility would be a faceted approach like in UsiXML (sec. 4.1.2) or [Trætteberg02] where each AIO is more generic and has multiple facets like input, output, navigation, and control. This approach is more flexible but the elements and their notation becomes somewhat more complex. Others like Canonical Abstract Prototypes [Constantine03] use more fine-grained distinction and provide various AIOs for different kinds of actions, like *move*, *duplicate*, *modify*, etc. Then, the models can become more expressive but is also more difficult to learn the different AIOs. Such a fine-grained distinction is not mandatory as the purpose of an AIO becomes apparent also by its name and by the domain object, property, or operation it refers to (see UI Representation relationship below).

MML sticks to the simplest solution and provides a small set of AIOs: An *Input Component* enables the user to freely input data like a text field. An *Output Component* shows information to the user like a text label. An *Edit Component* enables the user to edit data like a slider. It is a combination of Input Component and Output Component. An *Action Component* enables the user to trigger an action of the system like a button. A *Selection Component* enables to select an element from a given

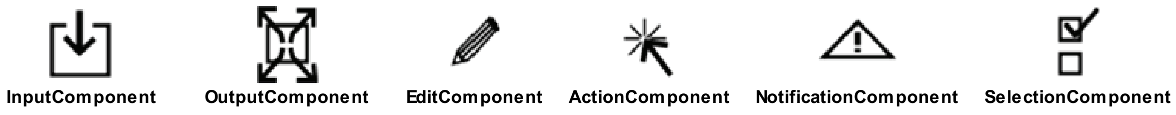


Figure 6.12: Icons for Abstract Interaction Objects.

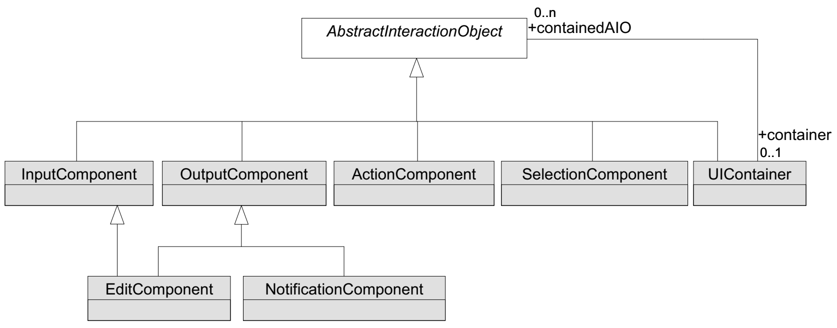


Figure 6.13: Metamodel for MML Abstract Interaction Objects.

set of choices like a drop-down listbox. A *UI Container* is a container for any kind of AIOs and is used to structure the AIOs within the Presentation Unit like a panel. Finally, MML provides in addition a *Notification Component* which is a specific case of Output Component but is more conspicuous than a conventional Output Component. By default it is not visible to the user and is only displayed during runtime like a dialog box, e.g. after an event of a temporal media object. For instance, in the racing game application a Notification Component can be used to notify the user when the race is finished which can be implemented for example as a text banner or an animation.

Figure 6.12 shows the visual notation for AIOs in MML developed as described in section 5.1.4.

An example is shown in figure 6.14. It shows the Presentation Unit for the Scene Game. It contains Output Components for information like the player's name, the laps already completed, etc. Also the track and the obstacles on the track must be shown to the user and are represented by an Output Component. The car is shown to the user but in parallel the user can edit some parameters of the car like its speed and rotation. Thus, car is defined as an Edit Component. There are also Action Components to trigger functionality like starting the race or call the help.

Figure 6.13 shows the metamodel part for the AIOs available in MML. Abstract Interaction Object is defined as an abstract superclass.

Extending MML with additional AIOs is relatively easy possible by adding an additional subclasses to the metamodel and adding an additional rule in the ATL transformation (for the mapping from MML models to code, see chapter 7).

Abstract Interaction Objects: Properties For each AIO it is possible to specify that the AIO is initially invisible on the user interface. This can be useful for multimedia user interfaces where sometimes elements become visible only in specific situations, e.g. when an animation appears on the screen at a specific point of time. For code generation this means that the user interface element is created but not initially visible on the user interface, e.g. by setting a corresponding parameter or by putting it outside the visible screen area. The most common case is the Notification Component which is invisible by default. All other AIO types are visible by default but can be defined as invisible as well. In case that the AIO should be realized by another modality like sound it might have no effect. An invisible AIO can be denoted in the diagram by a dashed bounding box around the element.

An AIO in the Presentation Diagram can have a multiplicity. Multiplicities are very useful in context of multimedia applications as the user interface is often dynamic and number and placement of user interface elements can depend e.g. on user input. For example, there might be a varying number of obstacles on the track in a racing game An AIO with a multiplicity corresponds to a set of AIOs whose names are composed of the AIO name specified in the model and a consecutive number. During code generation it has to be decided how many AIOs to initially instantiate on the generated user interface. This can be done according to the following rule: For a multiplicity n..m, m elements are generated if m is a number (i.e. not '*'). Else, n elements are generated if n is greater than zero. If the multiplicity is 0..* then three example elements are instantiated. Multiplicities are denoted behind the AIO's name.

An example can be found in figure 6.14: A multiplicity is specified for the Output Component obstacle as there is a varying number of obstacles. According to the rule described above three obstacles will be generated from the model named as obstacle1, obstacle2, and obstacle3.

Domain Objects and UI Representations Like in most user interface modeling approaches an AIO is associated with the application logic. This is specified by UI Representation relationships between AIOs and domain objects.

A *Domain Object* represents an instance of a Domain Class from the Structure Model (sec. 6.2). More precisely, it is a placeholder for an instance having a specific role within the Scene (not a concrete instance with concrete values defined in the model)[2]. They can be interpreted as properties of the Scene and are mapped to properties of the Scene class during code generation.

The domain objects are denoted as a solid-outline rectangle containing its name and, separated by colon, its type, i.e. the name of the domain class it instantiates.

A *UI Representation* relationship specifies that an AIO represents a domain object. Analogous to Media Representation relationships (sec. 5.2.4) it is possible to specify concrete properties and/or operations of the domain object which are represented by the AIO. The detailed meaning of the UI Representation differs according to the AIO type: An Output Component represents either a property or an operation, i.e. its return result. An Input Component refers to a property or an operation which implies that it sets the value of the property or of the operation's parameters. An Action Component usually refers to an operation which means that the Action Component triggers the operation when it is activated by the user. A specific case is the Selection Component as it consists of multiple choices (which basically could be interpreted as kind or output elements). Thus, a Selection Component refers either to a property with multiplicity greater than one or to an operation which returns a collection of elements.

In general, the relationship between AIO and represented domain object is always 1:1 in MML. If no specific property or operation is specified, it means that it is either not decided yet or that the AIO represents the domain class as a whole.

The UI Representations are denoted as a solid line between an AIO and a Domain Object. It can be annotated with the names of Domain Object's properties and/or operations represented by the AIO. Sometimes an AIO represents a property or operation of the containing Scene itself, like an Exit Operation of the Scene. In this case no UI Representation relationship is depicted in the diagram. Instead, the name of the represented property or operation is denoted in curly brackets below the AIO.

[2]One can distinguish between three layers of abstraction: 1) the "object level" describes properties for concrete objects with concrete values 2) the "class level" describes properties which hold for *all* instances of a class. The "role level" resides between those two and describes properties which hold for all instances of a class in a *specific context*. For example, class attributes reside on role level as a property defined for a class attribute holds for all objects referenced by the attribute. See [Hitz et al.05]

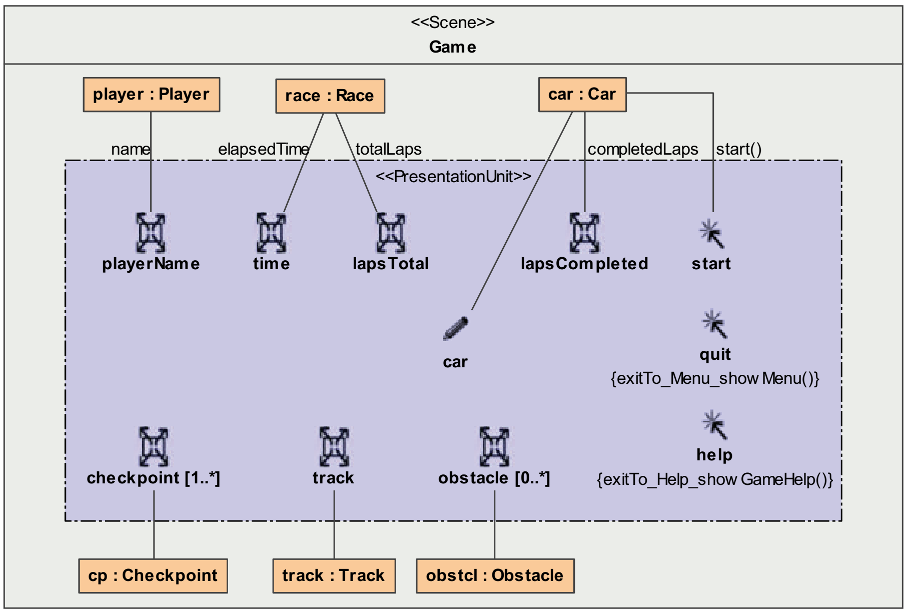


Figure 6.14: MML Presentation Diagram containing the Presentation Unit for the Scene Game.

Figure 6.14 shows a Presentation Diagram for the Scene Game from the racing game example. For instance, the Edit Component car, the Output Component lapsCompleted, and the Action Component start all represent an instance car of the class Car. Thereby, lapsCompleted represents the class attribute completedLaps, start represents the operation start(), while the EditComponent car represents the whole car object (i.e. multiple properties not further specified in the model). The Action Components help and quit all represent operations of the Scene itself.

The metaclasses for the Abstract User Interface will be shown later in figure 6.16 together with the metaclasses for UI Realizations.

Abstract Layout Existing user interface modeling approaches usually support to define the layout of the AIOs on the user interface in an abstract way as well. It can be specified in an abstract way by constraints on the AIOs and by relationships between them. For example, one can define that a user interface element should always have a quadratic shape or that one element should be positioned left to another. A discussion can be found e.g. in [Feuerstack et al.08].

For approaches which address independence from modality it is important to specify the abstract layout independent from modality as well. For instance in a vocal user interface there are no spatial relationships like "left-to". Thus, they use more abstract relationships between the models. [Limbourg04] uses the 13 basic relationships identified by [Allen83]. These relationships originally define the temporal relation between to time intervals, e.g. "before", "starts with", "overlays", "during", etc. They can also be applied to the visual dimension (e.g. one visual user interface element is placed before another). Moreover, by composition they can also be applied to multiple dimensions like 2D space or 3D space.

An alternative approach is to specify the designer's intention instead of resulting physical layout properties: for instance "grouping" , "ordering", "hierarchy", "relation". For example, "grouping" user interface elements can be realized on a visual user interface e.g. by "lining them vertically or horizontally, or with the same colour or with bullets" [Paternò and Sansone06].

In the current MML version no layout mechanisms are implemented. When generating code for an authoring tool like Flash the designer can specify the layout visually in the authoring tool. However, MML can easily be extended e.g. with abstract layout relationships like specified in UsiXML [Usib].

6.4.2 UI Realizations

Until now, the Abstract User Interface has been specified like in user interface modeling approaches. In conventional applications they would then be implemented by standard widgets. However, in multimedia applications the AIOs can also be realized by instances of the Media Components instead. MML enables to specify this by *UI Realization* relationships between AIOs and Media Instances.

Media Instance A *Media Instance* is an instance of a Media Component from the Structure Diagram (sec. 5.2). Analogous to AIOs and Domain Objects, the Media Instances are interpreted as properties of the Scene as well. Although in fact (during implementation) a Media Instance is always an instance of a concrete Media Artifact, the Media Instances in MML Presentation Diagrams are specified more abstractly as instances of Media Components. This is necessary as the concrete Media Artifact might be selected at runtime. However, it is optionally possible to specify a specific Media Artifact name for the Media Instance.

The Media Instances are denoted like Media Components but in addition with an instance name, separated by a colon, before the name of the Media Component. Thereby, the different alternatives to depict a Media Component are all allowed, e.g. collapsed or with a compartment showing its inner structure (see sec. 5.2.2).

UI Realization A UI Realization relationship specifies that an AIO is realized by a Media Instance. It means that the Media Instance fulfills the AIOs role on the user interface. Thus, during code generation such an AIO is not mapped to a standard widget but is implemented by an instance of a Media Component on the user interface. Other AIOs which are not realized by Media Components represent conventional user interface elements outside of the customer's or designer's multimedia-specific visions. They are mapped to conventional standard widgets during code generation.

A UI Realization is denoted as a dashed arrow from a Media Instance to an AIO.

Figure 6.15 extends the example from figure 6.14 with UI Realizations. For instance, the Edit Component car is realized by the Media Instance car which instantiates the Media Component CarAnimation and by an instance of EngineSound. The Output Component track is realized by an instance of TrackAnimation.

UI Realizations: Media type vs. AIO type The most obvious case is that a Media Instance realizes an Output Component. But in interactive applications Media Instances can also be used for user interaction. For example, it can be possible to select a specific region in an image or to drag an animation with a pointing device. It depends on the media type whether and how a media instance may act as an interactive AIO, i.e. as an Input Component, Edit Component, Selection Component or Action Component. Audio can usually not act as an interactive AIO as it can not be manipulated. Of course it is possible to record and parse audio using a microphone as accomplished in speech recognition. However, this has no relationship at all to playing an auditive media object (see distinction between the terms "multimedia" and "multimodality" in sec. 2.1). The same holds for video, where a camera and gesture recognition are necessary for user inputs. However, all visual elements, including

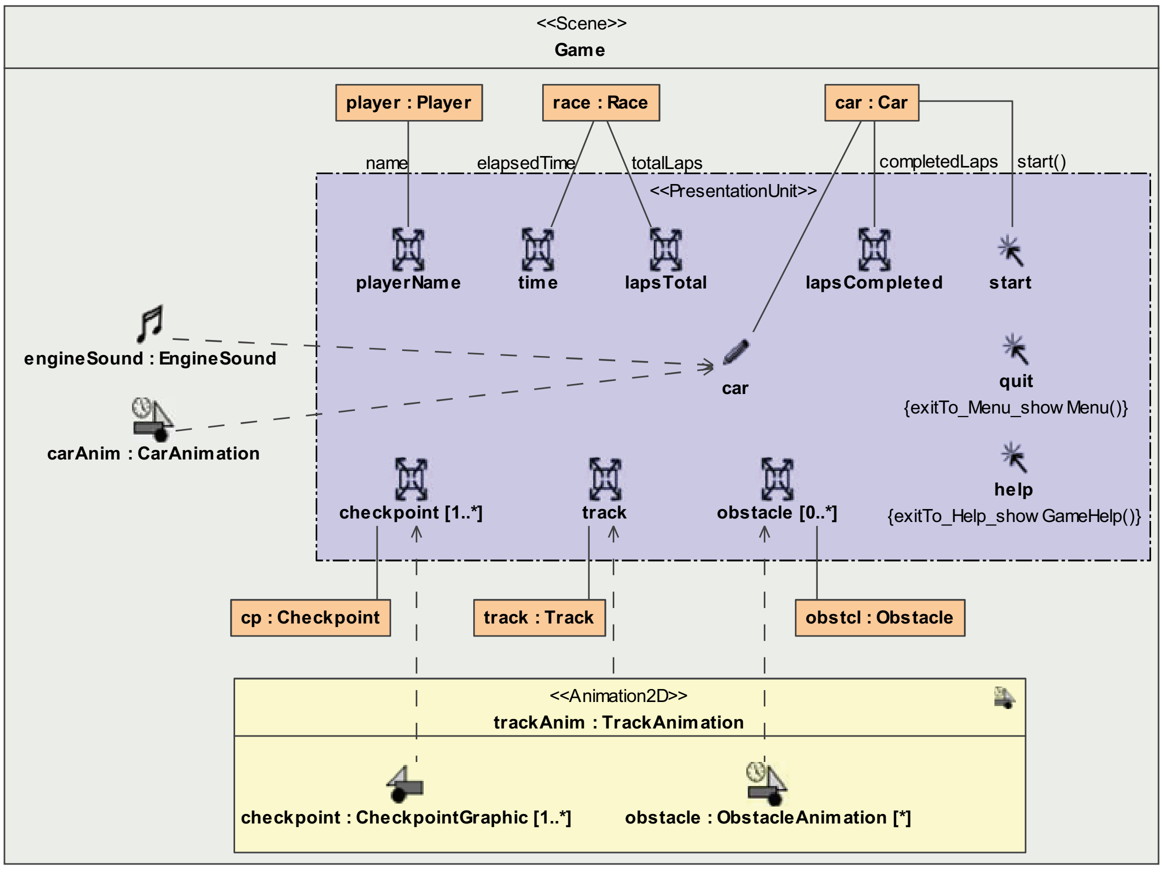


Figure 6.15: MML Presentation Diagram including UI Realizations for the Scene Game.

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 Components.

As animations can dynamically change their content dependent on the application logic, they can additionally act as Edit Components or Input Components. An example is a car animation which represents e.g. the current rotation of the car. The user might manipulate the animation with an input device to edit the car's rotation value.

All visual objects can also act as Selection Component. As mentioned above Selection Components are more complex as they consist of multiple choices. A visual Media Instance can act either as the Selection Component as a whole or can represent a single choice. The former case requires that the Media Instance consists of several parts which act as choices, for instance an image of a map where different image regions can be selected. The latter case implies that there are multiple Media Instances – one for each choice. The distinguish between this two cases in the model and for code generation the latter case is specified by assigning the keyword *choices* (denoted in curly braces) to the UI Realization.

UI Realizations of Inner Properties As defined above the UI Realization relationship is specified between a Media Instance and an AIO. However, it is also possible that a specific part of the Media Instance actually realizes the AIO. For example in a graphic representing a map, it might be useful that a click on a specific region in the map triggers an event. In the racing game example, it might for instance be possible to select the kind of tires (rain tires or dry-weather tires) for a car by clicking on the wheels in the car animation.

In case of an Action Component, the part for instance triggers an event. In case of an Edit Component, the part can for instance be manipulated by drag and drop. In case of a Selection Component, the part can be selected.

In the Presentation Model this is specified by attaching a reference on the Inner Property (referring to a specific part of a Media Component, see sec. 5.2.8) to the UI Realization.

In the diagram, the UI Realization can be annotated with the name of one ore more Inner Properties. Alternatively it is possible to draw the UI Realization directly as connection between the Inner Property and the AIO like in figure 6.15. Of course, the latter alternative requires that the Inner Property is visible in the diagram (i.e. the Media Instance must not be depicted in collapsed style) and that only one Inner Property is referenced by the UI Realization.

In the example in figure 6.15 the AIO checkpoint is realized by the Inner Property checkpoint of trackAnim and the AIO obstacle is realized by the Inner Property obstacle of trackAnim.

Figure 6.16 shows the metamodel for the MML Presentation Model introduced so far. The middle part shows the basic metaclasses: a PresentationUnit is owned by a Scene and containts AbstractInteractionObjects. The subclasses of AbstractInteractionObject have been shown in figure 6.13. The upper part defines the UIRepresentation from an AIO to a DomainObject. Analogous to MediaRepresentation (see fig. 6.7) a UIRepresentation can refer to properties and operations. The UIRealization refers to a MediaInstance. A MediaInstance instantiates a MediaComponent and may in addition refer to a specific MediaArtifact. The UIRealization may refer to a number of InnerProperty elements. The Domain Objects and the Media Instances are owned by the Scene and can be interpreted as properties of the Scene (see above).

6.4.3 Sensors

Another specific property of multimedia user interfaces is caused by temporal Media Components: They can invoke actions independently from the user [Vazirgiannis and Boll97]. First of all, temporal Media Components can trigger time-related events, e.g. when they reach a specific point on their timeline or have finished. Second, there are further events depending on the media type caused by dynamic behavior. For instance, a moving animation can trigger an event when it collides with other animations or touches a specific region on the screen. However, a Media Component often does not trigger such events by default. Instead, the developers often have to implement the corresponding functionality (e.g. collision detection) themselves and integrate it with the Media Components. Thus, a modeling language should provide support to specify such event triggers in the models. Moreover, they are required in the Interaction Model (sec. 6.5) to specify the Scene's behavior.

MML uses a metaphor from 3D domain called *Sensor* to model such event triggers. One can distinguish between different types of sensors. The Time Sensor models temporal events in general. In addition, there are other sensors for specific events which can be caused by temporal media. They can be derived from 3D domain as the media type with the most dimensions. Common sensor types are *touch*, *proximity*, *visibility*, and *collision* [Vitzthum08]. The following section discusses how and to which media types they apply in MML.

Types of Sensors A *Touch Sensor* is attached to a visual object and triggers an event when the user touches the object with a pointing device. Visual objects in MML are: Animation3D, Object3D, Animation2D, SubAnimation2D, Video, VideoRegion, Image, ImageRegion, Graphics, and SubGraphics. However, this kind of sensor corresponds to user interaction which is in MML already covered by the

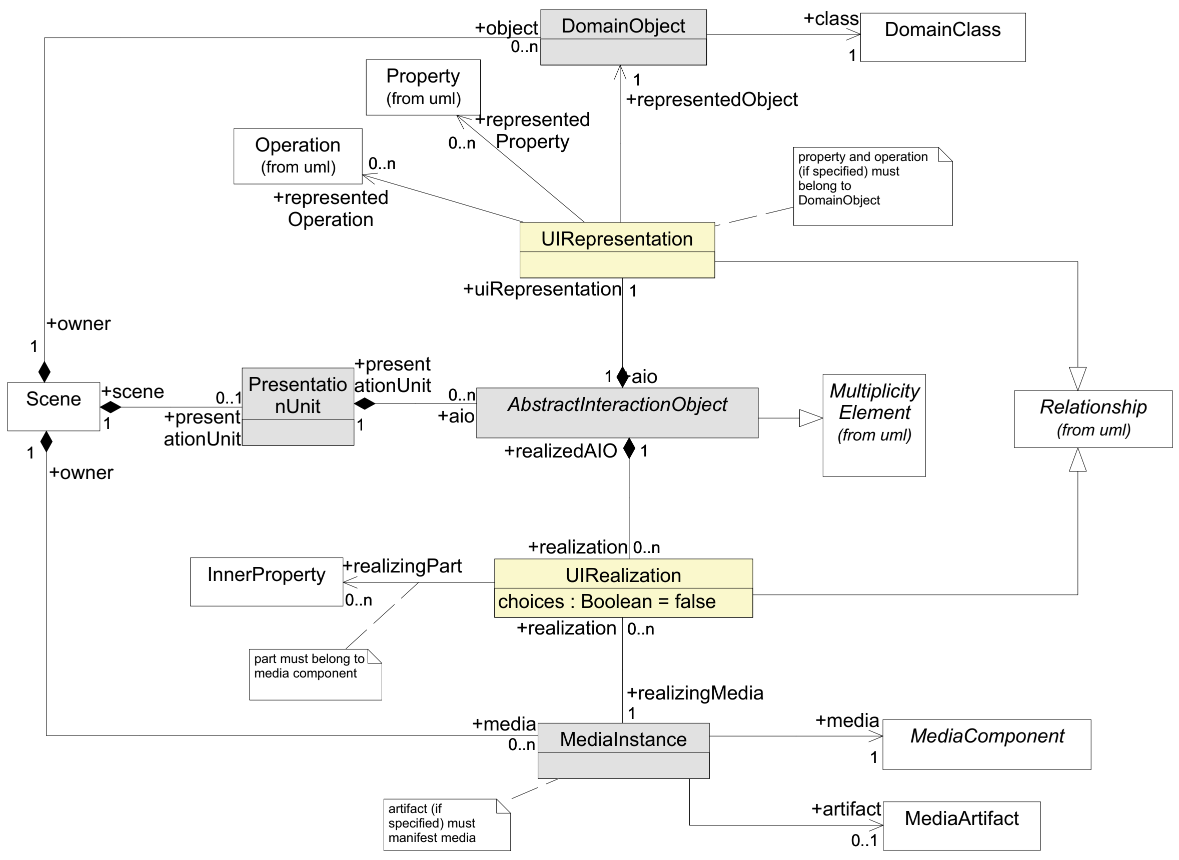


Figure 6.16: Metamodel for MML Presentation Models.

AIOs. The semantics of a touch sensor can be described more precisely in MML by defining a UI Realization between a part of a Media Instance and an Action Component, Edit Component, or Selection Component. Thus, Touch Sensors are not required in MML.

A *Visibility Sensor* can be attached to visual object and triggers an event when the object becomes visible, e.g. after it has been covered by another object or was located outside the screen. In MML this applies to Presentation Units which contain animations. An animation might not only become visible itself (e.g. by moving from outside into the screen) but it can also overlay other visual objects on the user interface which then become visible when the animation moves. Thus, a Visibility Sensor can be attached to any visual object in MML.

A *Proximity Sensor* is attached to a 3D object in a 3D scene and triggers an event when the user navigates within the 3D scene close to this object. More precisely, this means that the difference between the camera position and the object is lower than a specified threshold value (defined during implementation). This sensor can only be owned by 3D animations as there is no analogy for other media types[3].

A *Collision Sensor* can be attached to animated objects and triggers an event when the object collides (i.e. intersects) with another visual object. Possible moving visual objects in MML are Animation3D, Object3D, Animation2D, and SubAnimation2D. The set of objects which are tested for

[3]In 2D domain one can also argue that there is a kind of camera which can be used for zooming. However, zooming in 2D domain is actually just a transformation on a 2D animation and is handled as such in MML. There is no useful semantics for a Proximity Sensor in 2D space which is not already covered by the Visibility Sensor

a collision has to be specified in MML by a relationship Collision Test between the Collision Sensor and the opponent objects.

A *Time Sensor* triggers an event at specific points of time in the application. In MML this points of time can be either frequently after a given time interval (not further specified in MML) or one or more Cue Points of temporal Media Components. In the first case the Time Sensor act as a kind of clock in the application which can be used e.g. to frequently execute an action on the user interface, like starting an animation. This is the default case if no Cue Points are associated with the sensor. The second case means that the sensor triggers an event at specific point of time defined by Cue Points. This can be for example used to execute an action when a temporal media object has reached a specific point in its timeline or has finished.

Currently, there is no specific icon notation for Sensors. They are denoted with a solid-outline rectangle containing the type of sensor (denoted as keyword in guillemets) and the Sensor's name.

Usage of Sensors As discussed above, a sensor can either be assigned to a whole Media Component (like an animation) or only to specific parts of it (i.e. to an Inner Property) like a single geometrical 3D object within a 3D animation. Moreover, a sensor can be owned either by a Media Component or by a Media Instance. In the former case, the sensor is a part of the Media Component and is globally available in all its instances. Then the sensor can only reference to Inner Properties within this Media Component. In the latter case the sensor is only available in the specific instance of the Media Component in the respective Presentation Unit. In this case it might also refer to other Media Instances within this Presentation Unit. This is mainly important for Collision Sensors as they refer to other objects by the Collision Test relationship.

In the diagram, Sensors are assigned to Media Components or Media Instances by a solid line between the Media Component/Media Instance and the Sensor. If the Sensor is assigned to an Inner Property then Inner Property's name can be annotated at the relationship. Alternatively, it is possible to draw the relationship directly as connection between the Inner Property and the Sensor.

The *CollisionTest* relationship is denoted by a dashed line between the Collision Sensor and the element observed and is marked with the keyword CollisionTest.

Figure 6.17 shows as example the Scene Game containing two Collision Sensors obstacleSensor and checkpointSensor. They belong to the Media Instance carAnim and test for collisions with obstacle and checkpoint.

Figure 6.18 shows the metamodel for sensors. A Sensor is always owned either by a MediaComponent or a MediaInstance. (This relationship is not defined as a metaclass of its own as it has no further properties.) A Sensor may in addition refer to an InnerProperty which must belong to the Sensor's owner, i.e. the MediaComponent or the MediaInstance. A CollisionSensor owns one or more relationships CollisionTest which refer either to a MediaInstance or an InnerProperty.

Tool Support for Presentation Diagrams The Presentation Diagram contains three different aspects: The Presentation Units with UI Realizations, the UI Realizations, and the Sensors. Thus, a modeling tool should support the modeler to view only one or two of these aspects and hide all other model elements by selecting between the different views. The modeler should be able to hide the UI Realizations and domain objects, the UI Realizations and Media Components, and the Sensors with all their relationships. Hiding can mean either to set them completely invisible or to set them into background by displaying them in light gray color or with an increased alpha value.

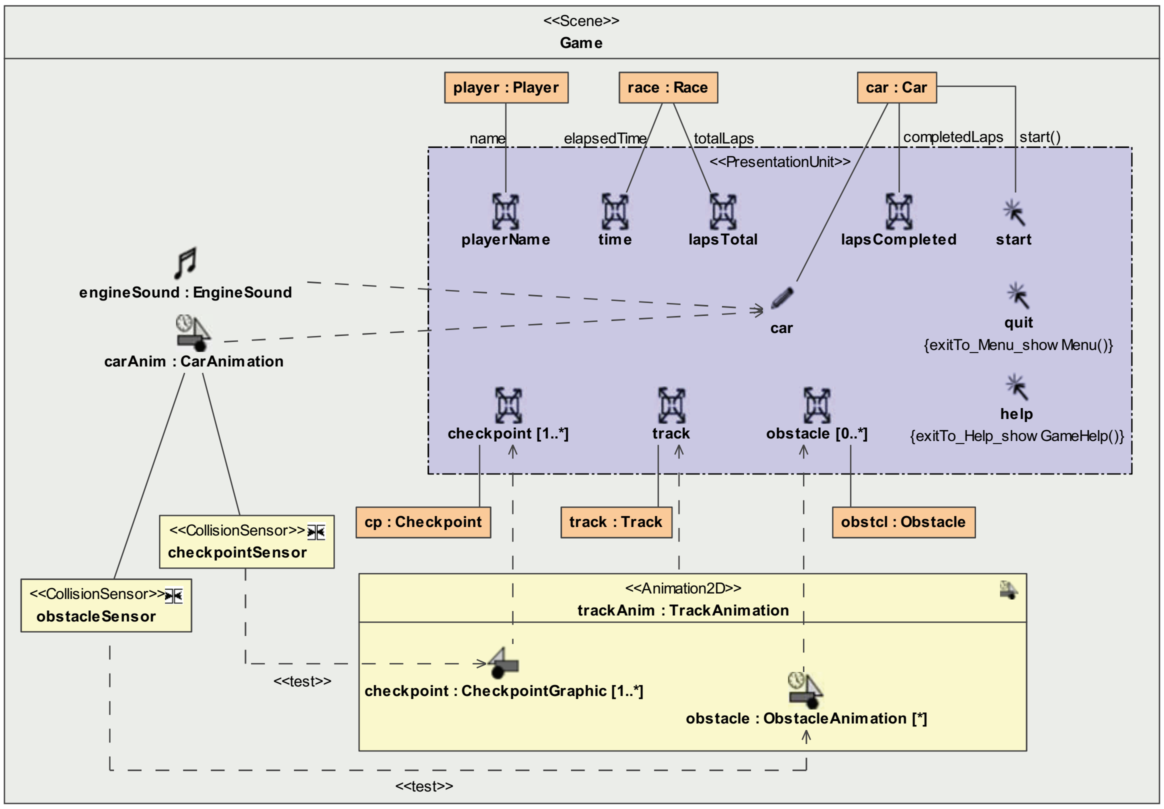


Figure 6.17: MML Presentation Diagram for the Scene Game enhanced with Sensors.

6.5 MML – Interaction Model

The Interaction Model models the user interaction and the resulting behavior of the Scene. Each Scene has an Interaction Model of its own. The core idea is to specify how events from the user interface trigger operations from the application logic or from user interface elements. In multimedia applications this includes also predefined temporal behavior between media instances within a Scene. In that way the Interaction Model specifies the interplay between the elements defined in the foregoing models.

This section starts with a general discussion on the basic concepts, the required level of abstraction, and existing modeling concepts which can be reused for its realization (subsection 6.5.1). The second part of this section (sec. 6.5.2) presents as proof of concept MML Interaction Diagrams based on UML Activity Diagrams. Finally, section 6.5.3 provides a short discussion on temporal synchronization.

6.5.1 Rationale

Main Concept The different MML models defined so far consist of various elements which can influence the application's behavior within a Scene. Some of them, like user events, trigger behavior while others, like domain class operations, encapsulate some behavior to be executed. The coordination of triggers and triggered behavior is managed by the Scenes. In detail, the following triggers are available within a Scene:

- Events from contained AIOs (possibly realized by Media Components) defined in the Scene's

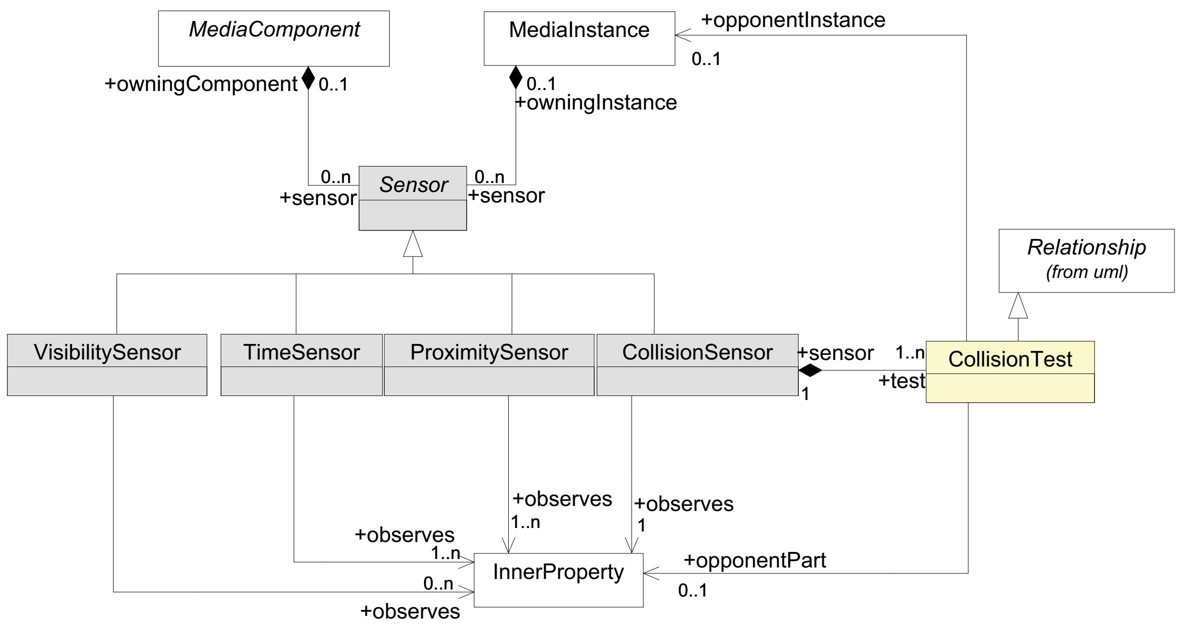


Figure 6.18: MML Metamodel for Sensors.

Presentation Model (sec. 6.4).

- Events from Sensors defined in the Scene's Presentation Model (sec. 6.4).
- In addition, Entry-Operations (sec. 6.3) are executed by definition when a Scene becomes active and can thus trigger other behavior as well.

The following elements represent encapsulated behavior which can be triggered within a Scene:

- Operations from domain classes defined in the Structural Model (sec.6.2). Therefore, the Scene refers to domain objects as defined in the Scene's Presentation Model (sec. 6.4). If additional domain objects are required to be accessed by the Scene then they can be passed as parameters of the Scene's Entry-Operations.
- Operations of Media Instances contained in the Scene (e.g. to play a video) defined in the Interfaces of Media Components (sec. 5.2.3). The Media Instances contained in the Scene are specified in the Scene's Presentation Model (sec. 6.4).
- In addition, in a dynamic interface it must be possible to modify the AIOs and Sensors on the user interface as well (e.g. to disable, highlight or hide an AIO or to disable a Sensor). Analogously to Media Components, MML defines standard interfaces for the different types of AIOs and Sensors. The interface contain operations which encapsulate required behavior of Sensors and AIOs. Table 6.1 shows the standard interfaces and some exemplary operations currently defined in MML. The code generator can then map the standard operations defined within these interfaces into code on the specific target platform (e.g. setting the value of the property enabled of a Button in ActionScript).

The Interaction Model specifies the interplay between those elements.

Level of Abstraction According to the general goals of MML (p. 34) it is intended to specify the Interaction Model on a level of abstraction which enables to generate code from the models.

[4]The datatype OCLAny is used to refer to any kind of model element.

Metaclass	**Standard Operations**

AIO	setValue()
	getValue():OCLAny[4]
	setVisible()
	setInvisible()
	enable()
	disable()
	highlight()
	deHighlight
	update()

Sensor	hasSensed() getTrigger():OCLAny
CollisionSensor (in addition to Sensor operations)	getOpponent():OCLAny

Table 6.1: AIO and Sensor Standard Interfaces

As stated in section 6.4, a Scene is mapped to a class and its contained elements are mapped to class properties. As the lists above show, all elements (AIOs, domain objects, Media Instances, and Sensors) are already defined in the foregoing models and are thus available as properties in the Scene. The Interaction Model thus has to specify only the behavioral relationships between them. The idea is that the behavior within a Scene does usually not require much complex algorithms – those are usually encapsulated in the domain class operations. Thus, it is mostly possible to model a Scene's behavior with acceptable effort. Consequently, basically the complete code for a Scene (i.e. including the the operation bodies) can be generated from the models. In turn, it is necessary to provide modeling concepts on a sufficiently low level of abstraction so that the modeler is able to specify all details of the Scene's behavior if desired.

Reuse of Modeling Concepts Given the basic concept for Interaction Models above, this paragraph now briefly discusses possible existing modeling concepts and notations which can be used for MML Interaction Diagrams.

The basic behavior of Scenes is already defined by the temporal relationships defined in the Task Model (sec. 6.1). For less dynamic user interfaces the information in the Task Model is sufficient to directly generate code – as realized in many existing approaches in user interface modeling area (see sec. 4.1). For example, the Menu Scene in the racing game example requires only AIOs to input the player's name and select a car and a track. The Task Model specifies that these tasks can be performed in any order. The Presentation Model specifies how the AIOs are related to domain objects. If the relationships between Tasks and corresponding AIOs is available in the model (defined e.g. by isExecutedIn relationships like in UsiXML, see sec. 4.1.2) it is possible to generate the complete code for the Scene analogous to existing user interface modeling approaches. In this case, an additional Interaction Model is not mandatory.

However, for complex, dynamic Scenes, like the Scene Game in the racing game example, it might be desired to specify the Interaction Model on a lower level of abstraction. For example, it can be an important decision in the Scene Game whether user input is handled asynchronous, i.e. by executing an event listener for each occurring event, or synchronized by polling for input in a specified order. The latter mechanism is commonly used for highly dynamic and time-dependent user interfaces

[Besley et al.03, Pol08, Lord07]. In general, Task Models focus on the viewpoint of user tasks – not on the system to be implemented [Paternò01]. Thus, MML complements the Task Models with another diagram type supporting a more system-related point of view and modeling detailed behavior. For this purpose MML reuses UML Activity Diagrams.

UML Activity Diagrams are part of the UML standard and provide the possibility to model a system's behavior very close to implementation if desired. Moreover, extended Activity Diagrams are already used in other UML-oriented user interface modeling approaches like UMLi [da Silva and Paton00] and CUP [Van den Bergh06] (see sec. 4.1.3) as UML-based variant of Task Models. Moreover, the semantics of UML 2 Activity Diagrams has already been formally defined [Störrle04]. A general introduction and discussion of UML2 Activities can be found in the article series by Bock [Bock03a, Bock03b, Bock03c, Bock04a, Bock04b].

As discussed in [Van den Bergh06] and [Nóbrega et al.05] Task Models can be mapped to (extended) Activity Diagrams. Thereby, each task is represented by an UML Action while the temporal relationships are mapped to corresponding control flow. This can be used for MML as well to automatically derive an Activity Diagram (i.e. MML Interaction Model) from the Task Models – either as starting point to add more implementation-specific details or for direct transformation into code.

6.5.2 MML Interaction Model based on UML Activity Diagrams

This subsection shows how the concepts of MML Interaction Models can be realized based on UML Activity Diagram. It briefly presents the most important restrictions and adaptations compared to plain UML Activity Diagrams and shows some concrete examples from the racing game application.

Activities An Activity specifies the behavior of an operation of the Scene. These operations are mainly the Scene's Entry-Operations. But it is also possible to define additional (helper) operations for a Scene to encapsulate some behavior (e.g. behavior which should be reused in multiple Entry-Operations). An Activity is always associated with a Scene (the operation owner) and can access all properties of the Scene.

The Activities are denoted like in UML. Their names result from a concatenation of the name of the Scene and (separated by '_') the name of the the represented operation. Figure 6.19 shows as example the Activity for the operation start of the Scene Game.

Objects All properties of a Scene defined in the Presentation Model can be represented as Object Nodes in the Activity. This includes domain objects, AIOs and sensors. In addition, the operation's parameters are represented by Activity Parameters, like in UML. All Object Nodes, including Activity Parameters, can be used as targets of Actions and as parameter values for Actions.

If an Object Node's type has a specific icon notation then the icon is used to denote the Object Node itself as well. For example, an Object Node from type MML Input Component is denoted by the icon for MML Input Components. The Object Node's name is denoted below the icon. Optionally it is possible to place the type of the node behind its name, separated by ':'. If the Object Node's type has no specific icon notation (e.g. domain classes) it is denoted like as a rectangle like conventional Object Nodes in UML. Activity Parameters are denoted like in UML as well.

In the example in figure 6.19 the Object Node representing the Action Component start is denoted like an MML Action Component. There are two Activity Parameters :Player and :Track which correspond to the operation parameters defined in fig. 6.9. In addition, the is an Object Node car from type Car.

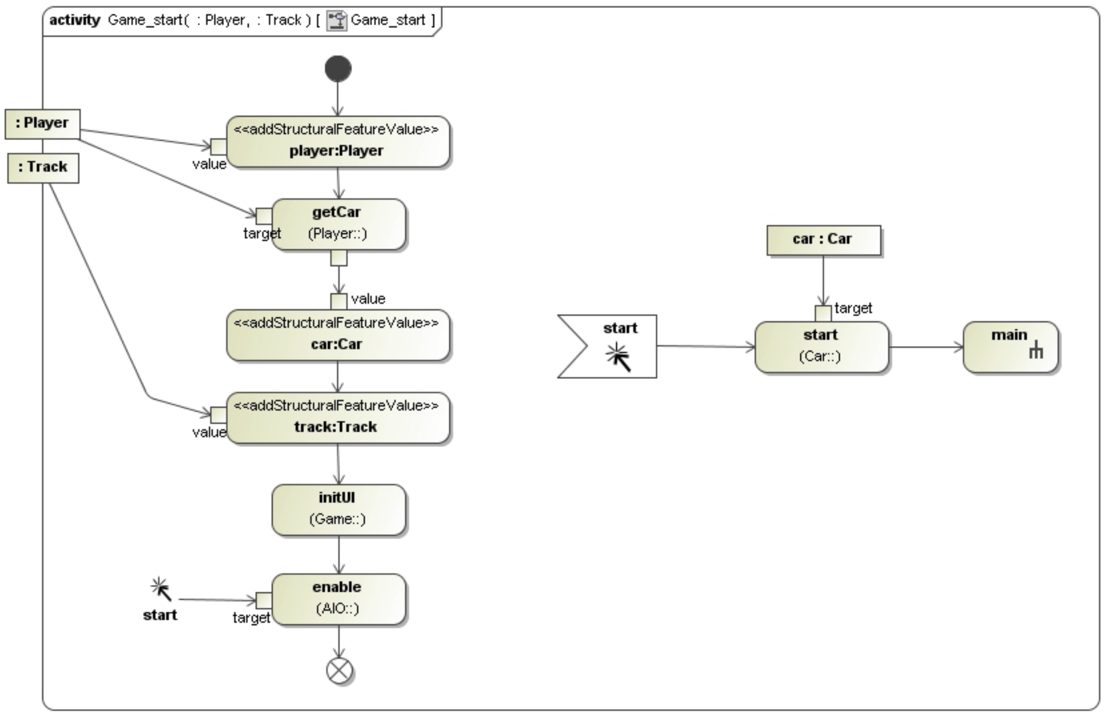


Figure 6.19: MML Interaction Diagram for Entry Operation start of Scene Game.

Actions The primary kind of Action to be used in MML Interaction Diagrams is the CallOperationAction from UML. It represents an operation call on a target object. The target object can be specified by an Object Flow to an Input Pin target of the Action. If no target object is specified then the target object is by definition the Scene itself.[5] If the called operation has parameters then they must be passed to the CallOperationAction via Input Pins marked as with the parameter name and optionally with the parameter type (separated by ':'). As discussed in sec. 6.5.1 the available operations can belong to a domain object, a Media Instance, an AIO, or a sensor.

Besides CallOperationActions it is possible to use other kinds of UML Actions like *Object Actions* to e.g. create new objects, *Structural Feature Actions* to e.g. read a property value, or *Variable Actions* to e.g. store a value in a local variable, etc. (see chapter 11 in the UML specification [Obj07c]).

The Actions and Pins are denoted like in UML: CallOperationActions show the name of the operation they represent. The name of the class may optionally appear below the name of the operation, in parentheses postfixed by a double colon. Other kinds of Actions are marked with a keyword indicating their type. If an CallOperationAction refers to another operation (of the Scene) which is defined by an Activity itself then this is marked (like in UML) by a rake symbol within the Action.

The example in figure 6.19 shows the different possibilities: The first four Actions are just used to initialize the Scene's class properties player, car, and track. The values for player and track are passed as the Entry-Operations parameters. The value for car is fetched from the player object by calling its operation getCar().

The next step in the example is a CallOperationAction initUI which encapsulates the initialization of the user interface and is not further specified in the example. No target object is specified for initUI which indicates that the operation is owned by the Scene itself.

[5] in UML the owner of an Activity (here: the Scene) would be retrieved by ReadSelfAction.

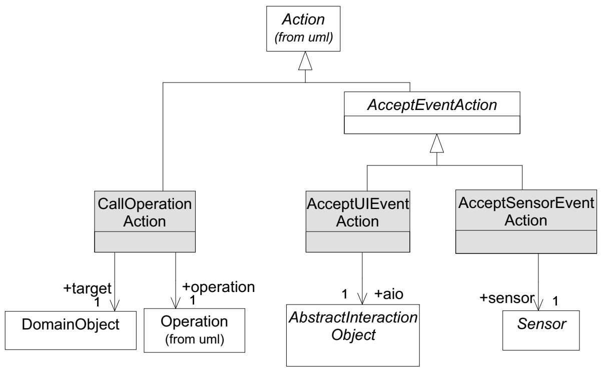


Figure 6.20: Metamodel for MML-specific actions.

In the next step, the operation enable of the AIO start is called. This operation has been defined in the AIOs standard interface. There are two more CallOperationActions: start is an operation of the domain object car and main is an operation of the Scene itself which encapsulates the Scene's main behavior. The rake symbol indicates that main is defined in another Activity (see figure 6.21).

Events The events from AIOs and Sensors can be modeled in terms of UML AcceptEventActions. MML specifies specialized subclasses for the different types of events triggered by AIOs and Sensors: An *AcceptUIEventAction* is associated with an AIO and indicates that the AIO has triggered an event. In case of an Input Component or Edit Component it means that the user has input some data. In case of an Action Component it means that the user has triggered the Action Component. An *AcceptSensorEventAction* specifies that a sensor reports the occurrence of the observed event.

All types of AcceptEventAction in MML are denoted like in UML as a concave pentagon. They are anonymous and contain the name of the associated AIO or sensor instead of a name of its own. In addition, the type of AIO or Sensor is indicated by the corresponding icon or keyword defined in section 6.4.

Figure 6.19 shows as example an AcceptUIEventAction triggered by the Action Component start. It calls the operation start() of the domain object car. This adheres to the Presentation Model in figure 6.14 which already specifies that the Action Component start represents the domain object car. However, the Interaction Model can refine the specification from the Presentation Model. For instance, figure 6.14 shows that the operation main() is called after the operation start().

Like in UML, the AcceptEventActions can be used in combination with InterruptibleActivityRegions. When an InterruptibleActivityRegion is left via an edge defined as interruptible edge then all remaining tokens inside the InterruptibleActivityRegion are terminated. This mechanism can be used to model that an event interrupts the control flow. For example, figure 6.21 shows (on the bottom left side) an AcceptUIEventAction associated with the Action Component help. If help triggers an event, the Scene's main control flow is terminated and the Exit Operation exitTo_Help_showGameHelp is executed.

Figure 6.20 shows the metamodel for the MML-specific actions.

Further Elements Besides the elements described above, the Interaction Model supports the model elements from conventional UML Activity Diagrams to model behavior, like *Control Flow*, *Control Node*, *Object Flow*, etc.

Figure 6.21 shows a more complex example specifying the operation main from the Scene Game. It shows the core control flow for the racing game. In this model the input from AIOs and sensors is polled by synchronized operation calls instead of using event listeners which is a common mechanism in highly interactive and dynamic Scenes[6]. Conditional activity nodes are used to specify decisions. The final actions in the example calls an Exit Operation of the Scene.

For such complex Scenes the Interaction Model provides additional support for the developers as it clearly reflects the intended main program flow. This provides for instance the software developers a good orientation for the final implementation of the domain class operations.

Figure 6.22 shows a simplified metamodel extract from UML for the basic Activity Diagram elements. The abstract metaclass Action is refined by the MML specific actions previously shown in figure 6.20. In addition, the UML subclasses of Action mentioned above (like AddStructuralFeatureAction, etc.) are reused from the UML specification (see [Obj07d], chapter 11). In MML each Activity is owned by an operation of a Scene (figure 6.23).

6.5.3 Temporal Synchronization

Various existing approaches in multimedia area propose concepts to model the temporal synchronization between media objects. [Bertino and Ferrari98] provides a good overview on them. For application design only the *inter-object synchronization* is relevant, i.e. the synchronization between different media objects (in contrast to *intra-object synchronization* which refers to internal synchronization and is provided by the implementation platform).

[Bertino and Ferrari98] distinguishes between two concepts to model synchronization: Timline models and constraint-based models. Timeline models show the media objects on a temporal axis which is very intuitive but less flexible. In particular, it provides low support for interactivity and control constructs like decisions. For example, OMMMA (sec. 4.3.2) uses extended UML Sequence Diagrams. Although since UML 2.0 Sequence Diagrams support constructs like loops and decisions as well they still become very complex when modeling non-trivial interactive applications. OMMMA solves this by using the Sequence Diagrams for modeling predefined behavior only and by putting all interactive behavior into the State Charts. However, the result is that in highly interactive applications the State Charts become very complex while the Sequence Diagrams become trivial.

A more flexible approach is the usage of constraints for the synchronization. Examples are interval-based relationships like defined by [Allen83] or [Wahl and Rothermel94]. As described in section 6.4.1 such relationships are used in user interface modeling approaches for modeling the spatio-temporal layout and can thus already be covered by the Abstract User Interface Models.

The MML Interaction Model is already on an abstraction level more close to implementation and focuses on modeling interactivity. Consequently, temporal synchronization is specified here in terms of Cue Points and Events. More abstract relationships would have to be modeled as part of the spatio-temporal layout in the Presentation Diagram.

Figure 6.24 shows as example an MML Interaction Diagram for the Entry Operation show of the Scene Intro. Here a sound and video should be played in parallel. During the last sequence of the video an animation should be played which fades in a headline showing the game title. This can be modeled in MML by a Time Sensor LastVideoSequence which triggers an event when the last video sequence is reached. The corresponding AcceptSensorEventAction then causes the animation to play. Another Time Sensor VideoFinished is used to exit the Scene when the video has finished.

[6]For purpose of clarity, Car is represented here by an Output Component and two additional Action Components accelerate and leftRight instead of just an Edit Component Car like in the Presentation Model

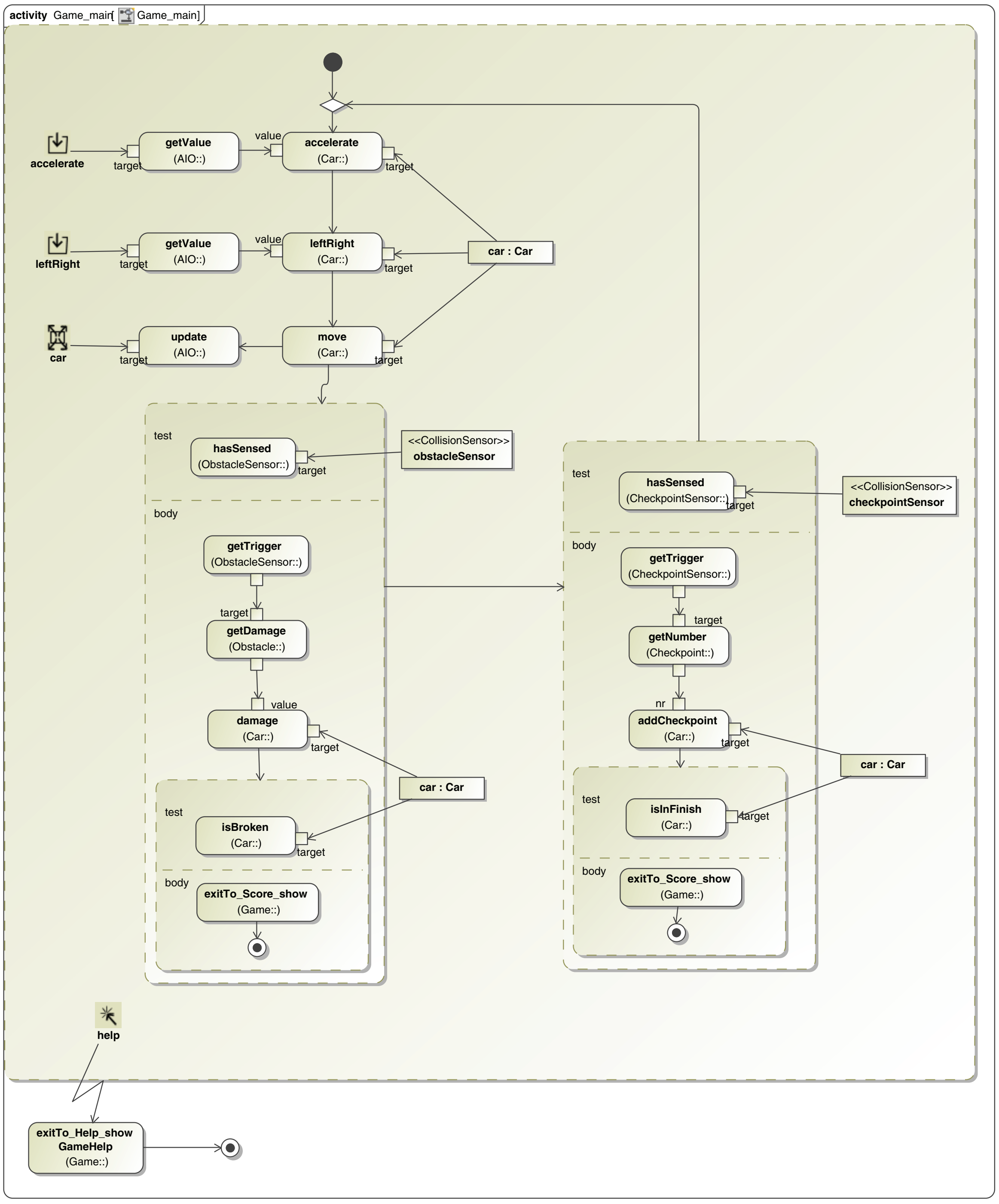


Figure 6.21: MML Interaction Diagram for operation main of Scene Game.

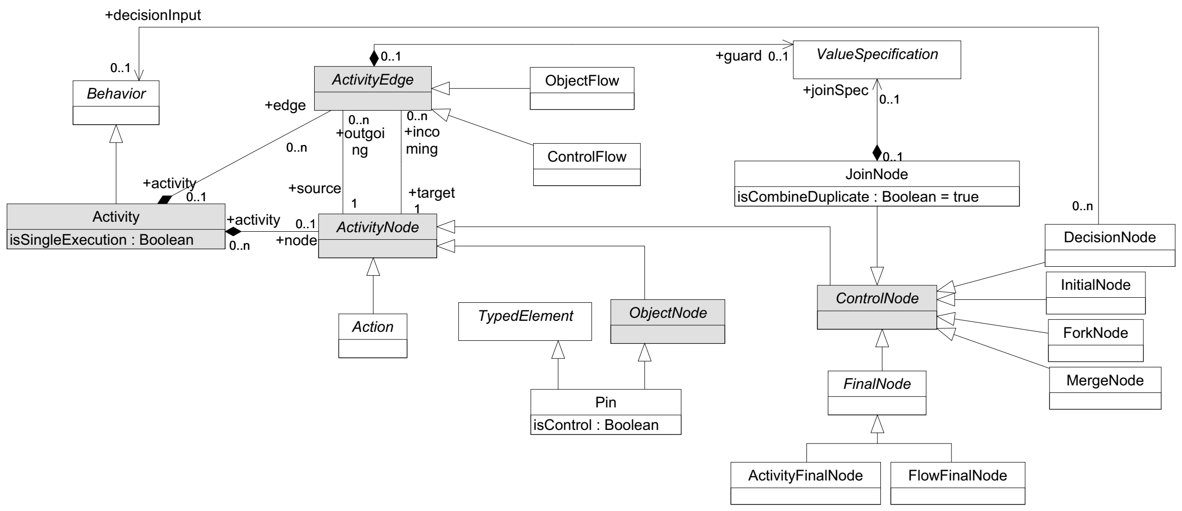


Figure 6.22: Simplified extract from the UML metamodel for Activities.

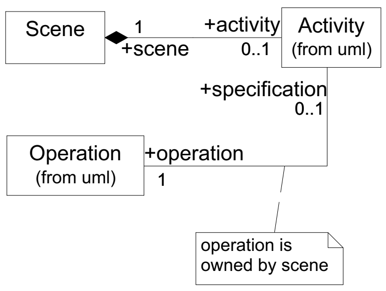


Figure 6.23: Metamodel defining the integration of Activities and Scenes.

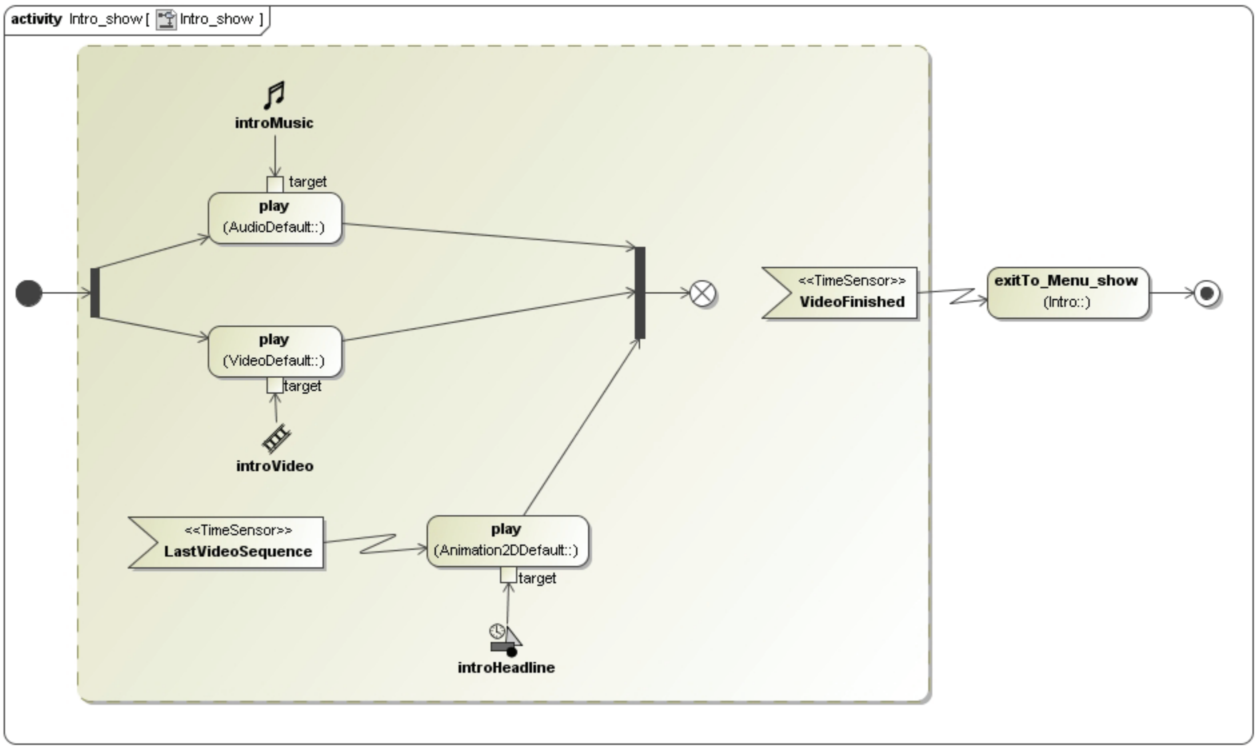


Figure 6.24: MML Interaction Diagram for operation show of Scene Intro.

6.6 Model Interrelations and Modeling Process

This section shows the modeling process and gives an overview on the relationships between the different kinds of MML models. Here the focus lies on the modeling language itself, independently from the details of code generation which are discussed in the next chapter 7.

6.6.1 Modeling Process

Figure 6.25 shows a typical modeling process for MML models. This section will go in detail through the diagram.

Temporal Dimension The vertical axis in figure 6.25 shows the temporal dimension in the process. Modeling languages like MML support the design phase in the development process, i.e. MML builds a bridge between the analysis phase and implementation phase. Thus, MML models should base on the results of analysis. Those are e.g. textual specifications like a detailed listing of expected functionalities, visual examples like sketches, mock-ups, and storyboards , but also analysis models like domain models or flowcharts (see [Bailey and Konstan03, Osswald03, Mallon95]). It is assumed here that the analysis phase is not different from that in projects without MML. A small example for a possible concrete overall development process is also shown later in figure 8.3.

The requirement analysis builds the starting point for the MML modeling process shown in the diagram. The Task Models are located on topmost as they can be classified as on the borderline between requirements analysis and design phase. While the other models describe the system to be

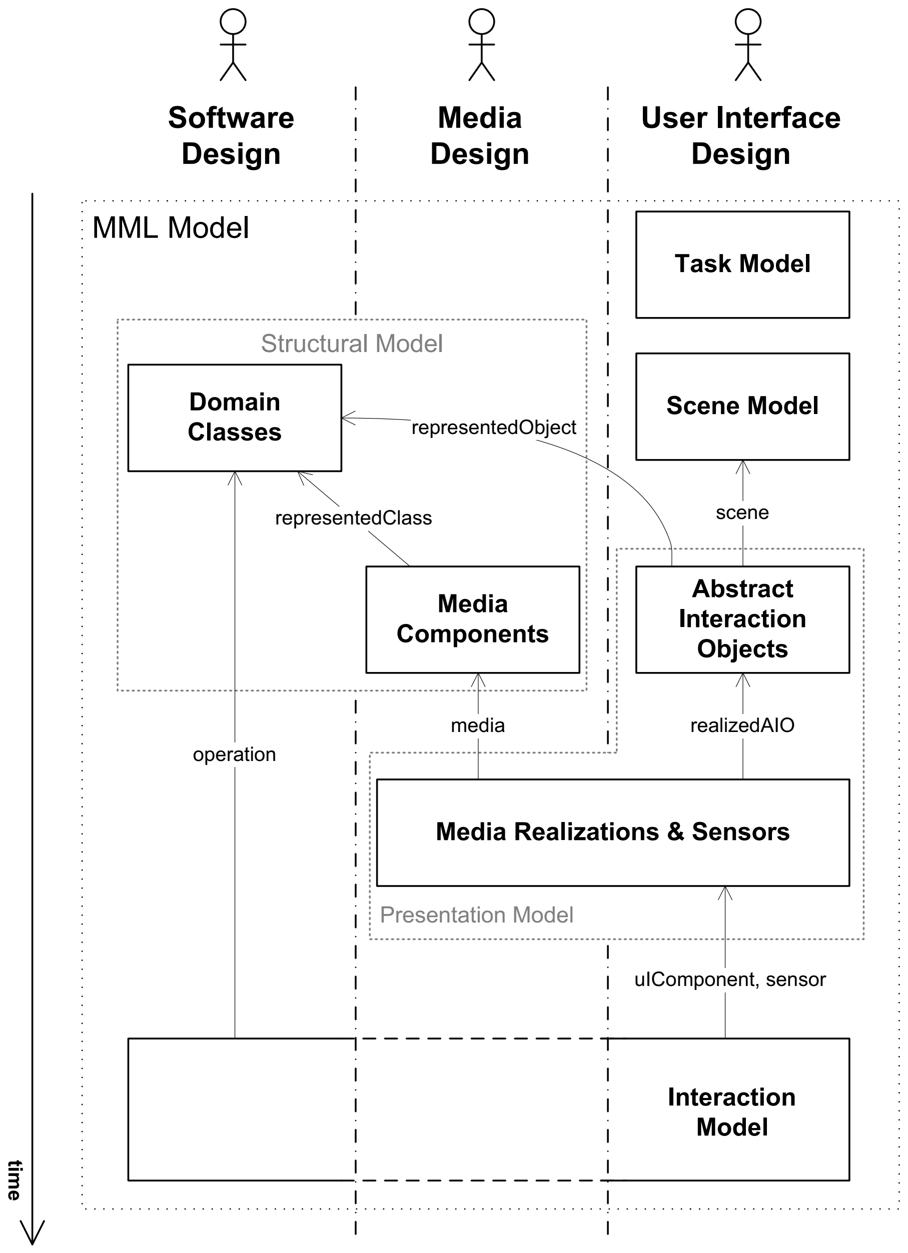


Figure 6.25: Typical modeling process for MML models.

implemented, the Task Model is still on a higher level of abstraction. It can be used in MML to derive other MML models from it but (if MML Interaction models are fully specified) it is not mandatory for the code generation.

Role Dimension One of the main goals of MML is the integration of multimedia design, software design, and user interface design (p. 34). As discussed in section 3.1.1 the models specifies an interface for the respective application part and form a kind of contract between the developers. The horizontal axis in figure 6.25 shows how the MML models can be assigned to the corresponding developer roles to indicate which expert knowledge is considered for a specific kind of MML model. Of course, in practice one role can be assumed by multiple developers or, in small projects, one developer can assume multiple roles.

Some models require knowledge of multiple developer roles. This means that the developer groups have to define the relationships between different expert areas in cooperation. In addition to the three developer roles above, it can be useful to establish the role of a "*modeling expert*" in the modeling process. This can be a software designer with good knowledge in MML. The modeling expert supports the other developers in modeling and takes care that the models are correct and consistent.

Model Interrelations The center of figure 6.25 shows the MML models introduced in the foregoing sections. Some of them are split up into several parts to arrange them according to the two axes. The arrows show how the models build up on each other. If a model builds up on another one, e.g. by referencing elements, this often means in turn that the existing model is refined during this step. For example, the modeler might detect that some elements in the existing model are still missing or that the structure must be refined.

Task and Structural Model The modeling starts with the Task Model (sec. 6.1) and the Structural Model (sec. 6.2). They can be specified independently from each other. The Structural Model can be divided into domain classes and the Media Components. The domain classes are specified by the software designer. They can be derived from the requirement specification in the same way as in any conventional object-oriented software development process.

The media designer specifies the Media Components identified during requirement analysis. The Media Representation relationships between domain classes and Media Components have to be specified in cooperation between software designer and media designer. They also specify together the inner structure for Media Components. This is necessary for Media Components where inner parts should be modified or accessed by domain classes or should trigger events. During this steps the software designer adds domain classes to the Structural Model or refines them, if necessary.

Scene and Presentation Model The Scene Model (sec. 6.3) is specified by the user interface designer. The Scenes and the transitions between them can be derived from the Task Model as explained in section 6.3. Each Scene is associated with a Presentation Unit specified in an Presentation Model.

The Presentation Model (sec. 6.4) can be split into several parts. The Abstract Interaction Objects are, similar like the Scene Model, specified by the user interface designer and can be derived from the Task Model as well. They are associated with domain objects which refer to domain classes from the Structural Model. If some domain classes are missing or must be refined then the user interface designer has to coordinate this with the software designer at this point.

In the next step the Presentation Model is complemented with Media Components and Sensors. At this point the user interface designer and the media designer have to cooperate. They specify together Media Representation relationships between the Abstract Interaction Objects and Media Instances. The Media Instances refer to Media Components from the Structural Model. If Media Components are missing they must be added to the Structural Model. In case that added Media Components are related with application logic then it has to be coordinated with the software designer. If inner parts of a Media Component, which has not been specified explicitly yet, should trigger user interface events then these inner parts must be specified in the Structural Model. In addition, the user interface designer and the media designer can add Sensors to Media Components.

Interaction Model Finally, the Interaction Model (sec. 6.5) is specified in cooperation between the software designer and the user interface designer. It specifies the how the user interface events and Sensor events trigger operation calls on domain objects. Basically, the user interface designer is responsible for the interaction. The interaction can be derived from the Task Model as well – at least to some degree, depending on the Scene's complexity. The software designer's knowledge is mainly required for specifying the behavior of complex, dynamic Scenes like the Scene Game in figure 6.21.

The Interaction Model refers to Abstract Interaction Objects, Sensors and domain objects from the Presentation Diagram. If Abstract Interaction Objects or Sensors are missing they have to be added to the diagram. Domain objects usually need not to be added to the Presentation Model at this point as often they can be accessed via properties or operations of available domain objects or as parameter of the Scene's Entry Operation. As the Interaction Model refers to operations of domain objects it thus refers also to the Structural Model where they are defined. If class operations are missing or should be refined then the Structural Model must be changed accordingly.

Alternatives It is not mandatory to follow the modeling process always as described here. Basically it is possible to start with any kind of MML model. An iterative process is possible as well. Of course, model elements from other models must be defined at least at the moment when they should be referenced. For example, it is possible to start the process with the Scene Models and the Presentation Models. Specifying the UI Representations then requires domain objects which in turn requires domain classes. Of course it is possible that the user interface designer initially creates them. The software designer then still can create the Structural Model on that base. Alternatively, in a more iterative process it is also possible that the UI Representations are added later to the Presentation Diagram.

Indeed, it is also possible that all three developer groups iteratively specify all models in cooperation together.

6.6.2 Conceptual Interrelations

Another more abstract view on the integration of software design, media design, and user interface design in MML is given in figure 6.26. It shows the very abstract "essence" of the concepts which turned out during the development of MML: Media Components represent domain classes. This is modeled by Media Representation relationships. User interface elements, i.e. Abstract Interaction Objects in MML, represent domain classes as well which is modeled by UI Representation relationships. The Media Components can realize Abstract Interaction Objects which is modeled by UI Realization relationships. Finally, the Interaction Diagram refers to all three design aspects and defines the behavioral connections between them.

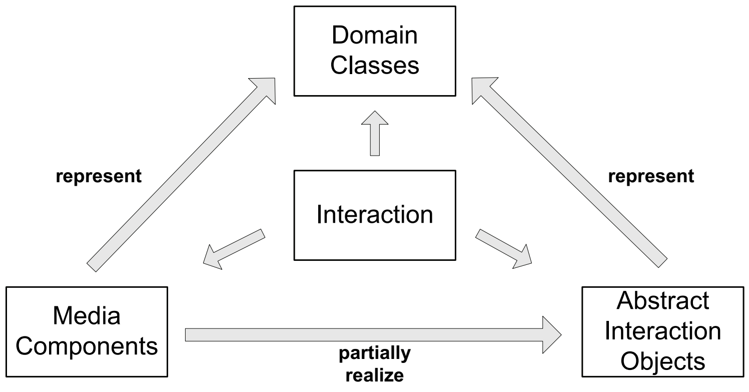


Figure 6.26: “Essence” of MML.

6.7 Modeling Tool Support

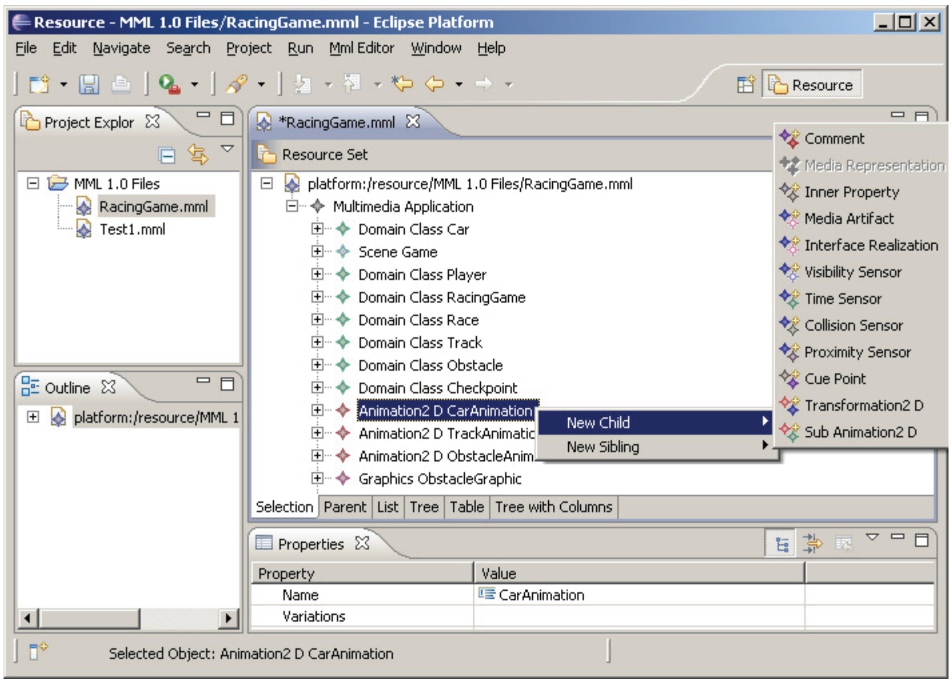
Some prototypical tool support has been created for MML to apply and evaluate the language and to create code from the models. Even if the tool support is mainly for evaluation purposes it is still necessary to aim for a sufficient degree of usability as otherwise users are not willing to use it. Also, often users unintentionally judge the feasibility and quality of a modeling approach by the quality of its tool support. On the other hand, creating a fully featured visual modeling tool requires very much effort. Thus, here the goal was to find a pragmatic solution with a good ratio between effort and result.

Section 3.4.3 lists three common ways for implementing modeling tools: Extending an existing modeling tool, using a Meta-CASE tool, or using frameworks and APIs from MDE area. This section briefly explains which technologies have been selected for which reasons and presents the resulting tools: A metamodel implementation based on EMF, a visual modeling tool based on the UML tool *Magic Draw*, and a transformation for integrating these two tools. These tools have been used for the application of MML described in chapter 8.2.

EMF-based Metamodel Implementation One of the advantages of Model-Driven Engineering is the fact that due to common concepts and standards it is possible to create frameworks which can be applied to any MDE-compliant modeling language. An important example are the Eclipse-based frameworks described in section 3.4.3. A core framework among them is the *Eclipse Modeling Framework* (*EMF*) which has been used to generate a basic metamodel implementation from MML: The MML metamodel is specified using the UML tool *Rational Rose* (2002 Enterprise Edition). EMF supports files in the Rational Rose format and enables to automatically generate from them (among other things):

- An implementation of the metamodel in Java and
- a simple tree editor for creating and editing models compliant to the metamodel.

Figure 6.27 shows a screenshot of the tree editor generated from the MML metamodel. The with using the racing game application from this work as example MML model. The editor automatically ensures that created models comply to the abstract syntax specified in the metamodel, like the containment hierarchy, attributes types, relationships, multiplicities, etc. For example in figure 6.27 a new model element is added as child to the 2D animation CarAnimation. The context menu allows only to select metaclasses which are valid children of 2D animation.



Property	Value
Name	CarAnimation
Variations	

Selected Object: Animation2 D CarAnimation

Figure 6.27: Tree editor for MML generated with EMF.

The EMF-based editor also automatically provides basic editor features like loading and saving models as files. The file format is XMI, the XML-based exchange format for models defined by the OMG (see sec. 3.4.2).

EMF-based models and metamodels have a large advantage: They are supported as input in various other Eclipse-based MDE tools. For example, the MDT OCL plugin for Eclipse has been applied in [Shaykhit07] to automatically validate whether an MML model complies to the OCL constraints from the MML metamodel. Another example used in this thesis (chapters 7 and 8.1) is the Eclipse plugin for ATL model transformations.

Visual Modeling Editor The next logical step is to build a visual editor on top of the EMF-based MML implementation. The common way for this was at that time the Eclipse *Graphical Editing Framework* (*GEF*) which supports creating visual editors. However, GEF does neither provide any specific support for modeling tool functionality nor support for integration with EMF. Today the Eclipse *Graphical Modeling Framework* (*GMF*) is available for this purpose but it was just under development at this time in 2004. An available framework which provides semi-automatic support for creating GEF modeling editors was *openArchitectureWare* (*oAW* [oAW, Völter and Kolb05]). Two project theses [Eicher05, Mbakop06] supervised by the author of this thesis conducted very first prototypes using GEF and oAW. However, it turned out that implementing a usable visual modeling tool for all types of MML diagrams would require too much time and effort.

An alternative solution would be using Meta-CASE tools like MetaEdit or GME2000. However, these tools provide their own proprietary mechanisms and were not compliant to Eclipse-based tools from the research community while they still need considerably learning effort.

To gain results quickly, it was decided to extend an existing UML tool, *Magic Draw*, instead. Magic Draw has established in the last years as one of the most successful professional UML tools and provides good and easy to use support for UML extensions (UML Profiles). In addition, it provides

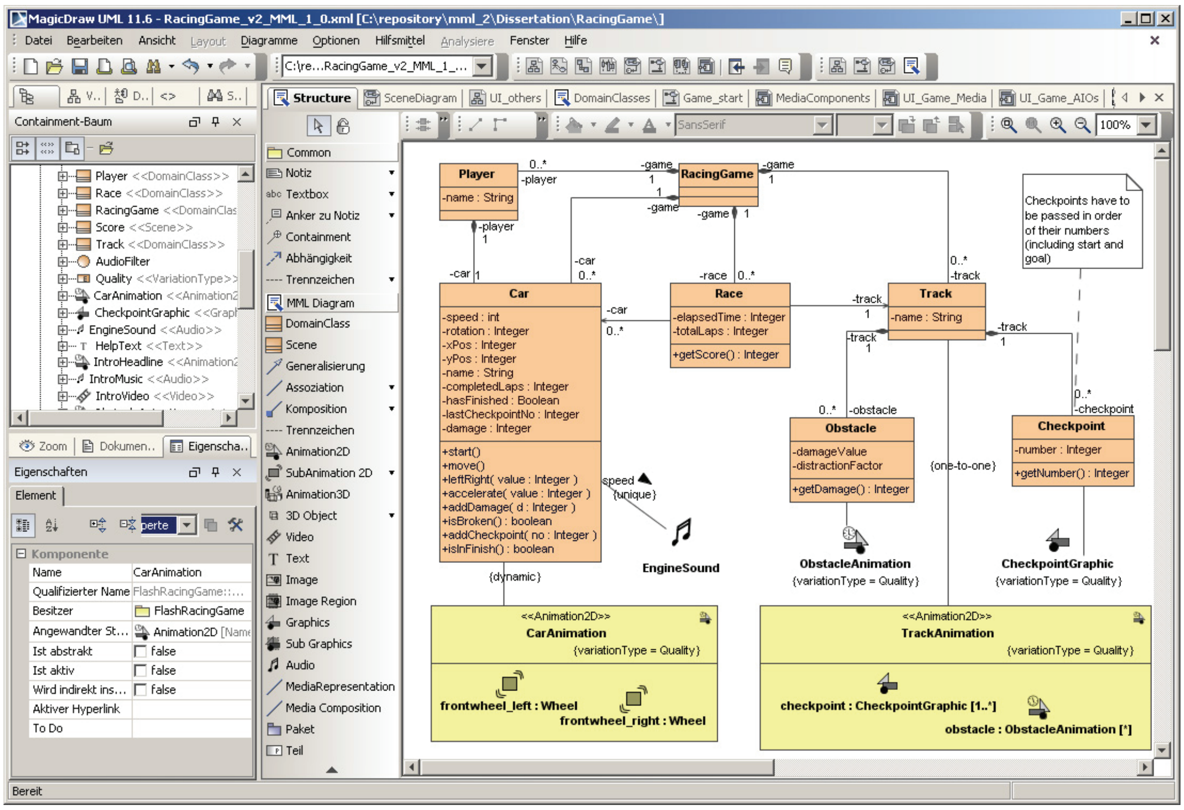


Figure 6.28: MML Structure Model in Magic Draw

academic classroom licenses for free which includes that students may install a version at home.

The prerequisite for using Magic Draw is that MML has to be defined as UML Profile (see sec. 3.6). The implementation as Magic Draw Profile could be finished very quickly in a project thesis [Shaykhit06] supervised by the author of this thesis. Indeed, defining MML as a UML Profile is just a workaround and not the original intension of MML. Nevertheless, the result works very well.

The main drawback of this solution is that is that Magic Draw did not support custom constraints at that time. This means that it does not prevent the modeler to create models which do not comply to the MML metamodel. This could be improved to some extent as Magic Draw allowed to create custom diagram types with customized tool bars. So it is possible to provide the modeler for each MML diagram type a toolbar containing only customized model elements (using UML stereotypes and a custom icon notation) valid for the respective diagram type. The advantage of Magic Draw beside the very quick implementation of the Profiles is its very good basic usability. Professional UML tools like Magic Draw provide high robustness, very quick creation of models, various support for diagram layout, easy switch between different notations, functions for managing large models, etc. This is very important for the modelers in practice.

Figure 6.28 and 6.29 show screenshots from Magic Draw with the custom MML diagrams. They show as example an MML Structure Diagram (figure 6.28) and a MML Presentation Diagram. The customized toolbars contain only the MML-specific elements for the diagrams, e.g. Media Components and Media Parts in the Structural Diagram or Abstract Interation Objects in the Presentation Diagram. The MML-specific visual notation is supported to a large extent. The plugins work for all Magic Draw versions since version 11.6 at least up to the current version 15. All MML models

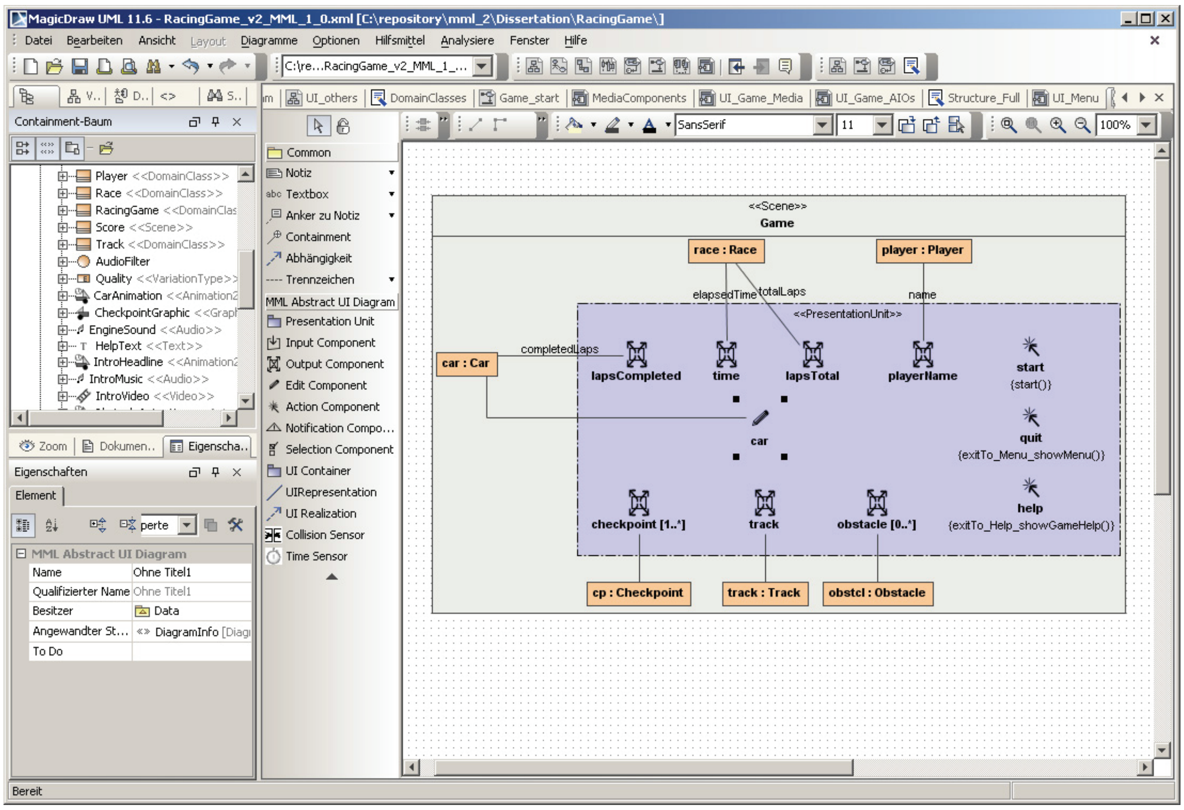


Figure 6.29: MML Presentation Model in Magic Draw.

depicted in this thesis have been created with the Magic Draw MML plugins.

Transformation from Magic Draw to EMF The final step in creating tool support was bridging the gap between Magic Draw and EMF, so that models created with Magic Draw can be further processed by the various Eclipse-based modeling tools. Magic Draw and EMF both save models in the XMI format. However, there is a large difference between them: in Magic Draw MML is defined as a UML Profile. Thus, the resulting models are UML models with some (MML) extensions (UML Stereotypes). In contrast, EMF handles MML as a language of its own without any relationship to UML.

To solve this issue, an XSLT transformation was created to automatically transform the models

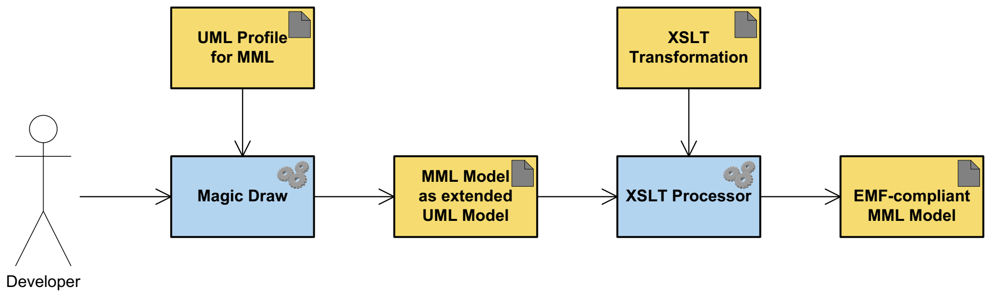


Figure 6.30: Steps for visually creating EMF-compliant MML models.

created with Magic Draw into EMF-compliant models. It was implemented in [Shaykhit06] and has been extended and maintained by the author of this thesis. Figure 6.30 illustrates the resulting tool chain. After this transformation, the models can be opened with the EMF-based tree editor described above and can be e.g. processed with ATL model transformations for code generation purposes.

Lessons Learned As concluding remarks for future work it can be said that the solution chosen here is a workaround. New frameworks like the Graphical Modeling Framework (*GMF* [GMF]), integrated in modeling environments like Topcased [Top], provide a good possibility to create real metamodel-compliant and custom visual modeling editors today. However, Magic Draw still remains as a very fast solution. In addition, the latest versions of Magic Draw provide even a "DSL-Engine" supporting custom metamodels and modeling constraints. But this has not been tested in the context of this thesis yet.

Using XSLT for the transformation turned out to be not an optimal solution. The XSLT transformation is complex and difficult to maintain. The main reason is that models contain many references between model elements. XSLT does not provide sufficient support for references, e.g. it is not directly possible to refer in the transformation to the target element created from a given source element. Thus, either such an mechanism must be implemented manually or the transformation has to be coded in an imperative style which makes it very complex. A better solution would be using a model transformation language. This is possible, as Magic Draw supports also exporting models into EMF UML2 [EMFa] format which is now for example supported by the transformation language ATL.

Chapter 7

Integrating Creative Design: From MML to Flash

This section presents the transformation from MML models to code skeletons. As example target platform Flash has been selected as it is one of the most relevant professional multimedia authoring tools today (see section 2.3.3). The chapter shows how the MML approach targets the integration of creative design into the systematic model-driven development process (goal 3 from page 34).

The transformation is performed in two steps. The first step is a model-to-model transformation from the platform-independent MML model to a model for the Flash platform. This step captures the conceptual mapping on the abstraction level of models – without the need to consider the concrete syntax and file formats of the target platform. The second step then transforms the Flash model into the final code skeletons. The transformations adhere to the concepts of Model Driven Engineering. They are specified as an explicit transformation written in the *Atlas Transformation Language* (*ATL*, [AMM], see sec. 3.4.2).

The chapter is structured as follows: The first section gives a more detailed introduction into the target platform Flash by hand of a metamodel: The metamodel has been defined in the context of MML for the transformation from MML to platform-specific Flash models. The second section discusses how the resulting Flash applications can be structured. This is necessary as no common systematic structuring mechanism for Flash application yet exists. The third section describes the transformations and a solution how to generate files in the authoring tool's proprietary, binary file format. While the foregoing sections are more technical, the fourth section finally illustrates by screenshots how easy to work with the generated code skeletons in the authoring tool. It demonstrates the tight integration of systematic model-driven development and the authoring tool's advanced support for creative design.

7.1 A Metamodel for Flash and ActionScript

This section gives an introduction to the authoring tool Flash and introduces a metamodel for this platform. Section 2.3.3 already provided a first general overview on Flash and ActionScript and argued why Flash has been selected as example platform. The current section goes more into the details of Flash which are required to understand the concepts for code generation. Thereby, the metamodel for Flash is introduced which provides an abstract overview on the elements in Flash and their relationships.

The section starts with an introduction of some general considerations for the Flash metamodel.

It consists of two parts – one for Flash Documents and one for ActionScript – which are explained in the succeeding two sections[1].

7.1.1 Flash Metamodel Basics

The metamodel presented here describes is platform-specific and describes applications to be developed with Flash and ActionScript. It is independent from MML and and enables to model any kind of Flash application. Analogous to the MML metamodel it is defined according to the concepts of Model-Driven Engineering (sec. 3.4) and has been implemented using the Eclipse Modeling Framework (see sec. 6.7). The basic metamodel principles and conventions for presentation applied here are the same as described in section 5.1.5. The Flash metamodel presented here adheres to Flash version 8. An overview on Flash versions and MML support is provided in table 7.1.

Purpose The main purpose of the metamodel in this thesis is to build the base for an intermediate step in the transformation from MML to the final code skeletons, i.e. a first transformation from MML to Flash models and a second transformation from Flash models to code. This provides several benefits: First, it separates the conceptual mapping from MML to Flash from the concrete syntax for Flash applications and is thus easier to maintain and extend. Second, the Flash metamodel and the transformation into the concrete Flash syntax can be directly reused by other modeling approaches which want to use Flash as target platform.

It is not necessary for the MML approach to edit the generated Flash models as they can be transformed directly into the final code. Instead, the Flash authoring tool should be used to finalize the application. Thus, no concrete visual syntax and no visual modeling tool has been defined for the Flash models yet. However, an EMF-based tree editor analogous to that for MML (see figure 6.27) exists for the Flash models. It can be used to check, modify, or even create Flash models if desired. Of course, it is also possible in the future to add a concrete syntax to the Flash metamodel and create and edit Flash models directly in a visual modeling tool.

Deriving the Metamodel from the Flash JavaScript API It is not always trivial to figure out all possible elements in Flash Documents and the relationships between them as they are partially hidden by the authoring tool. Existing literature (see section7.2) and documentation [Adobec][2] provide only step-by-step instruction for typical tasks in Flash but do not provide a precise systematic overview. Thus, this thesis proposes the solution to use the Flash JavaScript API to derive the metamodel in a systematic way. The following paragraph provides a brief excursus into the Flash JavaScript API and the associated scripting language JSFL. A basic understanding of them is also required later in section 7.3).

Excursus: JSFL and the Flash JavaScript API Since Flash MX 2004 it is possible to write custom extensions for the authoring tool. They have to be written in a scripting language *JSFL* (*Java Script for Flash*, [Yard and Peters04, Ado05c]). This is a version of JavaScript to be used together with the *Flash JavaScript API*. A JSFL file is a simple text file with the file extension *jsfl* which is interpreted and executed in the authoring tool. It can be created using the authoring tool or any text editor. It

[1]The term 'Flash' refers to the overall applications while 'Flash Document' refers to the Flash Documents only, i.e. FLA files without ActionScript classes).

[2][Adobec] refers to a complete online version of the Flash documentation. Each chapter is alternatively available for download as PDF manual. In the following we refer to the PDF manuals but the reader might also use [Adobec] instead.

is possbile to use it in combination with custom dialogue boxes defined in XMLUI as subset of the XML-based user interface description language XUL ([XUL], see also section 4.1.1).

The intended purpose of JSFL scripts is to extend the Flash authoring tool with custom functionality. Therefore, JSFL files and XMLUI files (and other resources) can be packed in a Flash Extension file with the file extension *mxi*. The mxi files can be distributed and installed as extension for the Flash authoring tool. Alternatively, it is possible to execute a JSFL file directly in the authoring tool. As third alternative, JSFL files can also be started directly from the command line which causes the Flash authoring tool to start and execute the file.

While JSFL itself is a simple JavaScript version without any specific features, the associated Flash JavaScript API provides full control over the Flash authoring tool. It provides access in terms of a *Document Object Model* (*DOM*) similar to that implemented by web browsers for HTML. The Flash DOM provides full access on the content of a Flash Document. It enables to browse, manipulate, and create new elements within the document. Moreover, another part of the API (sometimes called *Flash file API*) allows common file management functionality. Consequently, it is possible for instance to load a Flash Document into the authoring tool, manipulate it, and save it, or even to create an entirely new Flash Document via JSFL.

Listing 7.1 shows a simple example JSFL script. The variable fl in line 1 is a predefined global variable which refers to the *Flash Object* representing the Flash authoring tool. Here an operation of the Flash Object is used which retrieves the current active Flash Document in the tool. Line 2 draws a simple rectangle shape within the document. Finally, line 3 saves the document with the filename "MyDocument.fla".

```
var document = fl.getDocumentDOM();
document.addNewRectangle({left:0,top:0,right:100,bottom:100},0);
fl.saveDocument(document, "MyDocument.fla");
```

Listing 7.1: Simple example JSFL script.

Rules for Metamodel Derivation The metamodel for Flash Documents is derived from the Flash JavaScript API as follows:

- An API class representing an entity in a Flash Document (i.e. a part of the document itself instead of a functionality of the authoring tool) is mapped to a metaclass. Other API classes (like the Tools object [Ado05c]) are omitted in the metamodel.
- A property in the API representing a structural property (like the name) or a basic visual property (like x- and y-coordinates on the screen) is mapped to a property in the corresponding metaclass. Properties representing the internal state of the authoring tool or visual details are omitted.
- A properties or operation representing an reference to other API classes is mapped to an association in the metamodel.
- A generalization in the API is mapped to a generalization relationship in the metamodel.
- The API often uses class properties to specify the type of a class more in detail (like the property symbolType for the class Symbol). For each possible property value defined in the API a subclass is created in the metamodel.

In this way, also the semantics for the Flash metamodel is defined indirectly by the operational semantics of its counterpart in the Flash JavaScript API.

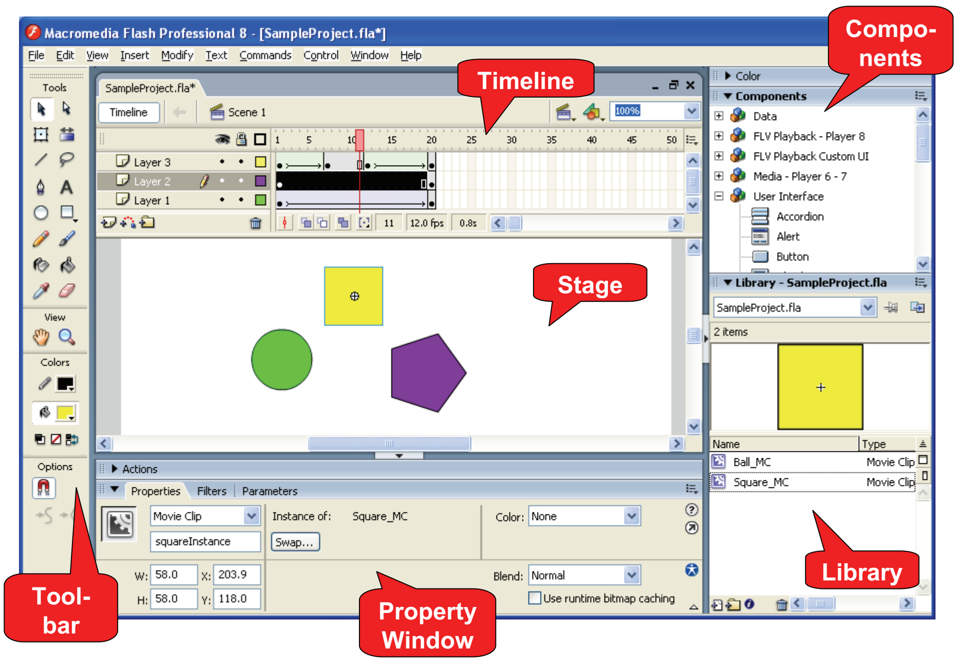


Figure 7.1: The Flash Authoring Tool

The Flash JavaScript API defines only Flash Documents. The resulting metamodel part is introduced in the next section (7.1.2). Afterwards section 7.1.3 introduces the metamodel part for ActionScript which is similar to metamodels for other programming languages like Java.

7.1.2 Flash Documents

As briefly introduced in section 2.3.3 the proprietary file format for Flash Documents is the *FLA* format. They are compiled into *SWF* files which can be executed with the Flash player. The Flash Documents are created in the Flash authoring tool.

Figure 7.1 shows a screenshot of the Flash authoring tool. Like many other development environments it provides large number of different windows and toolbars which can be individually arranged by the developers. In figure 7.1 only the most essential windows are visible. The center window shows the *Stage* which represents the actual 2D content visible to the user when the final application is executed. It contains as example three simple shapes (the rectangle, the circle, and the polygon). The *Toolbar* on the left hand side contains various drawing tools to create and manipulate 2D graphics on the Stage. The *Property Window* on the center bottom shows properties of the currently selected element, like its x- and y-coordinates, its size, an instance name, etc. On the right hand side there is a *Component Window* which offers several predefined user interface widgets – like buttons, lists, textfields, etc. – and the *Library*. The temporal dimension is represented by the *Timeline* in the top center window. Library and Timeline will be explained in the following in more detail.

Timeline The Timeline consists of multiple *Frames*. A Frame represents a point of time in the Timeline and is associated with content on the Stage. Figure 7.2 shows a larger image of the Timeline. The Frames are ordered in horizontal direction from left to right on the Timeline. The *Playhead* indicates the current Frame displayed on the stage. It is possible to play the Timeline directly in the

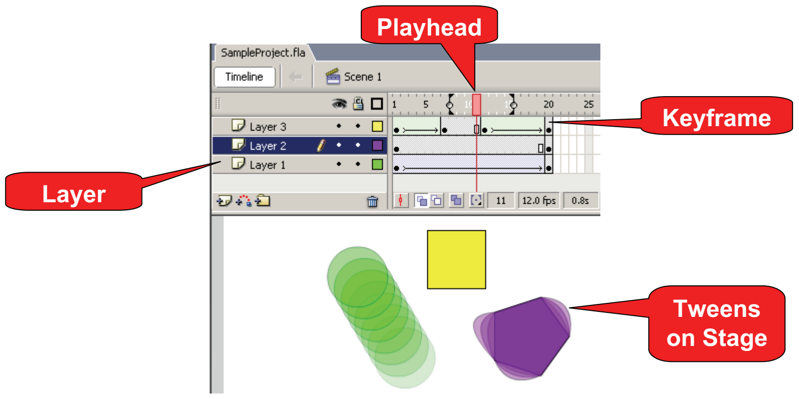


Figure 7.2: Timeline and Stage (containing some Tweens) in the Authoring Tool

authoring tool to preview the application: Then the Playhead moves continuously along the Frames. Otherwise, it is possible to select a Frame by placing the Playhead onto a Frame.

By default, the Timeline is played in a loop when a Flash application is executed. As this is to always useful it is possible to control the behavior of the Timeline by ActionScript commands. The script commands can be used for example to stop playing and set the Playhead to a specific Frame. In interactive applications the Timeline is usually used only for animations while the overall application is controlled by script commands. For instance, when the user clicks a button the Playhead is set to another Frame associated with different content.

In vertical direction the Timeline can be separated into multiple *Layers*. In the example there are three Layers named as Layer1, Layer2, and Layer3. They represent an additional z-axis for the content on the stage to define which object is the topmost when several objects overlap each other. Moreover, there are some specific kinds of Layers, for instance a *Guide Layer* which is invisible in the final application and is used to define a path along which an animation moves.

In the example, Frames are defined in all three Layers until frame 20. The Frames marked with a black filled ball represent *Keyframes*. Only Keyframes are associated with content defined by the developer. Other Frame's content is automatically derived from the foregoing Frames: If a Tweening is defined then the content is calculated accordingly, otherwise it remains the same as in the foregoing Keyframe.

Tweenings are used in Flash to visually create 2D animations. A Tweening is an automatic interpolation between two states of a graphical object. The start state and the end state are defined in Keyframes. Flash then automatically calculates the intermediate states in between these two Keyframes to create a smooth animation. A *Motion Tween* can manipulate the position, size, and rotation of a graphical object. A *Shape Tween* can manipulate the shape and color of a graphical object. In figure 7.2 the green ball on the Stage moves according to a Motion Tween and the polygon changes its shape defined by a Shape Tween. A Tween is indicated in the Timeline by an arrow between two Keyframes (figure 7.2, upper part). It is possible to fine-tune Tweenings by several parameters, for instance the animation's acceleration.

The metamodel extract in figure 7.3 shows the basic elements and their relationships: A Timeline is divided into one or more Layers. Each Layer consists of multiple frames. A Frame can be associated with graphical content on the Stage (represented by the abstract metaclass Element; see paragraph "Stage Content" below). If a Frame is associated with content then it is automatically a Keyframe. A Tween attached to a Frame runs until the next Keyframe. Finally, it is possible to attach *ActionScript code* (ASCode) to a Frame. The code is executed each time the application enters the Frame.

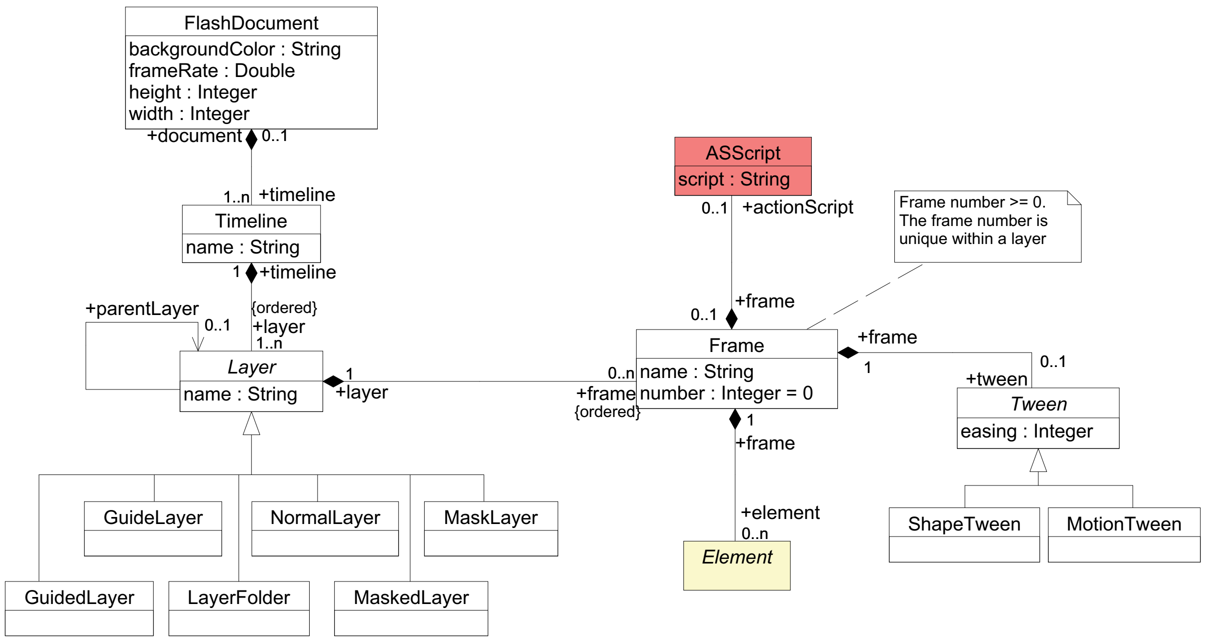


Figure 7.3: Metamodel for the Timeline Elements in Flash

Library Flash supports a prototype-based paradigm, i.e. it is possible to create multiple instances of a media element. When a new media object is created then it automatically serves as prototype for potential further instances. For this purpose, it is automatically added to the *Library* – for instance, when the developer imports a bitmap or a video into the authoring tool. The so-called *Items* in the Library can be instantiated in any number by just dragging them onto the Stage.

A specific type of Library Item in Flash is the so-called *Symbol*. A Symbol is a complex 2D graphics element. There are three kinds of Symbols: A *Button* is used to visually create custom buttons, i.e. user interface elements which can be used to trigger some behavior. A so-called *Graphic* is static graphic or a simple kind of animation.

The most important type of Symbol is the *MovieClip*. MovieClips are the most frequently used type of user interface element in Flash as it is the most flexible and powerful kind of element. A MovieClip is an arbitrary complex 2D graphic or animation. It owns a Timeline of its own that is independent from the application's main Timeline. The Frame's in its internal Timeline can contain all content like the main Timeline, including other media objects, interactive controls, ActionScript code, etc. As it can contain other MovieClip instances as well, it is possible to nest MovieClips up to any depth. Nesting MovieClips is a very common mechanism to achieve complex animations. For instance, creating an animated character is usually performed by nested MovieClips for its different body parts, e.g. a nested MovieClip moves the character's arms when the character moves and in turn another nested MovieClip moves the character's hands when its arms moves.

Since ActionScript supports object-oriented concepts it is now possible to attach a custom ActionScript class to a MovieClip. Then each MovieClip instance is automatically associated with an instance of this ActionScript class. Section 7.1.3 will provide more details on ActionScript.

Besides Symbols, Flash supports Bitmaps, Sound, and Video. Text is not supported by the Library; it is created directly on the Stage or imported from from external text files. 3D graphics is currently not supported by Flash yet.

Figure 7.4 shows the metamodel for Library Items. Each Flash Document contains a Library

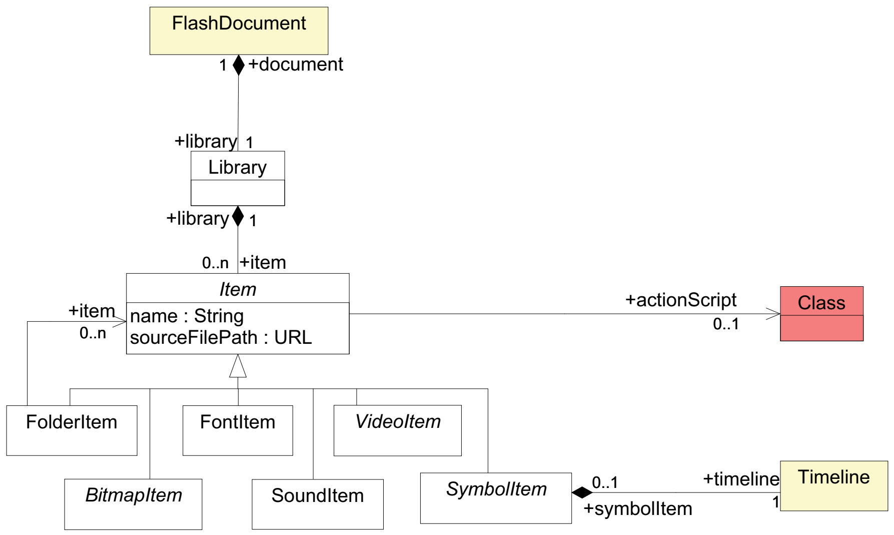


Figure 7.4: Metamodel for the Library Elements in Flash

containing Library Items. Beside the types of Items introduced above there is a FontItem which represents a specific text font and a FolderItem which is used to structure the Library into folders and subfolders. Figure 7.5 shows the concrete subclasses for media formats supported by Flash (see [Ado05c] for details). Some kinds of Items (usually MovieClips) can be associated with ActionScript classes. A *SymbolItem* owns a Timeline of its own.

Stage Content The content on the Stage can be either created directly with tools from the Toolbar (shapes and text) or by instantiating an Item from the Library. Beside Library Items it is also possible to instantiate a predefined Component from the Component Palette (upper left in fig. 7.1). Figure 7.6 shows the corresponding metamodel.

As visible in the metamodel, it is also possible in Flash to assign ActionScript code to a single SymbolInstance. However, this mechanism is deprecated and should no longer be used.

Files and Folders A Flash Application often consists of multiple Flash Documents. Moreover, it can include external media files, like audio or video files. Often it is useful to structure them into folders in the file system.

Figure 7.7 shows the metamodel. The metaclass *FlashApplication* is the metamodel's root element. It consists of different files (abstracted by the metaclass *MediaArtifact*) including FlashDocument (which have been further specified above), *MediaFile*, and *Folder*.

7.1.3 ActionScript

ActionScript is the built-in scripting language in Flash. As briefly introduced in section 2.3.3 ActionScript is close to the ECMA standard [Ecm99] and supports object-oriented constructs since version 2. The following section introduces ActionScript 2 and presents a metamodel.

Placement in Flash Documents The starting point for Flash applications is always a Flash Document (explained in the foregoing section). There is no kind of main() operation in ActionScript. Even

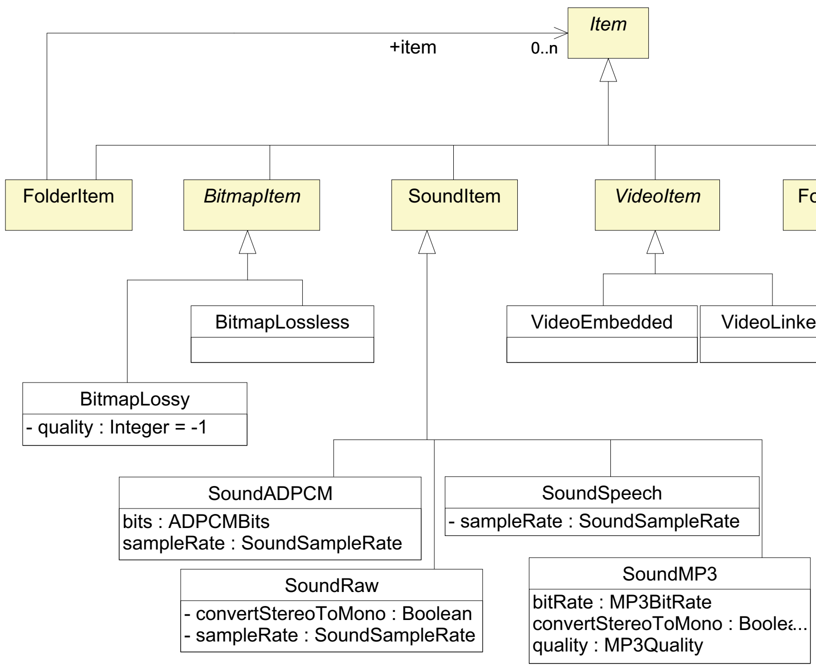


Figure 7.5: Metamodel for the specific Media Formats in Flash

if a Flash application consists only of ActionScript code it is still necessary to have a Flash Document around which invokes the ActionScript code.

According to the metamodel in section 7.1.2 there are three kinds where to place ActionScript code within a Flash Document:

1. **Frames:** It is possible to add code to a Frame in the Timeline (fig. 7.3). The code is executed each time the Playhead enters the Frame.

2. **Symbols:** It is possible to add code to a Symbol – usually a MovieClip – in the Library (fig. 7.4). It must be an ActionScript class which is then associated with the MovieClip. Each instance of the MovieClip then is represented by an instance of the corresponding ActionScript class. An ActionScript class must be a external text file, similar like a class file in Java.

3. **Instances:** It is possible to add code directly to a Symbol Instance (fig. 7.6). In this case the possible code is restricted to event handling code. It is executed each time the corresponding event occurs at this instance.

Consequently, there are two alternatives how code is created:

1. **Embedded Code:** Code added to a Frame or to an Instance is directly embedded in the Flash Document. The developer types the code into the *Actions* window in the authoring tool. The Actions window provides the basic features of simple source code editors, like a simple way of syntax checking.

2. **External Code:** ActionScript classes are external class files analogous to class files in other programming language like Java. They can be created in the Actions window as well but also with any other external text editor. There are also plugins for ActionScript support for programming environments like Eclipse [Powerflasher, ASD].

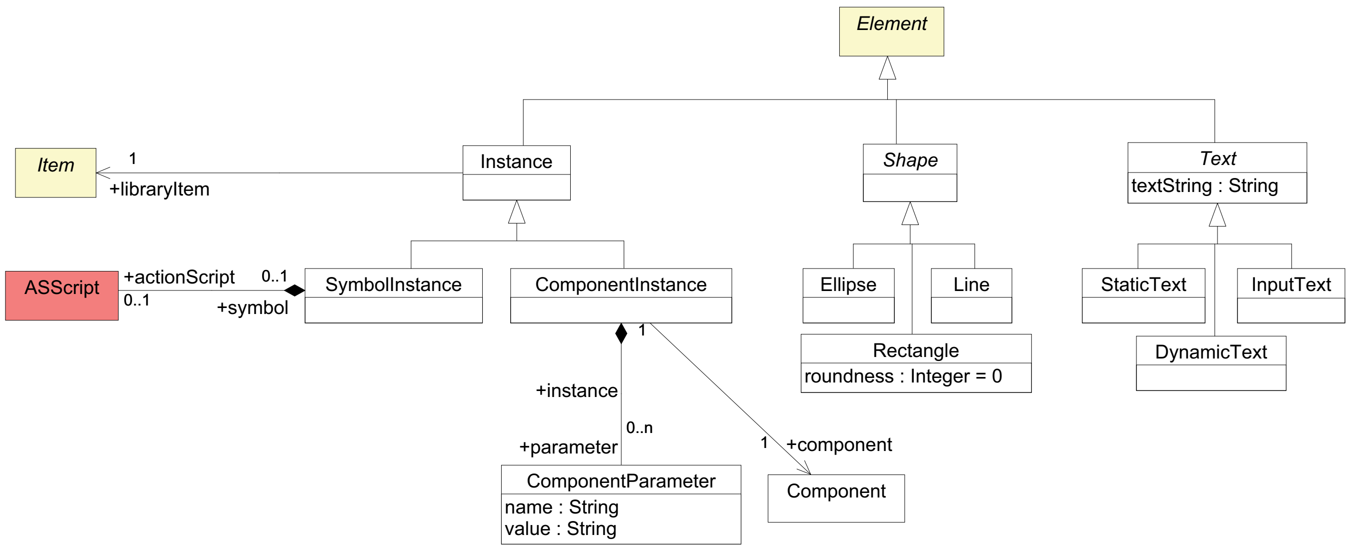


Figure 7.6: Metamodel for the Elements on the Stage in Flash

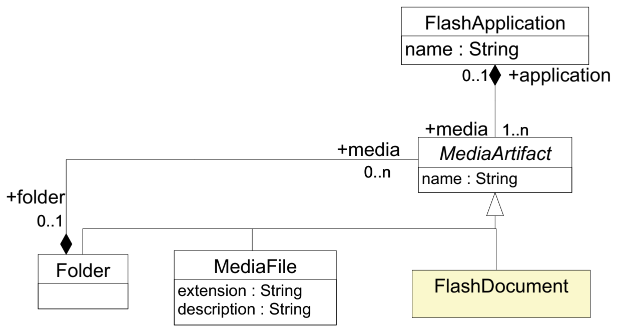


Figure 7.7: Metamodel for Files and Folders

Language Basics The language's syntax complies to the ECMA standard and is thus similar to JavaScript. Attributes and variables are defined using the keyword var:

```
var myText:String = "Hello World";
```

Functions are defined using the keyword function:

```
function increment(i:Number):Number {...}
```

ActionScript 2 offers the following primitive types: *String*, *Boolean*, *Number*, *Undefined*, and *Void*. *Number* is the only numeric type in ActionScript and contains double-precision floating point numbers as well as 64-bit integers. *Undefined* indicates that a variable has no value or an object instance does not exist yet. *Void* is used to specify that a function has no return value.

Listing 7.2 shows an example for a class definition. ActionScript 2 also supports interfaces and packages.

```
import game.*;                  //imports all classes from package 'game'

class game.Car extends Vehicle {    //class is in package 'game'

    private var name:String;            //class attributes
    private var topspeed:Number;
    public function Car(name) {         //constructor (has no return value)
        this.name = name;
    }

    public function getName():String {  //another class operation
        return this.name;
    }
}
```

Listing 7.2: Example class in ActionScript.

Attributes and operations of an object are accessed by the '.'-syntax:

`s = car1.getName();`

As ActionScript originally is object-based it is also possible to add new properties or functions to an object at runtime just by the '.'-syntax. For example:

```
o = new Object();
o.myProperty:Number = 5;
```

creates a new object and adds a new property myProperty to the object having the value "5". A new function can be added to the an object as follows:

`o.myFunction = function(param:Number):Number {...}`

This mechanism is often used in Flash to define operations of event listeners as ActionScript 2 does not support anonymous inner classes like e.g. in Java.

Built-in Classes and Components ActionScript includes various built-in classes which allow to control the application. Some of them directly correspond to visual elements in the Flash Document and provide an interface to control it via ActionScript. An important example is the *MovieClip* class. To give the reader an idea of its properties and operations they are listed in appendix B.

Other classes provide additional functionality which can not be achieved in the authoring tool without ActionScript. An example is the *XML* class which provides support to to load, parse, send, build, and manipulate XML documents. All ActionScript classes are subclasses of the class *Object* which provides basic properties and operations inherited by all other classes. A complete documentation on ActionScript classes can be found in [Ado05a].

The Components from the Component Window (see sec. 7.1.2) are represented by ActionScript classes as well. Basically, each component in Flash consists of a visual representation, a set of parameters, and an ActionScript class. It is possible to create custom components and add them to the authoring tool. The set of standard components delivered with the authoring tool includes components for the user interface (widgets), media (like a video player), or data handling (e.g. for connection to web services). The component reference for Flash 8 is available in [Ado05b].

Figure 7.8 shows the metamodel part for ActionScript. FlashApplication is the root element like in fig. 7.7. According to the descriptions above, ActionScript elements can be classified in ActionScriptArtifacts defined by the developer – like classes, interfaces, and packages – and BuiltInTypes – like BuiltInClass, Component, and PrimitiveType. All of them, except PrimitiveType, are a subclass of Classifier. Classifier and PrimitiveType both are a Type.

The lower part of figure 7.8 further specifies Classifier. This part is similar to metamodels for other object-oriented languages like Java [Obj04a, INR05]. ActionScript statements in operation bodies are not further modeled by the metamodel. They are just handled as strings so that source code can be specified directly. Otherwise the models would become too complicated and the additional level of abstraction provides no additional benefit for our purposes. Likewise, embedded ActionScript code

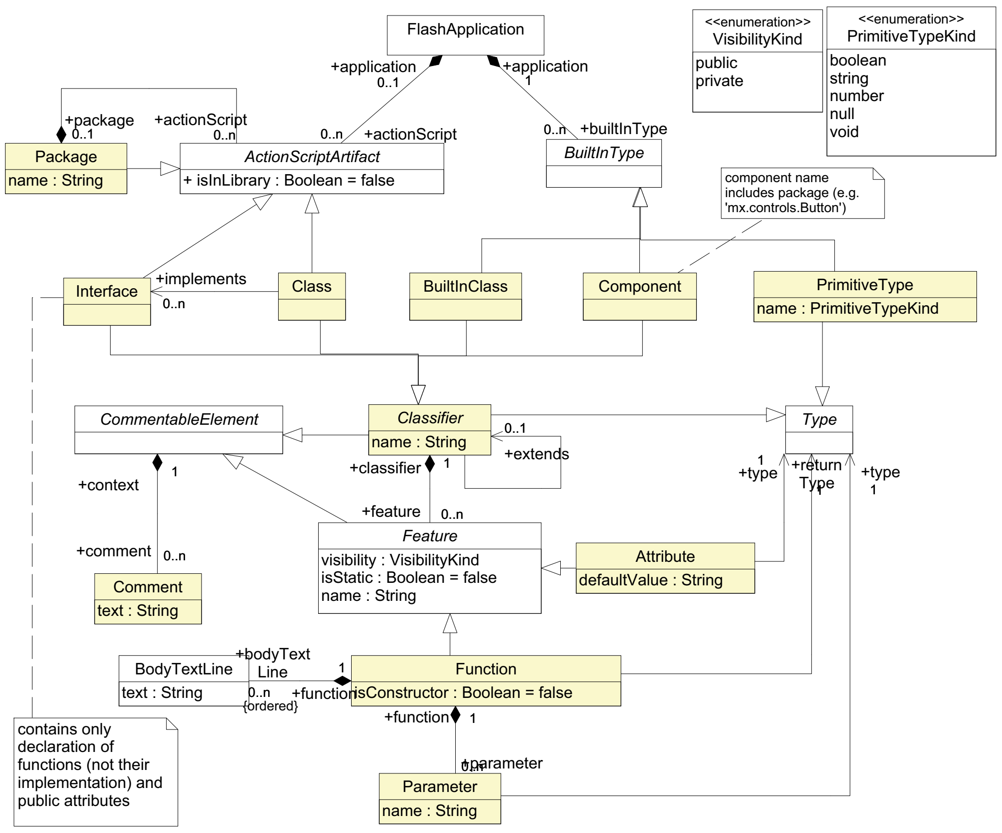


Figure 7.8: Metamodel for ActionScript

attached to a Frame in the Timeline or to a MovieClip instance is specified directly as a string (see class ASScript in the metamodels in fig. 7.3 and 7.6).

The metamodel part presented here, together with the metamodel parts fro Flash Documents from section 7.1.2, build the overall metamodel for Flash applications.

Accessing Stage Content So far, the metamodel has been presented and where ActionScript can be placed within a Flash Document. This paragraph now explains how ActionScript code can access the visual content in a Flash Document.

The authoring tool enables to assign an instance name to each instance on the Stage (fig. 7.6). These elements can be accessed from ActionScript code by their instance name. Thereby, the available namespace depends on where the ActionScript code is placed:

- Script code in a Timeline can directly access instances located in Frames on that Timeline by their instance name.
- A class assigned to a MovieClip can directly access instances on the MovieClip's Timeline as class attributes by their instance name.
- It is possible to navigate within the hierarchy of Timelines using the '.'syntax. For example if the main Timeline contains a MovieClip instance car1 which in turn contains an instance frontwheel_left it is possible to access the frontwheel from the Timeline by car1.frontwheel_left. The property _parent refers to the parent Timeline of a MovieClip. The global variable _root refers to the main Timeline of a Flash Document.

When assigning a custom ActionScript class to a MovieClip it is useful to specify the custom class as subclass of the ActionScript class MovieClip. In that way the custom class inherits the properties and operations of MovieClip (see appendix B). In particular, the custom class then can override the event handling operations like onPress(). This is an easy way to define custom event handling code for a MovieClips on the Stage.

It is also possible to create or remove Stage content dynamically by ActionScript code. In ActionScript 2 the developer must use one the predefined MovieClip operations attachMovie, createEmptyMovieClip, or duplicateMovieClip (see appendix B) which provide the created MovieClip as return result. (In ActionScript 2 there is no possibility to create a MovieClip in a more object-oriented way using e.g. the keyword new).

7.2 A General Structure for Flash Applications

For the transformation from MML models into Flash code it is necessary to select an applicable structure for the resulting applications. This could be a kind of "framework" where the platform-independent MML models can be mapped to. The specific problem for authoring tools like Flash is that common structuring techniques and patterns from other programming languages alone do not help – the application structure in Flash must integrate Flash Documents as well. Moreover, Flash provides different ways to connect code with the Flash Documents (see sec. 7.2) but none of them can be used for all purposes.

7.2.1 Principles

This section describes the problem and elaborates general principles for a framework for structuring Flash applications.

Problem When reviewing the existing literature and resources on the web, it turns out that such a framework currently not exist. When reviewing the existing literature one can observe that it reflects again the central observation from the end of section 2.1.3 – it can be classified in two ctaegories:

- **Visual Design/Authoring:** Books and articles focusing on the authoring tool like [Franklin and Makar03, Macromedia03, Kannengiesser and Kannengiesser06, Wenz et al.07] usually provide step-by-step instructions how to perform common tasks in the authoring tool. The main ActionScript code is usually just placed into the first Frame of the main Timeline. Since ActionScript 2 most books promote to use ActionScript classes for the application's main entities and also give some general advice on object-orientation like basic design patterns. But this usually affects only the Domain Classes themselves but still the remaining code (main programm flow, user interface management, event handling, etc.) still remains as a monolithic block in the Timeline. Other, purely design-related books like [Dawes01, Capraro et al.04] do not cover ActionScript at all.
- **Programming:** In contrast, several books like [Hall and Wan02, Moock04, Moock07] indeed focus on a well-structured application. They show how to structure the whole application in an object-oriented way using ActionScript classes. Unfortunately, these approaches do not use the authoring tool at all. Instead, the whole user interface is defined by ActionScript code – e.g. drawing MovieClips by ActionScript operations. This is a major drawback as abstaining from the authoring tool capabilities results in a step backwards concerning the visual creative design.

Existing frameworks like *Flex* (page 18) and related frameworks like *Cairngorm* [Adobea] also fall into this category.

Consequently, to avoid the drawbacks of both viewpoints found in literature it is necessary to design a framework which integrates Flash Documents for the visual elements to be designed in the authoring tool and well-structured object-oriented ActionScript code for the non-visual, functional application parts. A first step into this direction is given in [deHaan06] which discusses existing mechanisms and gives useful recommendations which are considered in the structure proposed here.

Principles As concluded above, the basic goal for the framework is to integrate Flash Documents for the visual elements with well-structured application code. Nevertheless, there are still various different solution how to realize such an application structure in Flash. Basically, simple straightforward structure can sometimes be useful, e.g. for smaller applications or for lightweight applications like for mobile devices. Here we aim for a very modular and flexible structure which is suitable to fit for large and complex projects. It is then still possible to simplify the structure if desired (see sec. 8.1.3).

Thus, the framework here aims to provide a large degree of modularity. This enables scalability up to large and complex applications and provides large developer groups to work in parallel. The latter is especially important for Flash applications as Flash provides only poor support for cooperative work on single Flash Documents. As Flash Documents are binary files they also can not be handled using conventional file version control systems. Thus, it can be significant for large projects to systematically divide the application into multiple Flash Documents.

The solution proposed here thus applies the following principles:

1. Modular, object-oriented style: As far as possible all ActionScript code is placed in external ActionScript classes.

2. Separation of different aspects: Aspects like user interface, interaction, and application logic are separated. Therefore, the framework orients towards the common *Model-View-Controller* pattern (*MVC*, [Krasner and Pope88]) which is also proposed in [deHaan06].

3. Separation of the user interface: Different Scenes (see sec. 6.3) of the application are separated into different Flash Documents so that they can be edited independently from each other. Moreover, this enables that each Scene in the final application can be loaded on demand (instead of being forced to load the whole application at once) which is a common mechanism in web-based Flash applications. It is possible to load them in background and perform other operations during the loading time.

4. Separation of Media Components: Complex Media Components are placed into a document of their own so that it can be edited independently from other documents. Moreover, this enables to apply a Media Component multiple times within different Scenes.

The following paragraphs present the realization of these principles and the resulting structure.

7.2.2 Scenes

The user interface should be modularized into multiple Scenes realized by independent Flash Documents (principle 3). ActionScript code for the Scene (e.g. to initialize its user interface and establish relationships to the application logic) should be placed in an ActionScript class associated with the Scene's Flash Document (principle 1). In addition, a mechanism is required to perform the transitions between Scenes at runtime.

```
class MyScene extends MovieClip implements Scene {

    private static var theInstance : MyScene;

    private function Scene1() {
    }

    public static function loadInstance():Scene {
        if(theInstance == null) {
            theInstance = new MyScene();
        }
        return theInstance;
    }
}
```

Listing 7.3: Implementation of the Singleton pattern in an example Scene class MyScene.

Accessing Scenes Basically, two different principles can be considered for their implementation: Either manage the different Scenes by a central instance (using e.g. the design pattern *State* [Gamma et al.95]) or by implementing them as independent objects without a central instance (using e.g. the design pattern *Singleton* [Gamma et al.95]). Here, the latter approach is used as it is more modular and enables to add additional Scenes more easily to the application. Therefore the Scenes are implemented using the *Singleton* design pattern so that they can be accessed and activated from any other Scene. Using Singleton, the Scene class instantiates itself and thus has full control whether to keep or re-initialize its inner state (see also "Resuming Scenes" on page 111). The transitions are managed directly by the Scene class itself as well.

Listing 7.3 shows the resulting code for an example Scene class MyScene. The class constructor is declared as private according to the Singleton pattern. The class provides a static operation which returns the current unique instance of the class. Its name is in the original Singleton pattern is `getInstance()`. As for our purposes we have to combine this with a loading mechanism (explained below) its name is `loadInstance()`. All concrete Scenes are subclasses of the interface Scene defined in our framework.

Loading Scenes As mentioned above, it should be possible to load Scenes separately. Be default, Scenes are loaded on demand, i.e. when a Scene should be invoked the first time. To allow execution of other operations during the loading time, an event listener (called `SceneLoadingListener`) is passed to the Scene which notifies (call of operation `sceneLoaded(s: Scene)`) when the loading process has finished. Listing 7.4 shows the resulting implementation of the operation `loadInstance()`.

The loading itself is performed using the built-in ActionScript class MovieClipLoader. As explained on page 150 listeners can be implemented using the object-based mechanisms of ActionScript as ActionScript 2 does not support anonymous inner classes.

Listing 7.5 shows how another Scene defines a SceneLoadingListener to load MyScene. In addition, it calls as example the operation startMyScene of MyScene. The example also shows how a parameter can be passed (here to the operation startMyScene).

Connecting a Scene with an ActionScript Class In the Flash versions examined here, it is only possible to associate an ActionScript class to a MovieClip. As Flash Documents are no MovieClips, it is remains the question how a Flash Document representing a Scene can be associated with the Ac-

```
public static function loadInstance(sll:SceneLoadingListener):Void {
    if(theInstance == null) {
        var loader = new MovieClipLoader(); //use built-in class MovieClipLoader
        loader.onLoadInit = function(loaded_mc) { //define event listener:
            sll.sceneLoaded(loaded_mc);            //notify sll when loaded
        }
        loader.loadClip("MyScene.swf", container); //start loading
    }
    else{  //if already loaded then just pass it to sll
        sl.sceneLoaded(theInstance);
    }
}
```

Listing 7.4: The revised operation loadInstance with a loading mechanism

```
public function invokeMyScene():Void {
    var listener = new DefaultSceneLoadingListener();
    var oldScene = this;
    var parameter = 5;
    listener.sceneLoaded = function (s: Scene) { // Define event handling
        oldScene._visible = false; // Old Scene must be set invisible
        MyScene(s).startMyScene(parameter); // Call operation 'startMyScene' of
            loaded Scene
    }
    MyScene.loadInstance(listener);
}
```

Listing 7.5: Example how to use the SceneLoadingListener to load the Scene MyScene and invoke an example operation startMyScene.

tionScript class for the Scene. Our proposed structure provides the following solution: A MovieClip is created which has no visual representation of its own and acts as container for the Scenes. When a Scene is loaded it is placed into the container MovieClip. This can be achieved using the built-in ActionScript class MovieClipLoader whose operation loadClip(url:String, target:Object) specifies as target a MovieClip where an SWF file from the URL is loaded into. The ActionScript class for the Scene can then be assigned to the container MovieClip.

Listing 7.6 shows the resulting (final) version of the operation `loadInstance()`.

Alternatives Flash offers some alternative mechanisms which should be briefly discussed in context of Scenes:

- **Flash "Scenes":** Earlier versions of Flash already provided a mechanism called "Scenes" to divide the user interface into multiple independent units. However, it is deprecated today for several reasons explained in [deHaan06].
- **Flash "Screens":** Since Flash 8[3] it is possible to structure an application into multiple *Screens* which are a quite similar concept like the Scene concept here in this thesis. It is also possible to load the content of a Screen dynamically from a separate document and to associate a Screen with a custom ActionScript class. Moreover, the Flash authoring provides an additional view

[3]in *Flash Professional* edition only

```
public static function loadInstance(sl:util.SceneLoadingListener):Void {
    if(theInstance == null) {
        var depth = _root.getNextHighestDepth(); // required for new MovieClip
        var container = _root.createEmptyMovieClip("container"+depth, depth); //
            create new container MovieClip
        var loader = new MovieClipLoader();
        loader.onLoadInit = function(loaded_mc) {
            loaded_mc.__proto__= new MyScene(); //attach MyScene as class to
                loaded_mc
            sl.sceneLoaded(loaded_mc);
        }
        loader.loadClip("MyScene.swf", container);
    }
    else{
        sl.sceneLoaded(theInstance);
    }
}
```

Listing 7.6: Final version of operation *loadInstance* in class *Scene1*

for Screen-based applications showing the application's Screen hierarchy and the transitions between the Screens.

A drawback of Screen usage is that Screens are not completely transparent, i.e. some of their properties and behavior can not be (directly) accessed by ActionScript code. For instance, the transitions between the Screens must be defined by predefined "Behaviors" instead of within ActionScript classes. Thus, the overall structure using Screens becomes even more complex. Nevertheless, the Screens concept fits well to the structure proposed here and could be a useful future extension.

7.2.3 Complete Structure

The Scene concept described in the last section builds the core for the application structure. This section describes the other elements according to the principles from section 7.2.1 and shows the resulting overall structure. Figure 7.9 exemplifies the structure by an extract of the Racing Game example from chapters5 and 6.

Application Logic The application logic is implemented in conventional way as ActionScript classes (principle 1). They constitute the 'model' in terms of MVC and are placed in a package model (principle 2).

In the Racing Game example, the folder model would contain ActionScript classes Car, Track, Player, etc. (fig. 7.9).

Media According to principle 4, (complex) Media Components are located in separate files in a central folder media so that it is possible to apply them multiple times within the application. Graphics and animations are implemented as a Flash Document containing the graphic or animation as MovieClip. In general, audio, video and images are not created in Flash itself but just imported into Flash Documents. Thus, within our structure, the audio, video or image files are placed into the folder media as well from where they can be imported into multiple Flash Documents.

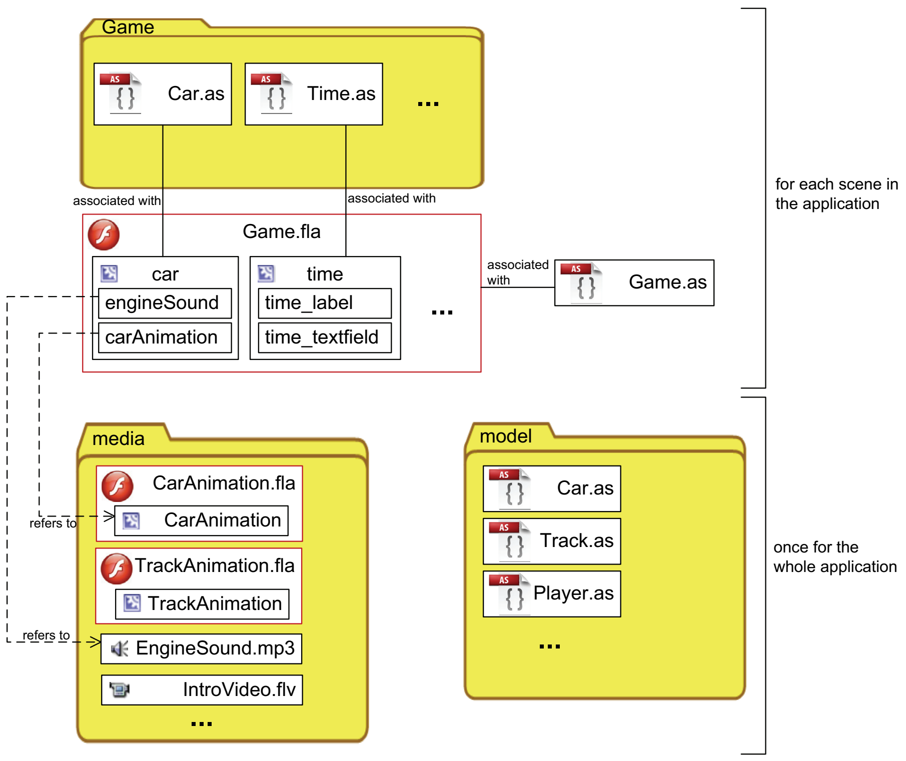


Figure 7.9: Proposed Structure applied to the Racing Game example.

To apply media from folder media in different Scenes it is possible to either create a reference in the Scenes' Libraries or to load them dynamically during runtime into the Scenes. For the purpose here, the former possibility should be preferred as it enables the developer to visually integrate the external MovieClip into the user interface at authoring time. A reference in the Library can be specified in the properties dialog of Library Items (invoked in their context-menu). For example, in case of a MovieClip, it is possible to specify a reference on a MovieClip in an external Flash Document. As result, the MovieClip adopts the content from the referenced external file. The relationship between source and target MovieClip is by reference, i.e. changes in the external MovieClip are adopted in the referencing MovieClip.

The folder media in in figure 7.9 contains for instance a Flash Document CarAnimation.fla which contains the actual MovieClip CarAnimation. It also contains sound and video files. The files in the folder media are referenced from elements e.g. in the Scene Game.

User Interface Elements The user interface elements are located in the Scene on the Stage. They are either instances of Media or from Flash user interface components. In our structure they constitute the 'view' in terms of the MVC pattern (principle 2).

The user interface elements are associated with ActionScript classes as well (principle 1). It depends on the type of element whether this is directly possible in Flash. For instance, for Flash user interface components can be not be associated with custom classes directly. It is also not possible to subtype them as no inheritance mechanism exists for Flash Components. Thus, all user interface elements are encapsulated into MovieClips which are associated with an ActionScript class. As explained in section 7.1.3 ("Accessing Stage Content") the MovieClip's content (e.g. a Flash Component) can be accessed in the ActionScript class as class properties via their instance name. The ActionScript classes are placed into a folder with the name of the Scene they belong to.

In the example in figure 7.9 the Scene Game contains for instance the user interface elements car and time. They are encapsulated into MovieClips so that they can be associated to the ActionScript classes Car and Time in the folder Game. The MovieClip time conatins two Flash Components, a text label for the name and a text field for the actual value. The MovieClip car contains two media instances engineSound and carAnimation which refer to EngineSound.mp3 and the MovieClip CarAnimation in the central folder media.

The relationship between user interface elements and application logic is implemented by the design pattern *Observer* [Gamma et al.95] (associations between ActionScript classes are not shown in fig. 7.9). 'View' classes thus implement the interface Observer while 'model' classes extend the class Observable and notify their Observers after they have changed.

Like in many other implementations the 'Controller' part (in terms of MVC) is simplified by placing the event listening operations directly into the 'View' class to avoid a very large number of small classes. Event handling operations are specified either by overwriting operations of the MovieClip directly (e.g. defining an operation onKeyDown()) or by attaching anonymous listeners to its content (e.g. a mouse listener for a contained user interface component). As explained in section 7.2.2 ("Loading Scenes") ActionScript 2 does not support anonymous inner classes but the object-based mechanisms can be used instead.

Scenes Finally, at some point the application must be initialized and the relationships between 'Model', 'View', and 'Controller' must be established. As the Scenes are the application's main building blocks they are used for this task. They contain the user interface elements on their stage so that they are available as class properties in the Scene's ActionScript class. This class thus initializes the application logic by creating new domain instances or receiving them by parameters of Entry Operations. It initializes the connections to 'View' classes by (due to the Observer pattern) calling addObserver() operations. Besides, the Scenes are implemented as described in section 7.2.2.

Figure 7.9 shows as example the Scene Game. It is represented by a Flash Document containing user interface elements, an associated ActionScript class, and a folder containing the ActionScript classes for the user interface elements.

7.3 Transformations

This section describes the transformation from MML models into skeletons for Flash applications. According to the general idea described in section 3.3, the overall application structure is generated from the models while for the concrete Media Components, final visual layout, and detailed application logic only placeholders are generated to be filled out in the authoring tool.

As explained in section 7.1.1 the transformation is split into two steps. The first step performs the conceptual mapping from MML models into Flash models while the second step performs the mapping to the final code.

MML to Flash Model The first transformation is a conventional model-to-model transformation from platform-independent MML models to platform-specific Flash Models. The Flash models comply to the Flash metamodel introduced in section 7.1. Moreover, in our approach the resulting application structure should also comply to the framework introduced in the previous section 7.2.

The detailed mapping rules are described as text in appendix C. The main ideas are summarized in the following.

Domain Classes are mapped to ActionScript classes in the folder model. As operation bodies (i.e. the detailed domain logic) is not specified in MML, they are also not part of the transformation and have to be filled out manually by the software developer.

Media Components are mapped to placeholders in the folder media. Graphics and 2D Animations are mapped to FLA files containing a placeholder MovieClip in the library. The placeholder can be a simple rectangle shape with text inside showing the Media Component's name. The media designer can just replace the placeholder with the actual graphic or animation. Other media types are mapped to corresponding files containing dummy content – e.g. a simple dummy image, video, or sound – to be replaced by the media designer.

A Scene is mapped to a FLA file containing its user interface, an ActionScript class, and a folder containing ActionScript classes for each user interface element. The FLA file contains instances of user interface elements. The ActionScript class contains the user interface elements as class properties. It also contains the domain objects specified for the Scene as class properties. Moreover, it initializes them by e.g. attaching user interface elements as Observer to domain objects. It also contains the code for Entry- and Exit-Operations and the Transitions between the Scenes. The basic code generated for the Scene class follows that proposed in section 7.2.2.

An AIOs is mapped to a MovieClip in the Scene and to an ActionScript class in the folder generated for the Scene. The MovieClip is located in the Scene's library and instantiated on the Scene's stage. It encapsulates instances of widgets (Flash Components) and/or Media Instances (if the AIO is realized by a Media Component). In the latter case, a reference is generated to the Media Component in the folder media (see sec. 7.2.3). The ActionScript class contains event handling operations and is attached to the MovieClip. It also implements the Observer pattern.

A detailed description of the mapping is described in appendix C. A fully functional prototype has been implemented in ATL.

Extended Media Support by Third-Party Tools The basic mapping described here focuses on Graphics and 2D Animations as they can be directly created within Flash. However, it is of course possible to generate additional code to integrates third-party tools into the development process. The diploma thesis in [Shaykhit07], supervised by the author of this thesis, discusses and implements several examples into this direction: For instance, videos in Flash must be in FLV format. However, in practice, videos are often encoded in a different video format like MPEG or AVI. Moreover, often a video should be encoded in different qualities, e.g. as specified by Variations (sec. 5.2.10). The approach in [Shaykhit07] uses a command line based video encoder *FFmpeg* [FFm] to address these issues. The URL of a source video can be specified as additional property in the MML model. In [Shaykhit07] the transformations have been extended to generate a shell script (a batch file in Windows) from the information in the MML model to automatically execute FFmpeg and convert all source videos into the FLV format – possibly in different versions with different qualities.

In general, by generating shell script commands from the models it is possible to integrate various advanced support for Media Components. The shell scripts are executed in the same step like the JSFL script for the Flash Documents. Of course, it is possible to generate a shell script which automatically invokes all other generated scripts, including the JSFL script, so that the developer needs only to start a single file.

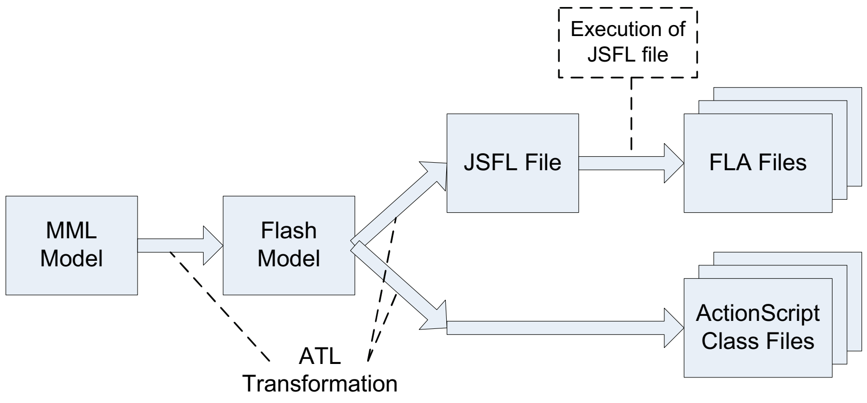


Figure 7.10: Transformation of MML models into Flash application skeletons.

Flash Model to Code This transformation step maps the Flash model into the concrete implementation, i.e. Flash Documents (FLA files) and ActionScript code. However, this step rises a problem: ActionScript classes can be generated in a straightforward manner as they are simple source code files analogous to Java files. Thus, a conventional model transformation language can be used for their generation. However, Flash Documents (FLA files) can not be generated directly as the FLA format is a proprietary binary file format. The compiled format for Flash applications (SWF files) has been made publically available by Adobe [Ado08] but it does not help here as the generated code skeletons should be processed in the authoring tool. Basically, it is possible to reverse engineer a SWF file into a FLA file using third-party tools like [Softwareb, Softwarea] but then still much authoring information – like Symbol names – is missing.

Thus, MML proposes a different solution to create Flash Documents: As introduced in section 7.1.1, JSFL scripts enable to create Flash Documents automatically. JSFL is well supported by Adobe and the scripts are conventional text-based source code files which can be directly generated by a transformation. Thus, the part of a Flash Model which describes the Flash Documents is mapped to a JSFL file in the transformation. Using JSFL, the mapping to Flash is relatively straightforward. The JSFL code which must be generated to create a FLA file looks similar like that in listing 7.1. The resulting JSFL file then just has to be executed to create the final FLA files.

Figure 7.10 summarizes the resulting transformation steps: The first transformation maps an MML model to a Flash model. The second transformation is split into two parts. ActionScript class files are generated directly from the Flash model. For the document part, a JSFL file is generated. The JSFL file then has to be executed on a system with Flash installed.

A fully functional prototype for this transformation has been written in ATL.

Supported Flash Versions The metamodel presented here reflects the Flash versions 7 and 8.

Over the years MML support has been developed for several Flash versions. Table 7.1 provides an overview: The left hand side gibes an overview on the Flash versions, its version number, the release date and the latest ActionScript version supported (all Flash versions support also previous ActionScript versions for backward compatibility).

The table's right hand side gives an overview which versions MML currently supports. The metamodel and the transformations described in this thesis can be used for Flash 7 and Flash 8. While the metamodel is exactly the same for both versions, the transformation from the Flash Model to Flash Code provides a parameter to set the target version as there are slight changes in the JSFL file between

Version				MML Support		
Product Name	**No.**	**Released**	**Action-Script**	**Flash Metamodel**	**Transformation**	**Overall**
Macromedia Flash MX 2004	7	Sept. 9, 2003	AS2	as described in sec. 7.1	as described in app. C; provides parameter to set version number.	✓
Macromedia Flash Professional 8	8	Sept. 13, 2005	AS2			✓
Adobe Flash CS3	9	April 16, 2007	AS3	Implemented in [Meyer08]	Implemented in [Meyer08]	✓
Adobe Flash CS4	10	Oct. 15, 2008	AS3	[Meyer08] is expected to work but was not tested yet.		

Table 7.1: Flash versions and MML support

Flash 7 and Flash 8. Flash CS3 brought major changes, in particular introduction of ActionScript 3. [Meyer08], supervised by the author of this thesis, discusses and fully implements an update of the metamodel and the transformation for Flash CS 3 and ActionScript 3. As Flash CS4 does not provide such structural changes it is expected that the implementation by [Meyer08] is still sufficient but this has not been further tested yet.

7.3.1 Round-Trip-Engineering

An important practical aspect to be discussed in context of transformations is the so-called *Round-Trip Engineering* [Sendall and Küster04, Hettel et al.08]. Round-Trip Engineering means that a source model and code generated from the model are kept synchronous. If changes are made in the code which affect the model, then the model is synchronized and vise-versa. In context of transformation between models, Round-Trip Engineering is also called *Model Synchronization*.[4]

General Concepts It is important to understand that Round-Trip Engineering is *not* equivalent to the sum of Forward Engineering (i.e. code generation) and Reverse Engineering. *Reverse Engineering* [Tonella and Potrich04] usually means to (re-)construct a model from some given code, or more generally, to derive an abstract model from a more concrete model. However, in Round-Trip Engineering the original source model should only be changed to that extent as necessary for synchronization – it is not useful to construct via Reverse Engineering a completely new source model which reflects the code but has nothing in common with the model specified originally by the modeler.

The basic steps in Round-Trip Engineering are [Sendall and Küster04]:

1. Deciding whether a model under consideration has been changed,
2. Deciding whether the changes cause any inconsistencies with the other models, and
3. Once inconsistencies have been detected, updating the other models so that they become consistent again.

[4]In context of Model Driven Engineering code is treated as a specific kind of model as well – in the following the term “target model” refers to any kind of transformation result including code.

A change means that an element is either added, modified, or deleted. To synchronize models it is necessary to keep additional *trace information*, i.e. information to trace an element in the target model back to its causing element in the source model (see examples in [Sendall and Küster04]). In addition, it must be ensured that information added manually is not overwritten when synchronizing the models. These two requirements correspond to the postulated properties of model transformations from section 3.4.2: *traceability* and *incremental consistency*.

Incremental consistency on source code can be achieved by using a merge mechanisms as provided e.g. by *JMerge* [JMe] which is part of the EMF. Ideally, a tool does not only preserve the custom code but also updates the custom code when e.g. names of properties or operations have changed (using e.g. refactoring functionality like provided by the Eclipse Java Development Tools [JDT]).

For keeping trace information two common ways exist: Either augment the target model with the additional information, for instance comments in code containing specific tags, or store the trace information externally, for instance in an additional model.

Several tool support already exists like *SyncATL* [Xiong et al.07, Syn], a synchronization tool based on Eclipse supporting ATL. An important concept in this tool is to work on the modifications performed by the user on a model since the last transformation. Thus, it requires as input the 1) original source model, 2) the modified source model, 3) the modified target model, and 4) the transformation (the original target model can just be calculated from the original source model and the transformation). The tool requires that each model element is always annotated with a unique identifier which can not be changed by the user. The synchronization process first calculates the modifications between original model and its modified version for both, source and target model. This calculation uses the IDs to distinguish whether e.g. a model element has been added or renamed. As result of the calculation, an annotation is added to each element and property in the model to indicate the modification by one of this four tags: insert, delete, replace, or none. The modifications on the target model are propagated back to the source model and merged with the modifications on the source model. The modifications in the resulting source model are then propagated to the target model and finally merged with the modified target model. In that way, both models are synchronized while preserving all modification made by users on the source and the target model. For details, like merging rules, see [Xiong et al.07]. Basic concepts for differencing and merging models can also be found in various other work like [Abi-Antoun et al.08, Mehra et al.05].

Another important tool is the *Atlas Model Weaver* (*AMW*, [AMW]) based on Eclipse and ATL as well. Its purpose is more general: AMW establishes links between two models and stores them in a so-called *Weaving Model*. On that base various different use cases are available for download including support for traceability relationships between models and calculating the differences between two models.

Even with complete trace information there are still various cases where this information is not sufficient to synchronize the models. For instance, if an element is added to the target model, multiple alternative possibilities where to add corresponding elements in the source model may occur. Such cases can be solved for instance by rules for default behavior or by additional user input. More information on advanced issues can be found e.g. at [Oldevik et al.08].

Round-Trip Engineering for MML Basically, the general principles of Round-Trip Engineering can also be applied to MML. In particular, the ActionScript classes can be treated like any other kind of source code. But two specific problems must be considered: First, most existing solutions require annotations on the model and code. Thus, it must be ensured that it is possible to annotate elements in Flash Documents with custom data. Second, parts of the MML models reside on a quite high level of abstraction compared to the generated Flash implementation. Thus, it must be discussed for which kind of modifications it is useful to propagate them back to the MML model. They are discussed in the following.

ActionScript classes can be handled like conventional source code and be annotated with comments. Annotations are important for Round-Trip Engineering, for instance to store trace information or for unique IDs to calculate modifications between models like in SyncATL (see above).

Fortunately, the Flash Java Script API enables to store additional information to the most important elements used in our structure: Data of any kind of primitive types can be added, read, and removed to Flash Documents (method document.addDataToDocument()), to all Library Items (item.addData()), and some types of Elements on the Stage including MovieClip instances and Component instances (element.setPersistentData(), see figure 7.6 for metaclass Element). These elements cover already the major part of the generated structure as elements on the Stage are encapsulated into MovieClips (see “MML to Flash Model” on page 158).

Annotating the Flash elements with unique identifiers can help to ensure incremental consistency for Flash Documents. The basic idea is to calculate modifications on the Flash model. The JSFL file then propagates only these modifications to the Flash Documents. For instance, when a MovieClip is renamed in the model, the JSFL file searches for the MovieClip by its identifier and modifies its property name while leaving all other properties and content unchanged.

The second problem is the partially high level of abstraction. Thus, for some parts of the MML models it is not trivial to synchronize a MML models when changes appeared in the generated code. A general requirement is that modifications in the Flash application must comply to the generated structure for Flash applications. If a developer adds an element which does not fit to the intended structure at all, it is almost impossible to interpret its meaning in terms of the MML model.

The main MML elements can be handled as follows:

- MML Domain Classes are mapped 1:1 into ActionScript classes in the folder model. All changes can be directly propagated to the MML Structure diagram just like in mappings from UML class diagrams to Java.
- An MML Scene is mapped to a large number of elements and code. Thus, even modifications like renaming can become tedious in the code. It is often better to add new Scenes only in the model and re-generate the code. Modifying conventional class properties or class operations of the Scene’s ActionScript class is unproblematic and can be directly propagated to the MML model.
- Each AIO in MML is mapped to a MovieClip in the Scene and a listener class. Basically, it should be possible to add, modify, or delete these MovieClips directly in Flash. The changes can be propagated to the model. The corresponding listener classes could then just be updated when re-generating code. However, it must be considered that adornments on the user interface should not be interpreted as AIOs in MML. A useful solution is that only MovieClips instances on the Stage having an instance name are interpreted as AIOs (in Flash, instances have no instance name by default) while all other elements are interpreted as adornments.
- MML Media Components are directly mapped into the folder media. Adding, deleting or renaming a Media Component directly in the folder media is possible and can be propagated back to the model. However, the developer has to consider Media Instances which refer to this Media Component. Advanced modifications, concerning inner structure or different variations, are not propagated back to the model.

Of course, when the developer fills out generated placeholders or operation bodies, this has not to be propagated back to the model as specifying their content is not part of MML models.

7.3.2 A Tool for Extracting Flash Models from Flash Applications

A first important contribution to Round-Trip Engineering as well as Reverse Engineering of Flash applications was implemented in [Mbakop07] supervised by the author of this thesis. The resulting tool analyzes a Flash application and creates a corresponding Flash model. The Flash model complies to the Flash metamodel presented in section 7.1. Like the Flash metamodel, the tool is independent from MML and works for any kind of Flash application.

The tool considers both Flash Docments and ActionScript code. As first step, the tool analyzes the Flash Documents. Therefore, a JSFL script is executed which browses all elements in the Flash Documents and extracts all found information to an XML file. Moreover, it collects the ActionScript code which belongs to the application. In a second step, a parser for ActionScript – implemented with JavaCC [Jav] and JJTree [JJT] – creates an Abstract Syntax Tree for the ActionScript code. The third step integrates the Abstract Syntax Tree and the information on the Flash Documents to the resulting Flash Model.

The tool can be useful in various contexts:

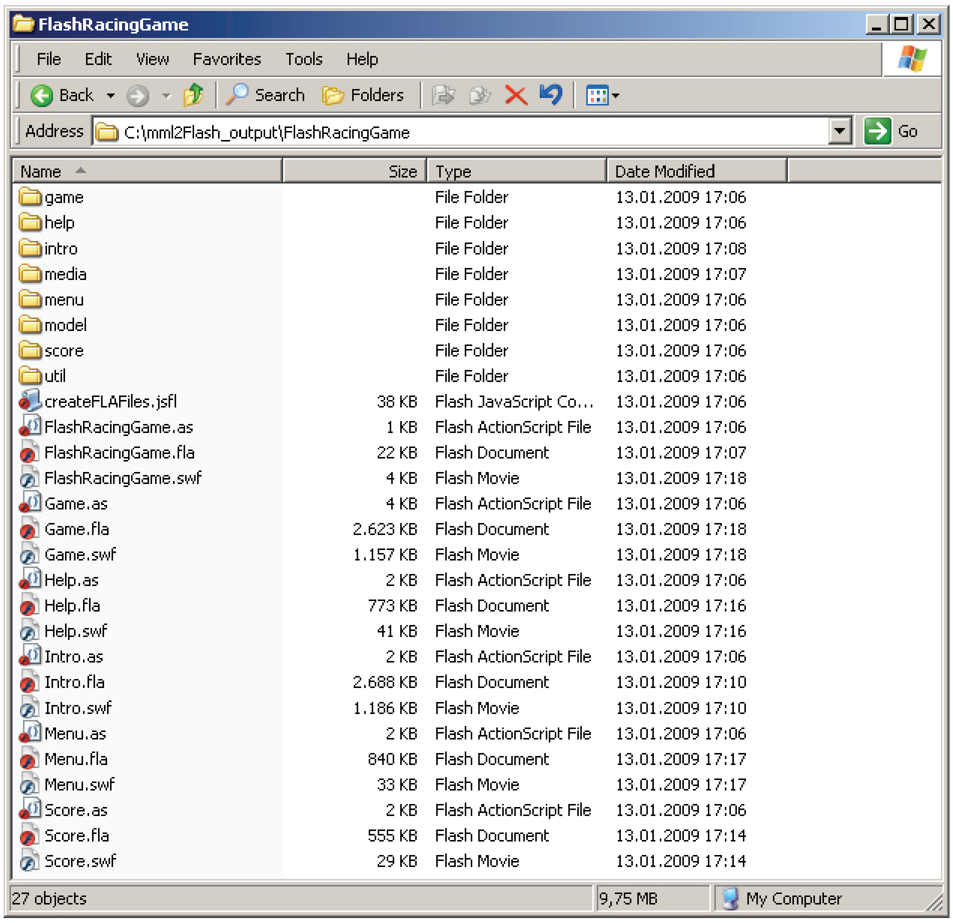
- Round-Trip Engineering: The tool gives the possibility to reuse existing Round-Trip Engineering tools operating on models. The precondition is that the transformation back to the Flash code preserves incremental consistency.
- Reverse Engineering: The resulting Flash models can be used as starting point for a transformation from Flash models to MML models to Reverse Engineer Flash applications. A very first prototype for such a transformation has been implemented in [Mbakop07].
- Migration and Refactoring: In particular, it is possible to refactor the Flash Model and transform it back to a Flash application. Thereby, Flash Documents can automatically be updated to another Flash version. For instance, executing the transformation from [Meyer08] automatically results in Flash Documents for Flash version 9. (Converting ActionScript to another ActionScript version as well would require to write an additional transformation for this purpose as the Flash model treats operation bodies and embedded scrips just as text strings – so the currently existing transformations can only be used to create Flash 9 Documents with ActionScript2 code.) This use case requires incremental consistency for the transformations back to Flash.

7.4 Working with the Code Skeletons in the Authoring Tool

This section illustrates by screenshots how to work with the generated code skeletons in the Flash authoring tool. It shows the application generated from the MML models for the example Racing Game application from section 5 and 6.

Figure 7.11 shows the folder with the generated files in the file explorer. For each Scene there is a FLA file and a folder. The folder model contains the domain classes. The folder media contains the Media Components. The folder util contains some library classes.

By default there are already compiled versions of the Flash Documents (SWF files) which can be directly executed by a double-click on the main file FlashRacingGame.swf. Then, – as specified in the Scene model (fig. 6.9) – the application starts with the generated skeleton for the Scene Intro. The generated Exit Operations in the Scene's ActionScript class contain the code for the transitions between the Scenes. However, it depends on the models whether they can already be triggered by the user. In the Racing Game example, the Presentation Model defines some Action Components which trigger the Exit Operations (e.g. the AIOs quit and help in fig. 6.14). As Action Components are



C:\mml2Flash_output\FlashRacingGame

Name	Size	Type	Date Modified
game		File Folder	13.01.2009 17:06
help		File Folder	13.01.2009 17:06
intro		File Folder	13.01.2009 17:08
media		File Folder	13.01.2009 17:07
menu		File Folder	13.01.2009 17:06
model		File Folder	13.01.2009 17:06
score		File Folder	13.01.2009 17:06
util		File Folder	13.01.2009 17:06
createFLAFiles.jsfl	38 KB	Flash JavaScript Co…	13.01.2009 17:06
FlashRacingGame.as	1 KB	Flash ActionScript File	13.01.2009 17:06
FlashRacingGame.fla	22 KB	Flash Document	13.01.2009 17:07
FlashRacingGame.swf	4 KB	Flash Movie	13.01.2009 17:18
Game.as	4 KB	Flash ActionScript File	13.01.2009 17:06
Game.fla	2.623 KB	Flash Document	13.01.2009 17:18
Game.swf	1.157 KB	Flash Movie	13.01.2009 17:18
Help.as	2 KB	Flash ActionScript File	13.01.2009 17:06
Help.fla	773 KB	Flash Document	13.01.2009 17:16
Help.swf	41 KB	Flash Movie	13.01.2009 17:16
Intro.as	2 KB	Flash ActionScript File	13.01.2009 17:06
Intro.fla	2.688 KB	Flash Document	13.01.2009 17:10
Intro.swf	1.186 KB	Flash Movie	13.01.2009 17:10
Menu.as	2 KB	Flash ActionScript File	13.01.2009 17:06
Menu.fla	840 KB	Flash Document	13.01.2009 17:17
Menu.swf	33 KB	Flash Movie	13.01.2009 17:17
Score.as	2 KB	Flash ActionScript File	13.01.2009 17:06
Score.fla	555 KB	Flash Document	13.01.2009 17:14
Score.swf	29 KB	Flash Movie	13.01.2009 17:14

27 objects | 9,75 MB | My Computer

Figure 7.11: The generated files in the file explorer.

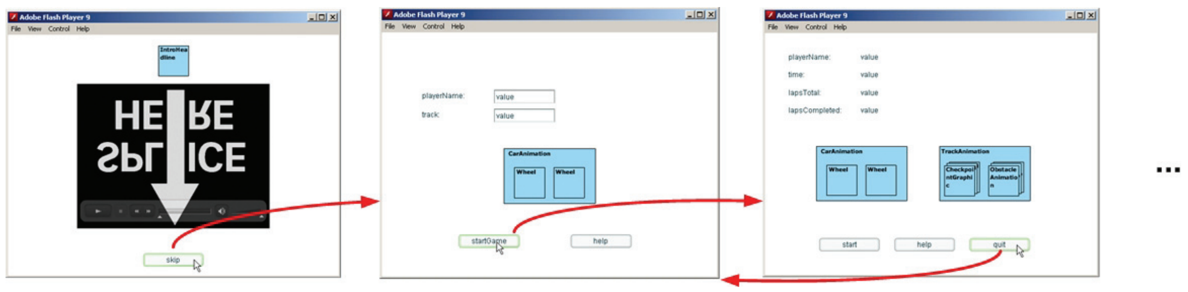


Figure 7.12: The generated application skeleton can be directly executed. The user can trigger transitions between the Scenes as far as specified in the MML model.

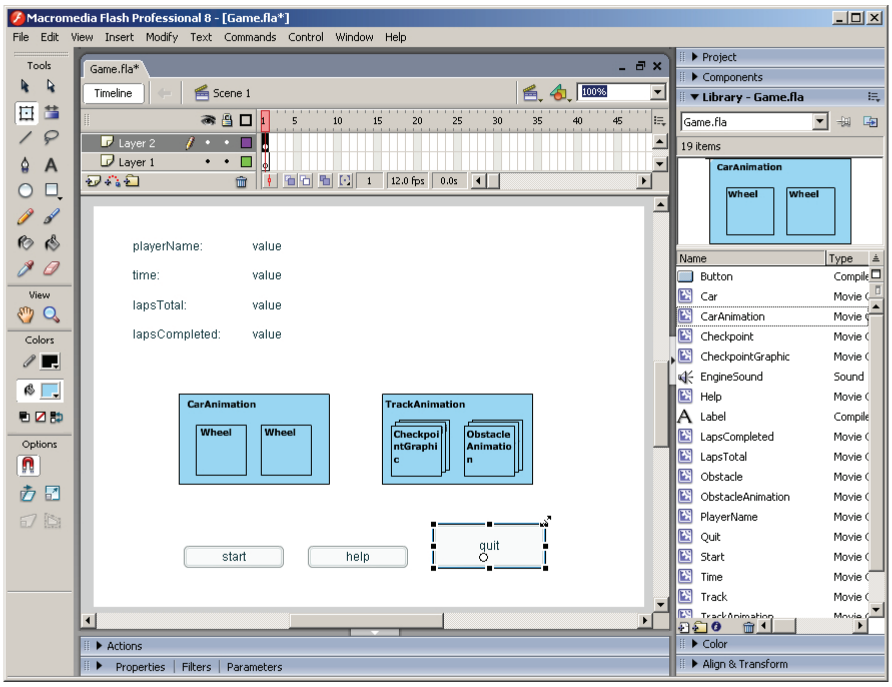


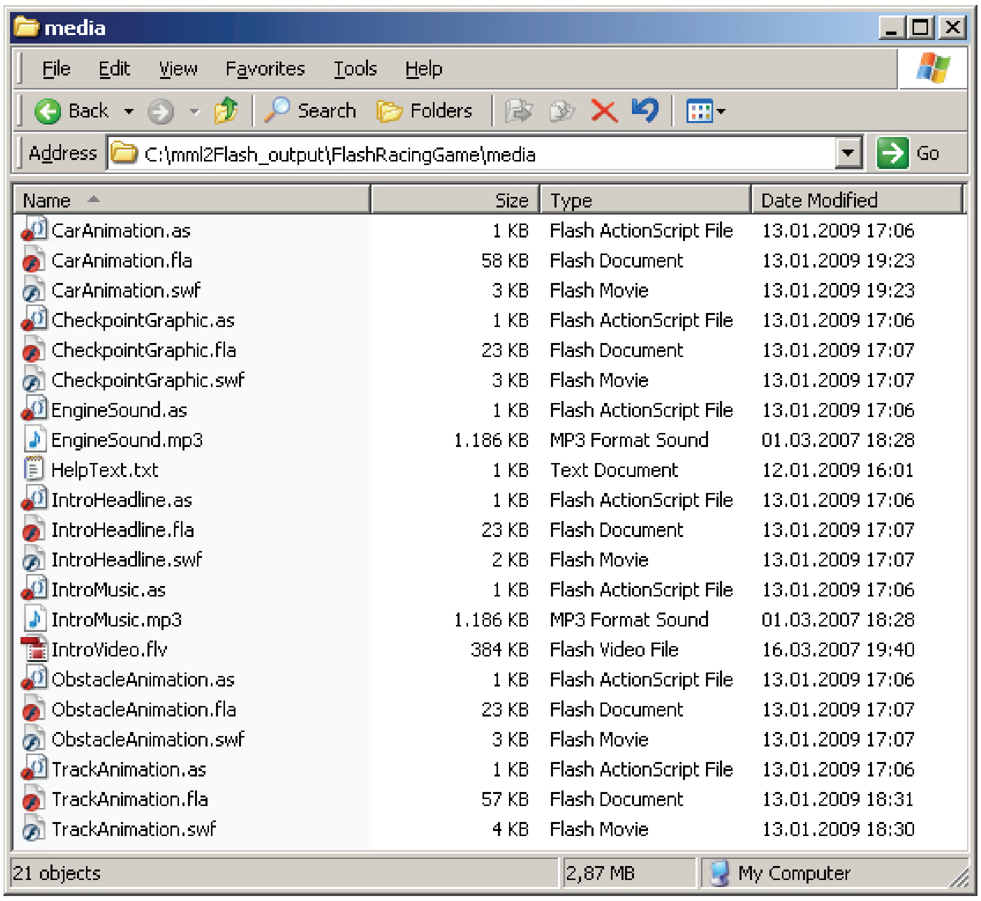
Figure 7.13: The generated FLA file for the Scene Game.

mapped to buttons, the user can use this buttons to trigger the transitions and navigate through the Scenes.

Figure 7.12 shows the generated SWF files when they are executed by the user. It shows from left to right how the user navigates through some generated Scenes. According to the Scene model, the application start with the Into Scene (left screenshot) which contains a placeholder for the IntroHeadline, a dummy video for IntroVideo, and a button skip. As specified in the Presentation Model, the user can navigate to the Menu Scene (middle screenshot) by clicking the button. The Menu contains a button startGame which leads the user to the Scene Game (right screenshot). In the Game Scene the user can use the button quit to navigate back to the Menu. The application, as shown here, is the direct result from the transformation – it has not been modified by the developer yet[5]. The following sections show how the developers edit and complete the generated skeletons in the Flash authoring tool.

The generated FLA files can be directly opened and processed in the authoring tool. Figure 7.13 shows the FLA file for the Scene Game generated according to the MML model in figure 6.17. The graphics and animations have been mapped to placeholder MovieClips. Their Inner Structure is represented by contained Movie Clips. AIOs not realized by Media Components have been mapped to widgets components. All generated elements can be very easily adapted and modified using the large spectrum of tools available in Flash. For instance, they can be arranged by drag and drop and be resized, rotated, skewed, reshaped, etc. using the transformation tool (like the button quit in fig. 7.13). The generated instance names are visible in the Property Window while the generated connection between Movie Clips and ActionScript classes are visible via the context menu in the library (not visible in the screenshot).

[5]except the spatial layout of the graphical elements as the current implementation does not include an algorithm for the initial layout yet


media
File Edit View Favorites Tools Help
Back Search Folders
Address C:\mml2Flash_output\FlashRacingGame\media
Go

Name	Size	Type	Date Modified
CarAnimation.as	1 KB	Flash ActionScript File	13.01.2009 17:06
CarAnimation.fla	58 KB	Flash Document	13.01.2009 19:23
CarAnimation.swf	3 KB	Flash Movie	13.01.2009 19:23
CheckpointGraphic.as	1 KB	Flash ActionScript File	13.01.2009 17:06
CheckpointGraphic.fla	23 KB	Flash Document	13.01.2009 17:07
CheckpointGraphic.swf	3 KB	Flash Movie	13.01.2009 17:07
EngineSound.as	1 KB	Flash ActionScript File	13.01.2009 17:06
EngineSound.mp3	1.186 KB	MP3 Format Sound	01.03.2007 18:28
HelpText.txt	1 KB	Text Document	12.01.2009 16:01
IntroHeadline.as	1 KB	Flash ActionScript File	13.01.2009 17:06
IntroHeadline.fla	23 KB	Flash Document	13.01.2009 17:07
IntroHeadline.swf	2 KB	Flash Movie	13.01.2009 17:07
IntroMusic.as	1 KB	Flash ActionScript File	13.01.2009 17:06
IntroMusic.mp3	1.186 KB	MP3 Format Sound	01.03.2007 18:28
IntroVideo.flv	384 KB	Flash Video File	16.03.2007 19:40
ObstacleAnimation.as	1 KB	Flash ActionScript File	13.01.2009 17:06
ObstacleAnimation.fla	23 KB	Flash Document	13.01.2009 17:07
ObstacleAnimation.swf	3 KB	Flash Movie	13.01.2009 17:07
TrackAnimation.as	1 KB	Flash ActionScript File	13.01.2009 17:06
TrackAnimation.fla	57 KB	Flash Document	13.01.2009 18:31
TrackAnimation.swf	4 KB	Flash Movie	13.01.2009 18:30

21 objects | 2,87 MB | My Computer

Figure 7.14: The generated folder media.

Figure 7.14 shows the content of the folder media in the file explorer. The FLA files generated for graphics and animations contain only placeholders to be filled out by the media designers. Figure 7.15 shows how easy to fill out for instance the generated placeholders in the FLA file for CarAnimation. The media designer can use all tools of the Flash authoring tool like normally. After the modified CarAnimation is saved and compiled it is automatically available in all Scenes which refer to it (like Game, see fig. 7.16a). For instance, the CarAnimation is used not only in the Scene Game but also in Menu where the user can select between different cars. Several properties can be changed individually for each instance including size, rotation (fig. 7.16b), and color properties like brightness, tint, alpha, etc.

Other media types which are not directly created within Flash, like images, audio, and video, are represented by files containing dummy content and are usually replaced directly in the file explorer. Text files can be edited directly in a text editor. For instance, in figure 7.17a the dummy file for video IntroVideo.flv is replaced in the file explorer. As result, all instances of IntroVideo in all Scenes are updated in the application (like the Scene Intro in fig. 7.17b).

Besides filling out the placeholders in the authoring tool and replacing dummy files, the software developers have to complete the ActionScript code. This will usually take some time as MML is not a prototyping tool. The developers have to perform the following tasks on the ActionScript skeletons:

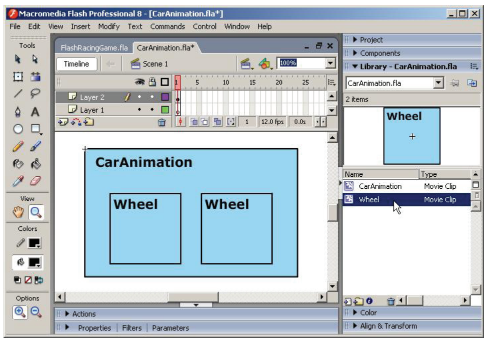
- Initialize Domain Objects in the Scene class.

- Specify parameters in Exit Operations for Entry Operation calls.
- Fill out Entry Operations in the Scene class.
- Fill out the bodies of Domain Class operations.
- Fill out the bodies of event handling operations or add own event listeners.

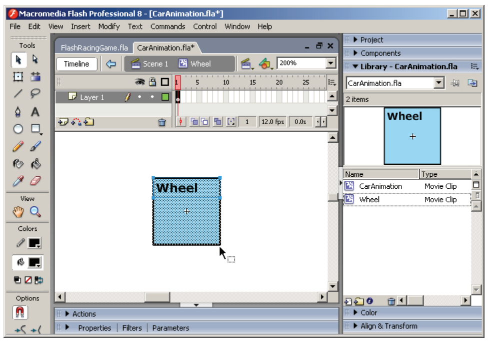
For instance, in the Racing Game example the developers would place the code to move the car into the operation move of the class Car in the folder model. As the design pattern Observer is generated for the connection to the user interface, the developers also have to fill out the operation update in the class Car in the folder Game[6].

All other basic relationships are already generated from the models: the class Car in the folder Game is already assigned to MovieClip Car, the MovieClip is already instantiated on the Stage and has an instance name, and the corresponding instance is accessible in the class Scene and already registered as Observer for the Domain Class Car. Thus, it is already possible to directly compile, execute, and test any added code. A possible result could look like in figure 7.18.

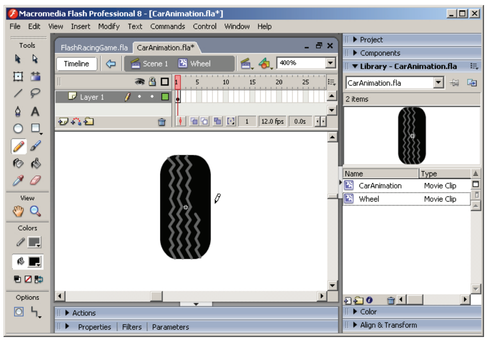
[6]The Domain Class and the AIO both have the same name in this example


Macromedia Flash Professional 8 - [CarAnimation.fla*]
CarAnimation
Wheel
Wheel
Library - CarAnimation.fla
Wheel

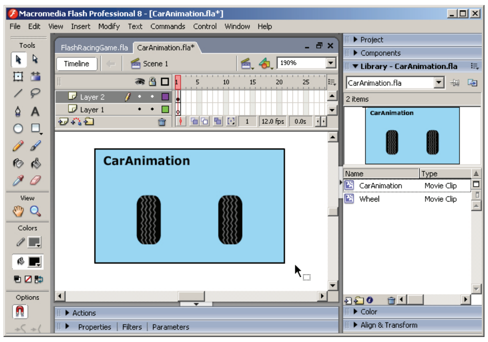
(a) Select wheel in the library


Macromedia Flash Professional 8 - [CarAnimation.fla*]
Wheel
Library - CarAnimation.fla
Wheel

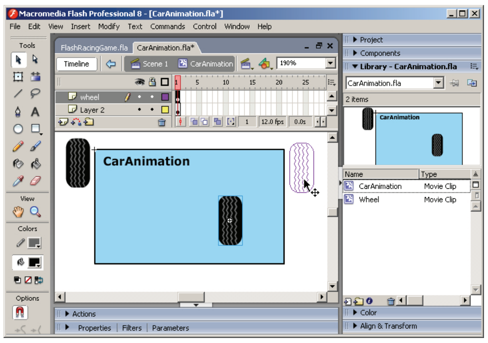
(b) Select the placeholder shape of wheel


Macromedia Flash Professional 8 - [CarAnimation.fla*]
Library - CarAnimation.fla

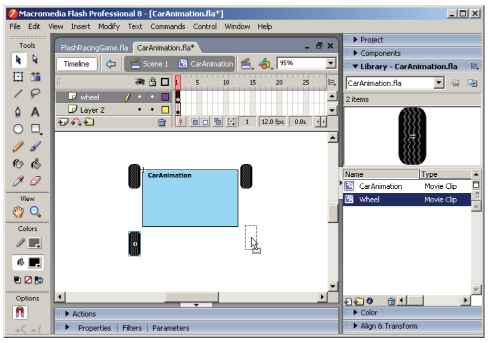
(c) Replace placeholder with custom drawing


Macromedia Flash Professional 8 - [CarAnimation.fla*]
CarAnimation
Library - CarAnimation.fla

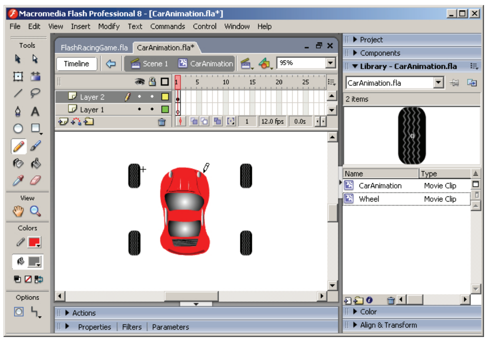
(d) Back to CarAnimation


Macromedia Flash Professional 8 - [CarAnimation.fla*]
CarAnimation
Library - CarAnimation.fla

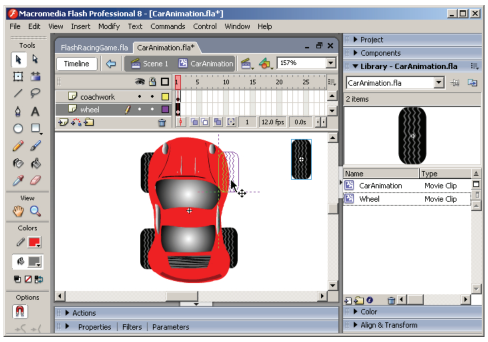
(e) Arrange the wheels


Macromedia Flash Professional 8 - [CarAnimation.fla*]
CarAnimation
Library - CarAnimation.fla

(f) Add two more wheel instances

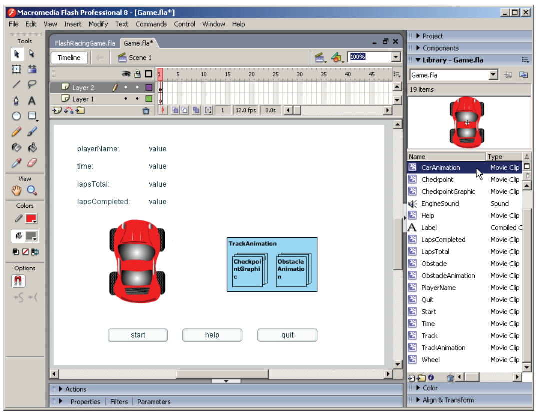

Macromedia Flash Professional 8 - [CarAnimation.fla*]
Library - CarAnimation.fla

(g) Replace car's placeholder with custom drawing

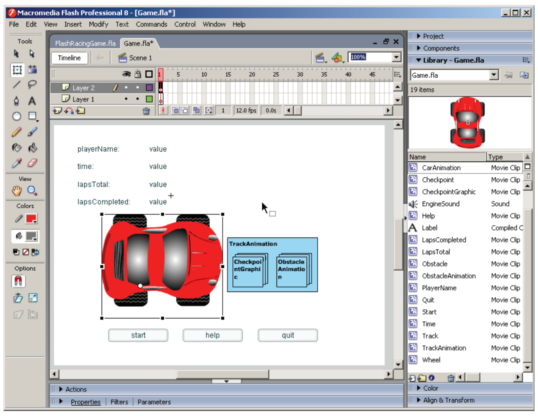

Macromedia Flash Professional 8 - [CarAnimation.fla*]
Library - CarAnimation.fla

(h) Finalize CarAnimation

Figure 7.15: Replacing the generated placeholder for CarAnimation.

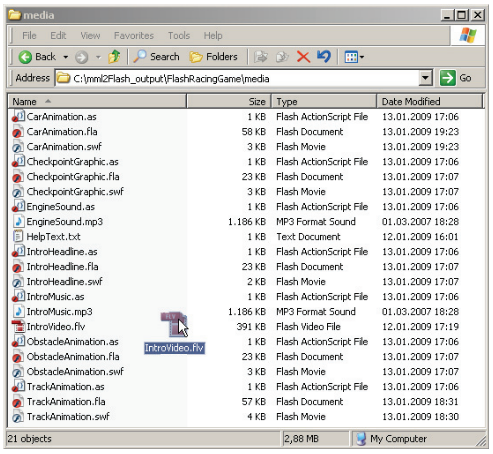


(a) Updated CarAnimation



(b) Modifying rotation and size of a single instance

Figure 7.16: CarAnimation in Scene Game.

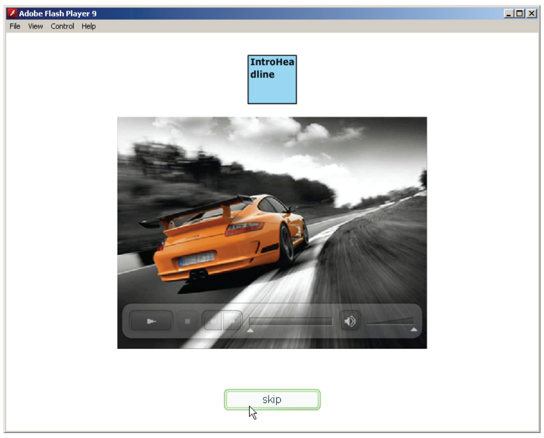

media

Address C:\mml2Flash_output\FlashRacingGame\media

Name	Size	Type	Date Modified
CarAnimation.as	1 KB	Flash ActionScript File	13.01.2009 17:06
CarAnimation.fla	58 KB	Flash Document	13.01.2009 19:23
CarAnimation.swf	3 KB	Flash Movie	13.01.2009 19:23
CheckpointGraphic.as	1 KB	Flash ActionScript File	13.01.2009 17:06
CheckpointGraphic.fla	23 KB	Flash Document	13.01.2009 17:07
CheckpointGraphic.swf	3 KB	Flash Movie	13.01.2009 17:07
EngineSound.as	1 KB	Flash ActionScript File	13.01.2009 17:06
EngineSound.mp3	1.186 KB	MP3 Format Sound	01.03.2007 18:28
HelpText.txt	1 KB	Text Document	12.01.2009 16:01
IntroHeadline.as	1 KB	Flash ActionScript File	13.01.2009 17:06
IntroHeadline.fla	23 KB	Flash Document	13.01.2009 17:07
IntroHeadline.swf	2 KB	Flash Movie	13.01.2009 17:07
IntroMusic.as	1 KB	Flash ActionScript File	13.01.2009 17:06
IntroMusic.mp3	1.186 KB	MP3 Format Sound	01.03.2007 18:28
IntroVideo.flv	391 KB	Flash Video File	12.01.2009 17:19
ObstacleAnimation.as	1 KB	Flash ActionScript File	13.01.2009 17:06
ObstacleAnimation.fla	23 KB	Flash Document	13.01.2009 17:07
ObstacleAnimation.swf	3 KB	Flash Movie	13.01.2009 17:07
TrackAnimation.as	1 KB	Flash ActionScript File	13.01.2009 17:06
TrackAnimation.fla	57 KB	Flash Document	13.01.2009 18:31
TrackAnimation.swf	4 KB	Flash Movie	13.01.2009 18:30

21 objects | 2,88 MB | My Computer

(a) Replacing IntroVideo.flv in folder media



(b) All instances of IntroVideo are automatically replaced.

Figure 7.17: Media files need just to be replaced in the file explorer.

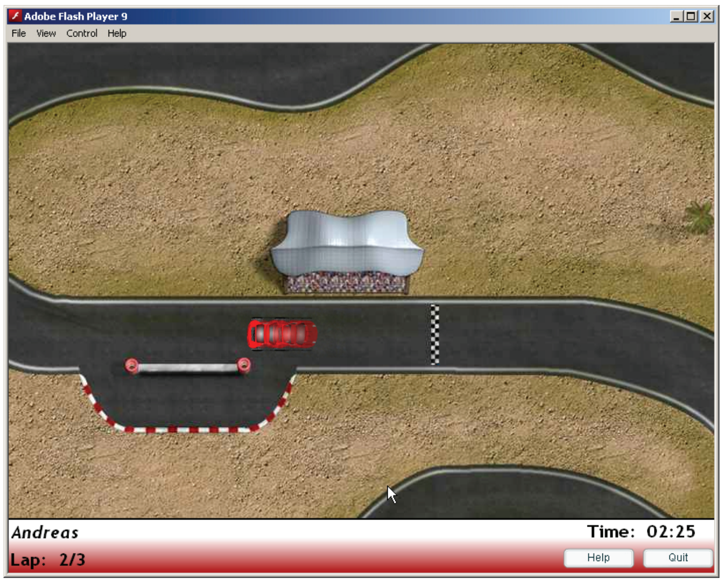


Figure 7.18: The final result.

Chapter 8

Validation

The basic problem addressed in this thesis is the lack of systematic development methods for the interactive multimedia applications. As discussed in section 3.1, this problem statement is supported by the existing research literature as well as by existing industry studies. A promising contribution to solve this problem is to apply concepts from model-driven development to the multimedia application domain (sec. 3.2). Thus, a modeling language and model-transformations were proposed. This section aims to validate the presented approach.

Unfortunately, development methods and development processes, such as MML, are difficult to evaluate. This is a specific problem of Software Engineering research: For instance, Human-Computer Interaction research mostly target end users. Often it is possible with limited effort to acquire people from the addressed target group and to perform user studies with them. For Software Engineering research which addresses large real-world development projects, no such straightforward way exists.

Thus, the next paragraph provides a short analysis to find possible validation methods for MML. The subsequent three section then present the validation based on the identified methods.

Validation Techniques in Software Engineering In [Shaw01], several common validation techniques in Software Engineering research are listed:

- **Persuasion:** Argue for the solution and explain it e.g. by a running example.
- **Implementation:** Demonstration by a prototype.
- **Evaluation:** Evaluation in comparison to other approaches, against given criteria or based on empirical data.
- **Analysis:** Derivation from facts by a formal proof or an empirical predictive model.
- **Experience:** Evaluation based on experience or on observations made during application in industrial practice.

The first two techniques are already applied within this thesis: The *Persuasion* technique applies to MML in chapters 5, 6, and 7 which discuss the reasons for modeling elements and show the basic feasibility by means of the Racing Game example.

The *Implementation* of the transformations (sec. 7) shows that the modeling language provides an adequate level of abstraction to enable code generation. It also shows that it is possible to generate code directly for an authoring tool. Moreover, there are additional implementations of transformations, to other target platforms than Flash, which demonstrate the platform independence of MML. They are presented in section 8.1.

To achieve stronger validation, the other three techniques need to be applied. *Analysis* is not considered in this thesis as the problem space (multimedia applications, integration of creative design, etc.) as well as the desired properties of the solution (expressiveness, usability, etc.) are hard to formalize.

Evaluation against empirical data again is difficult for research like in this thesis. There are rarely some opportunities to apply a new research approach directly in a real-world projects in a company. Alone the tool support is certainly not sufficient for a really professional project.

Validation based on new *Experience* in industry is not possible for the same reasons. Of course, already existing experience has already been considered carefully by the discussion of existing studies provided in section 3.1.

An alternative to gain empirical data empirical data (for *Evaluation against empirical data*) is to perform controlled experiments in the research lab. However – in contrast to other areas, like Interaction Design – development methods are hard to test due to many factors like the required learning effort and the large number of variables which includes experience, skills, project type, project size, etc. [Olsen07, Walker et al.03, Kitchenham et al.02]. Thus, it is usually not possible to create with reasonable effort an experiment which allows to proof the effect of a development approach in industrial practice.

Finally, *Evaluation against given criteria* is a feasible and important validation technique for development approaches and modeling languages in particular. Also *Evaluation in comparison to other approaches* could be examined at least on a theoretical level. Both methods are applied for MML in section 8.3.

Nevertheless, all feasible validation techniques identified so far are still on a theoretical level. This kind of validation is often called *internal validation* as it is usually conducted by the developers of the research approach themselves. The danger is that their point of view is too optimistic. They also certainly have a very different access to their own approach than other people would have. Hence, it is strongly desirable to perform an *external validation* as far as possible – even if it is not possible to definitely proof the effect of MML in industrial practice. To meet this demand, different kinds of student projects have been conducted which are described in section 8.2.

8.1 Platform Independence: Transformations to other Platforms

Flash is not the only target platform for MML. From the beginning, MML was designed as a platform independent language based on existing platform independent concepts. Thus, several transformations for other target platforms have been implemented during the development of MML.

For the choice of a target platform it is useful, of course, to select a platform with sufficient multimedia support. Moreover, an integration with authoring tools – like in the Flash example in section 7 – is only possible for platforms supported by authoring tools. Nevertheless, as MML models are platform independent they can be transformed into any target language – independent from its tool support.

Beside the main example Flash (sec. 7), transformations for three other platforms have been developed in context of this thesis:

1. *Java* as example for a conventional programming language in conjunction with multimedia frameworks,

2. SVG/JavaScript as example for a declarative language, and

3. Flash Lite as example for a platform for mobile devices.

Together with Flash, the first two examples cover all three categories of implementation platforms identified in section 2.3: frameworks, declarative languages, and authoring tools. The third example, Flash Lite, demonstrates that platform independence also adheres to the target devices.

While some of these transformations are (prototypically) implemented to a large extent, some are only realized as a proof of concept. The following sections briefly describe them.

8.1.1 Code Generation for Java

The transformation into Java code is the result of two project theses [Wu06a, Kaczkowski07] supervised by the author of this thesis. The frameworks *Piccolo* is used to support 2D graphics, animations, images, and text. For other media types, like audio and video, the *Java Media Framework* is used (both frameworks are introduced in section 2.3.1. The focus in the project thesis was on the support for 2D graphics and animations with Piccolo.

The structure of the generated application is analogous to that for Flash from section 7.2: The application part implemented by ActionScript classes in Flash applications is implemented by (plain) Java classes. The part implemented by Flash documents is implemented using the Piccolo framework. The overall application structure complies to the Model-View-Controller paradigm. Compared to Flash/ActionScript, it is much easier to gain a clear object-oriented structure for the overall application, as Java, Piccolo, and JMF already provide an object-oriented structure.

[Kaczkowski07] proposes to map the MML models into the following structure:

- The overall application is structured into packages model, view, controller, and media. In addition, [Kaczkowski07] provides a package mml containing library classes like a class Scene as superclass for all Scenes and classes for the different media types.
- An application is mapped to a main class which creates the application's main window on the user interface.
- Each Scene (sec. 6.3) is mapped to a class which initiates instances used in the Scene and connects model and view classes by the *Observer* pattern [Gamma et al.95]. It contains Entry and Exit Operations which pass the application's main window as parameter so that a Scene can set the actual user interface content.
- Domain Classes (sec. 6.2) are mapped to Java classes in the package model.
- Media Components (sec. 6.2) are mapped to classes in the package media. The package mml provides a library class for each media type with standard functionality. They are specialized by the classes in media.
- Each Presentation Unit (sec. 6.4) is mapped to a class in the package view. It inherits from Piccolo classes and provides different operations to easily add Piccolo Nodes as well as Java Swing elements as children[1]. Media Instances are mapped to instances from the package media and added as children. AIOs not realized by Media Instances are mapped to Java Swing widgets and added as children.
- For each Scene there is a Controller class in the package controller which contains event handling operations. They are assigned to the user interface elements in the Scene's view class.

The transformation was implemented in [Kaczkowski07] using ATL. [Kaczkowski07] also provides a metamodel for Java Piccolo applications. It consists of a part for Java classes and a part representing the Piccolo framework (which of course consists of Java classes as well but is represented on a higher level of abstraction).

[1]As described in section2.3.1, Piccolo code is structured in terms of a Scene Graph

The Java classes generated by the transformation can directly be compiled and executed. The application then shows a window with the (skeleton) user interface for the first Scene.

8.1.2 Code Generation for SVG/JavaScript

An approach developed in an early stage of MML was code generation for SVG and JavaScript in [Leichtenstern04]. SVG refers to the *Scalable Vector Graphics* [W3C03], an XML-based description language for 2D graphics and animations. It is a World Wide Web Consortium (W3C) [WWWa] standard and can be displayed in most web browsers and by external viewers. Most existing vector graphics software supports the SVG format and there are also some editors specifically for SVG. SVG has also become a graphic standard for mobile devices and provides different profiles e.g. for smartphones and PDAs.

It is possible to combine SVG with JavaScript to create interactive and dynamic documents. For this purpose the nodes in a SVG document can be accessed via a *Document Object Model* similar to that in HTML. It was chosen as example platform for MML (instead of e.g. SMIL) as it seemed a promising platform at that time. In conjunction with JavaScript, it basically provides similar possibilities as Flash. At that time, SVG was strongly supported by Adobe and some people expected that it might become a competitor to Flash. However, this has not come true so far, in particular, as Adobe acquired Macromedia with its products Flash and Director in 2005. Currently, SVG is sometimes used for still graphics but is far away from a usage for applications, like Flash, as neither libraries nor tool support exist for that purpose.

The work from [Leichtenstern04] falls in an early stage within the development of MML. It uses the MML version described in [Hußmann and Pleuß04]. For code generation it uses the Eclipse-based *Java Emitter Templates* (*JET*, [JET]) which are part of the Eclipse Modeling Framework (see sec. 3.4.3). Again, the generated structure adheres to the Model-View-Controller pattern. Domain classes, event handling, and the behavior of Scenes are implemented in JavaScript while each Presentation Unit is mapped to a SVG document. The SVG document contains JavaScript code to connect user interface elements with JavaScript objects. The detailed mapping and a working prototype can be found in [Leichtenstern04].

8.1.3 Code Generation for FlashLite

In the context of his teaching assistance in the course “Multimedia-Programming” (see sec. 8.2.1), Max Maurer examined a possible transformation from MML into code skeletons for Flash Lite [Flab]. Flash Lite is a lightweight version of Flash for mobile phones and other devices with limited computing power (see [Adobeb]).

With growing version numbers, Flash Lite supports more and more similar functionality like conventional Flash. The main difference lies in the limited computing power of target devices which requires a different programming style. For instance, a large number of MovieClip instances or ActionScript objects causes a significant loss of performance. In simple tests on a smartphone device (xde terra) from 2007, it was not possible to run about 20 instances of a simple MovieClip associated with an ActionScript class with acceptable performance. Thus, the proposed framework for Flash applications from section 7.2 is inappropriate for Flash Lite, as it focuses on modularity and thus uses a very large number of instances.

Hence, Maurer provides a proposal for simplifying the application structure:

- The whole application is mapped to a single Flash Document (FLA file). It contains on its Stage a MovieClip which acts as container for the Scenes.

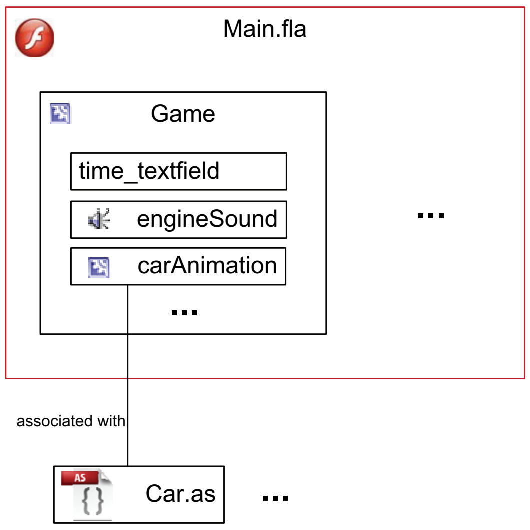


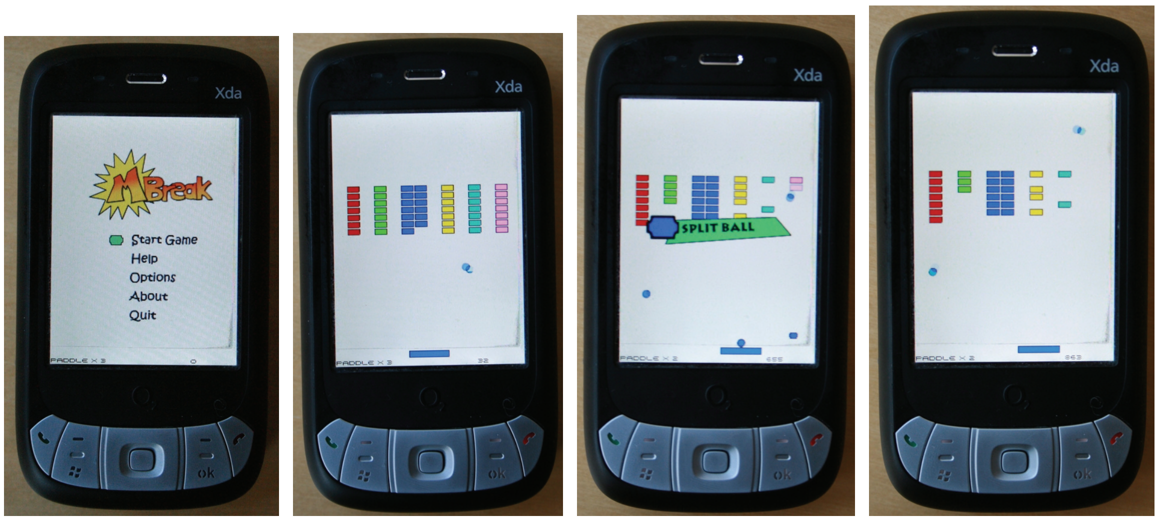
Figure 8.1: Simple target structure for Flash Lite applications and small Flash applications.

- Each Scene is mapped to a MovieClip in the main Flash Document's library. They are dynamically attached to the container MovieClip on the Stage. There are no ActionScript classes for the Scenes.
- The user interface elements are directly placed into the Scenes without an encapsulating MovieClip. They are not associated with ActionScript classes. Instead, the event handling code is placed directly on the main Timeline in the main Flash Document.
- As there is only a single Flash Document, the media objects need just to be placed in its library. They can then be instantiated in the MovieClips for the Scenes.
- The Domain Classes are mapped to ActionScript classes.

Figure 8.1 shows an example for the resulting simple structure. The figure shows the Racing Game example so that it can be easily compared to the Flash framework in figure 7.9. There is only a single Flash Document for the whole application (Main.fla). Each Scene is represented by a MovieClip (e.g. Game) which directly contains all Media Instances (e.g. carAnimation) and user interface elements (e.g. time_textfield). As all Media Components are located in the library of the main Flash Document, it is still possible to instantiate a Media Component multiple times within different Scenes. Only the Domain Classes are represented by external ActionScript classes (e.g. Car). Any other code (like event listeners) is embedded directly in the Flash Document.

This structure can also be used for conventional (desktop computer) Flash applications if a simple straightforward structure is desired. This is useful e.g. for applications with limited application logic like the CMD Tutorial later in section 8.2.2.

The transformation has not been implemented yet. However, Maurer demonstrates the feasibility by a manual mapping for the Blockout Game example from [Pleuß05a]. Figure 8.2 shows some screenshots from the resulting application. Despite of its high dynamics and fast animations, it runs without any performance problems on the test device.



(a) (b) (c) (d)

Figure 8.2: Blockout example on the Flash Lite platform.

8.2 External Validation

As discussed above (“Validation Techniques in Software Engineering”), it is desired to perform an external validation of MML, even if it can not provide a proof for the effect of MML in practice. However, an external validation provides several more benefits:

- Experience on the feasibility of MML.
- Experience on the usability of MML.
- Experience on, how in general people develop multimedia applications with and without MML.
- Finding problems with MML not expected by its developers, in order to improve the approach.
- Acquiring qualitative indicators whether MML is beneficial in practice.

Therefore, MML has been applied in several student projects in teaching context at the Media Informatics group at University of Munich. In each of the projects, students had to develop a multimedia application using MML. Thereby, the projects goals were independent from MML, i.e. the motivation for the students was not to apply MML but to successfully develop an application. The participants neither had previous knowledge on MML nor were concerned with the development of MML. MML was applied multiple times in two kinds of projects:

1. The practical course “Multimedia Programming” (“Multimedia-Programmierung”) [MMPf]. In this annual course about thirty students in teams of 5 to 7 people have to develop a multimedia application over the period of three months.

2. Project theses where a single student develops a multimedia application over the period of three to six months.

The applications developed in “Multimedia-Programmierung” were computer games similar like the Racing Game example in the previous chapters. However, the applications developed in the project theses are from different domains. In that way it is also shown that MML is not limited to the domain of computer games.

The following sections describe the projects and summarize the resulting observations.

8.2.1 MML in the Course "Multimedia-Programming"

The annual course "Multimedia Programming" consists of a lecture held by Prof. Hußmann, several exercises, and, as main part, a practical project where students have to develop a multimedia application on their own. The exercises and the practical project were supervised from 2004 until 2008 by the author of this thesis together with student assistants. The main implementation technology applied in this course is Flash/ActionScript.

While the course in its current form was held first in 2004, MML was mainly applied in 2006 and 2007. None of the participating students had previous knowledge of MML. They were informed that MML is developed by the Media Informatics Group but not that MML is part of a Phd thesis or that it is applied in the course for a kind of test.

The observations after each course were integrated in MML. So the students worked with different versions and on different stages of tool support.

The following paragraphs describe how MML was applied over the different years and the results.

Preliminary Experience in 2004 and 2005 The course in 2004 [MMPa] had no connection to MML. However, the students had to apply the *Extreme Programming* paradigm to their practical projects. They were supported to apply it as far as possible and also had to frequently report to their supervisors about their experience. Also, a questionnaire was filled out by all participating students at the end of the project. This provided new insights on problems and benefits when applying Extreme Programming to a multimedia project. Some of them are mentioned in section 3.2.

The applications to be developed by the students in 2004 were Racing Game applications similar to the example in this thesis. A restricted version of the final results can be found at [MMPb]. Please note that all applications (also in the following editions of the course) also provide a multiplayer mode (which makes a significant part of the overall development effort) which is not available in the versions on the webpage as it requires an additional multiplayer server.

In 2005 [MMPc], MML was applied the first time but only to a very limited extent. MML itself was applied only as part of an exercise. The students then had to submit a modeling task with MML as their homework. All submissions were briefly reviewed and each student received written feedback on his/her models. This time, the students had to apply in their projects either the Extreme Programming paradigm or a simplified version of the process proposed by Osswald [Osswald03]. Osswald's process is a version of the *Rational Unified Process* specifically adapted for multimedia applications (see sec. 3.1).

Application and Questionnaire in 2006 In 2006 [MMPd], MML was applied in exercises and as homework as well. But this time the students also had to use MML in the practical project. Altogether 35 students took part in the practical project in six teams of six to seven students each. Each team had to apply a given development process which was an adapted version of the process from Osswald based on the experience from 2005. It is depicted in figure 8.3. After each phase a milestone was defined which had to be met by the students to ensure that they adhere to the process.

According to the process (fig. 8.3), each team first had to plan the team organization (communication, infrastructure, etc.) and to fix the main agreements in a written document. Subsequently, they had to find ideas how to realize the application in terms of the gameplay and the overall theme for the visual design. These ideas had to be presented to the supervisors by created prototypes. Additional prototypes were created to get an idea on the implementation and the required algorithms.

On that base, each team had to design the application in terms of an MML model which was then discussed with their supervisor and, if necessary, revised. They had to use all kinds of MML

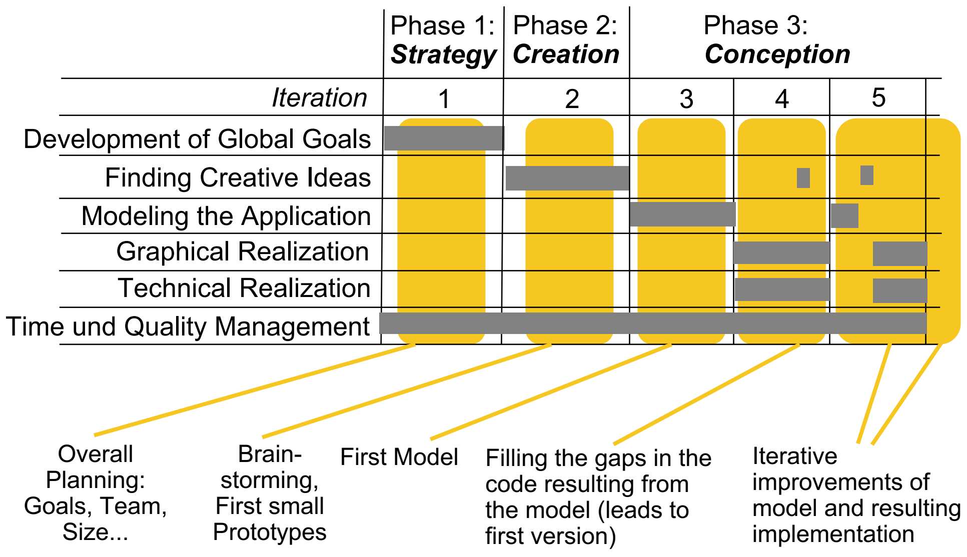

Phase 1: Strategy
Phase 2: Creation
Phase 3: Conception
Iteration
1
2
3
4
5
Development of Global Goals
Finding Creative Ideas
Modeling the Application
Graphical Realization
Technical Realization
Time und Quality Management
Overall Planning: Goals, Team, Size...
Brain-storming, First small Prototypes
First Model
Filling the gaps in the code resulting from the model (leads to first version)
Iterative improvements of model and resulting implementation

Figure 8.3: Development process for the practical project in 2006

models, i.e. Structural Model, Scene Model, Presentation Model, and Interaction Model. However, the Interaction Model was used only for a purely informative description of the interaction concepts in their application as it is the least supported model type yet (see sec. 6.5).

The students used the MagicDraw-based editor (sec. 6.7) to create and edit their MML models. The transformations for the mapping into Flash code were not finally implemented to that time. However, the framework proposed in section 7.2 for structuring Flash was already available. There was a written documentation for the framework and an example project which students could start with. So the students could manually map their MML models into a Flash implementation. However, using this framework was optional for the students.

The task was to develop a 'Jump and Run' game with multiplayer support. Besides others, the main requirements were:

- Animated characters,
- collision detection of animated characters with walls, platforms, bonus objects etc.,
- a camera which shows the current part of the level and moves with the player's character,
- different kinds of bonus objects and obstacles,
- a singleplayer mode and a multiplayer mode including chat functionality, and
- different screens for menu, options, help, highscore, etc.

Figure 8.4 shows some exemplary screenshots from one of the six results. All results can be found at [MMPe] (the multiplayer mode is disabled).

An important result from the project were the general observations on the usage of MML, what worked, and what needed to be improved. Problems with the modeling language itself, like missing expressiveness, were collected in a list and considered in the next version of MML. An example issue is that *Sensors* are not always children of a Media Component but can also be assigned to a Media Instance in a Scene (see sec. 6.4.3).

After the project was finished, the students had to answer a questionnaire about

- their previous experience on programming and development,
- the usage of MML in their project,

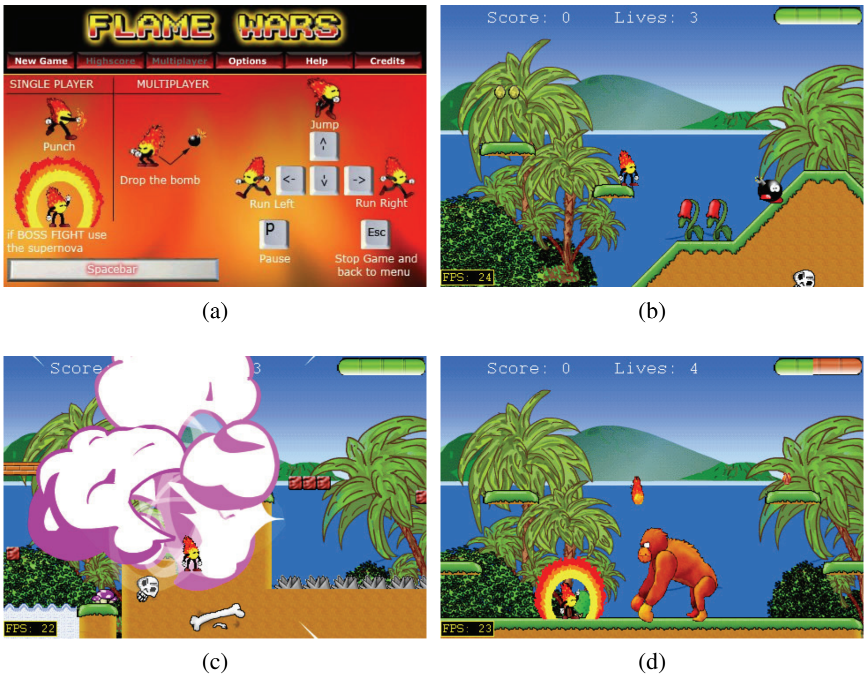


Figure 8.4: Screenshots from the result of Team 1 in 2006.

- the MagicDraw-based modeling tool for MML,
- the framework for structuring Flash applications, and
- their opinion about the applicability of MML.

It is attached in appendix D (in German language). The participants had to answer the questions in the questionnaire (except those on their previous experience) by stating their agreement to given statements. Thereby, a Likert Scale was used from 1 to 5 with the meaning: 1 = 'Strongly disagree', 3 = 'Neither agree or disagree', and 5 = 'Strongly agree'. The questionnaire was answered by 27 students.

The main findings are (values in terms of the Likert Scale):

- The respondents found a design phase in general useful (average 4,2).
- Students who used the proposed Flash framework (usage > 2) found that it is of high quality (average 4,2)
- Compared to the overall project time, the estimated percentage required for modeling was relatively low. Some critical students stated a implausible high percentage value like 70 percent. However, the median value is still (only) 5 percent.

It turned out that students had very different opinions about the development process with MML. The resulting average of the most answers was slightly positive (average values slightly above 3) but without significant findings. Also, no correlation was found between the stated previous experience on programming and development and the grading of MML.

The students also had the possibility to add own comments. Again, some comments were very positive about MML while others were very critical against it. Critical comments included that it is difficult and takes too much effort to plan the application in such a level of detail like in MML. Students who added positive comments to the questionnaire stated that they found MML helpful to plan the application, structure the system, and distribute the development tasks within the team.

Application in 2007 In 2007 [MMPf], MML was used again in the exercise and in the project. Seven teams with altogether 37 students participated the project. The development process, the usage of the modeling language, and the usage of Magic Draw was analogous to 2006. However, this time the ATL transformations were already available to automatically generate Flash code skeletons from the models (see sec. 7). However, the students were allowed to decide themselves whether they wanted to use the automatic transformation or not. Four of the seven teams (team3, team 4, team 5, team6) asked for automatic code generation. As executing the ATL transformations requires some basic knowledge of ATL, it was executed by the author of this thesis. All teams had to submit their models and those teams interested in code generation received the resulting code.

The task that year was a minigolf game with multiplayer mode. The basic requirements were as follows:

- mechanism to control the hits with the club,
- realistic simulation of ball physics like gravity,
- the courses should simulate different heights (e.g. ramps and stairs),
- different kind of obstacles including moving obstacles, elements to accelerate or slow down the ball, etc.,
- singleplayer mode with at least 12 holes,
- multiplayer mode where players can interact like placing obstacles on the course, place bets for the next hole, etc., and
- different screens for menu, options, help, highscore, etc.

Figure 8.5 shows screenshots of some of the resulting games. One result was even published on the website of *München TV*, a local TV station [Wie].

The experience in that year was that there were rarely problems with the modeling language itself. One expected problem for the code generation was that the simple modeling tool based on MagicDraw does not provide support to create valid models. Surprisingly, it took only little effort to generate code from the models received from the teams. Problems were mainly caused by incomplete information about Domain Classes, like missing or undefined attribute types, missing names of association ends, or artifacts which unintentionally were deleted only from the diagrams but not from the model, etc. Such mistakes seem not to indicate conceptual problems. However, it should be mentioned that one of the seven teams had general problems with the modeling phase and needed detailed help from the supervisor to finally create a meaningful model.

Unfortunately, the students did not make use of the generated code or the proposed Flash framework that year. All teams which asked for automatic code generation stated that they first planned to use this opportunity, as it was for free. However, as the resulting code consists of a large amount of files and folders (see sec. 7.2) they were concerned that it would take too much effort to learn the structure. They also stated, that it is very difficult, in particular for team members without previous knowledge of Flash, to learn Flash and the generated structure in parallel. Nevertheless, some students who are more interested in modeling later asked for future topics on MML and performed on their on initiative a project thesis in the context of MML.

In general, it was conspicuous, that in this year the teams, independent from MML, did not make use of tools or structuring mechanisms beyond the Flash authoring tool. For instance, in 2006 almost all teams used third-party Eclipse plugins for programming the ActionScript code. They also made heavy usage of structuring mechanisms like the proposed framework for Flash as well as own ideas

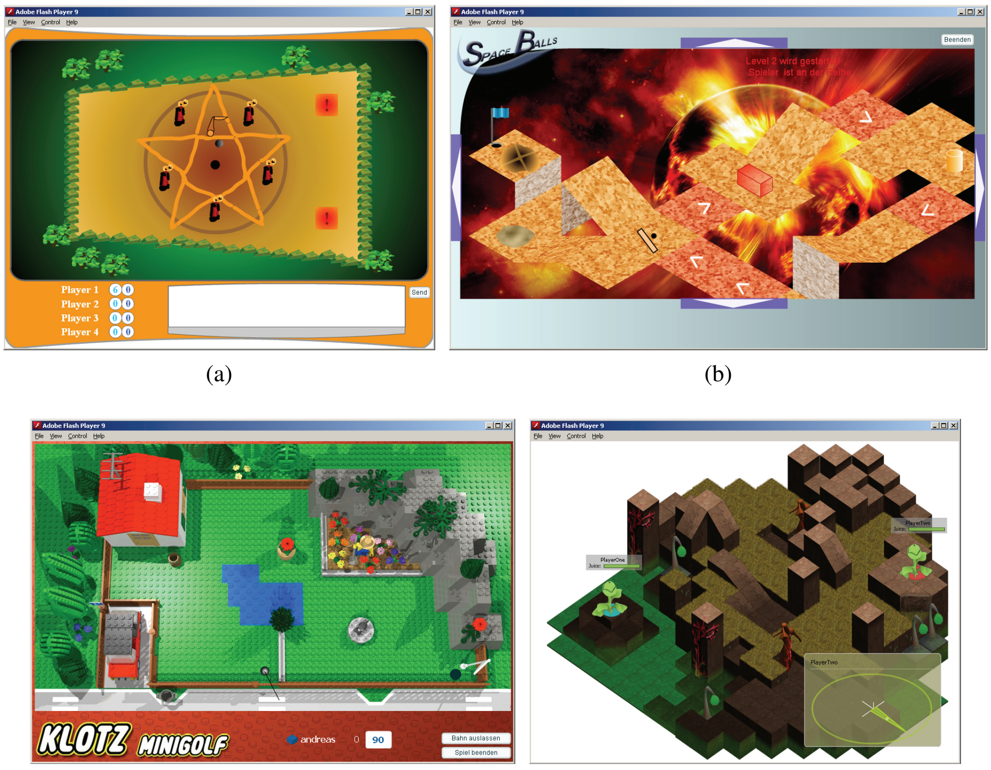


Figure 8.5: Screenshots from different results in 2007.

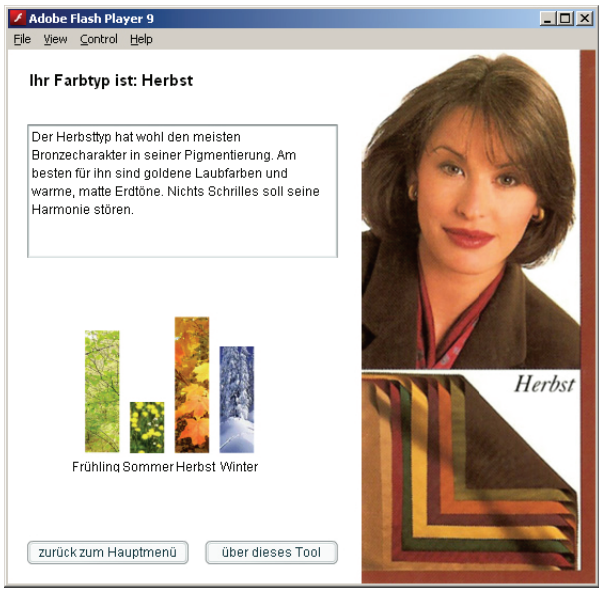
to structure the code in a modular way. In contrast, in 2007 only one team used the Eclipse plugin and most teams used a much more monolithic structure for their application. There was no specific reason found for this difference. At least, this shows that even with a large number of participants the influencing factors for such projects are too complex to generalize the experience in general.

8.2.2 MML in Project Theses

Besides the application in "Multimedia-Programming", MML was also applied in several project theses. In these theses the students modeled an application using MML and used the transformation into code for the final implementation. The applications had to be developed for third-party customers who were frequently in contact with the students. So, the students' main motivation was the resulting product itself and to fulfill the customer's requirements – not the use of MML.

The projects described here were implemented with Flash/ActionScript. The following paragraphs show the three projects in chronological order.

The StyleAdvisor for a Munich Hairdresser Kraiker [Kraiker07] developed an application for a Munich hairdresser called *StyleAdvisor*. Its purpose is to determine the best fitting color scheme for the user in terms of clothing, jewelery, and make-up. It mainly uses text and images to provide the user with several questions and calculates from the answers the user's color scheme. Figure 8.6 shows a screenshot from the application.


Ihr Farbtyp ist: Herbst

Der Herbsttyp hat wohl den meisten Bronzecharakter in seiner Pigmentierung. Am besten für ihn sind goldene Laubfarben und warme, matte Erdtöne. Nichts Schrilles soll seine Harmonie stören.

Figure 8.6: Screenshot of the StyleAdvisor application for a hairdresser.

The entire code generated from the MML models was used for the final implementation without noteworthy problems (see discussion in [Kraiker07]). In his final project presentation the student reported that he actually was surprised about the good result the code generator produced from the models. He also had the feeling that the model-driven process with MML and automatic code generation saved an significant amount of time compared to manual implementation.

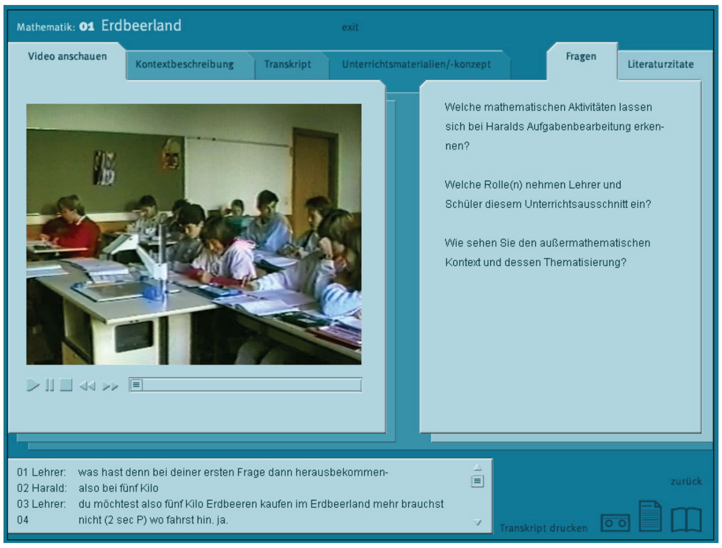
An Authoring Tool for the "Unterrichtsmitschau" In his project thesis [Kauntz07] Kauntz developed an authoring tool for the group for *Unterrichtsmitschau und didaktische Forschung* (short: *Unterrichtsmitschau*; [Unta]) at the *Department für Pädagogik und Rehabilitation* (department for pedagogy) at the University of Munich. This group produces, amongst others, multimedia learning applications for pedagogues and teachers. The applications are commercial products and can be ordered via online shop [Untb].

Figure 8.7 shows an example for such an application (here from taken from [Jungwirth and Stadler03]). It consists of different regions and tabs containing different kinds of learning content. The most important content are the videos which show for instance a teaching lesson in a school. The videos are accompanied by a transcript which runs synchronously to the videos but can also be viewed or printed independently from the videos. Other content is for instance explanations of important observations from the video or control questions.

The task in [Kauntz07] was to develop an authoring tool which supports the Unterrichtsmitschau for the production of such applications. Thereby, it should fulfill the following requirements:

- import of text documents, images, and videos,
- create a hierarchical structure for the application,
- define different layouts for the application,
- define the synchronization between the videos and the transcript, and
- store the applications in a non-proprietary format.

Figure 8.8 shows screenshots from the authoring tool. Figure 8.8a show the main screen of the



Welche mathematischen Aktivitäten lassen sich bei Haralds Aufgabenbearbeitung erkennen?

Welche Rolle(n) nehmen Lehrer und Schüler diesem Unterrichtsausschnitt ein?

Wie sehen Sie den außermathematischen Kontext und dessen Thematisierung?

```
01 Lehrer:   was hast denn bei deiner ersten Frage dann herausbekommen-
02 Harald:   also bei fünf Kilo
03 Lehrer:   du möchtest also fünf Kilo Erdbeeren kaufen im Erdbeerland mehr brauchst
04           nicht (2 sec P) wo fahrst hin. ja.
```

Figure 8.7: Example for a learning application with transcript.

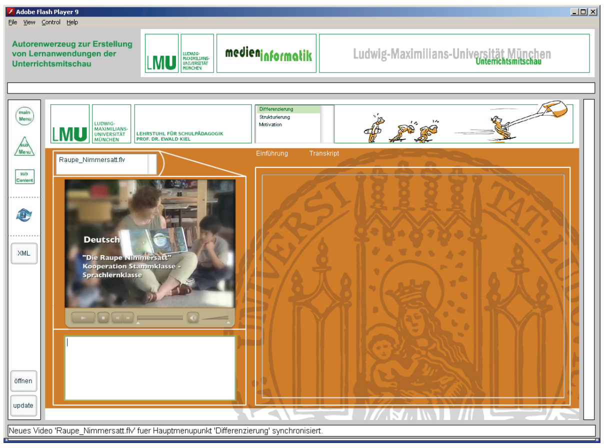
tool. The center area shows the learning application under development. Some parts can be edited directly in the center screen and some with help of the items in the toolbar on the left hand side. Figures 8.8b-8.8d show dialogues for creating sections (in the learning application to be created), importing documents, and synchronizing videos and transcripts.

To a large extent the project thesis was supervised by an external supervisor from the Unterrichtsmitschau. The author of this thesis provided supervision for the development process itself but was not involved in the requirements specification for the tool. Those were elaborated by Kauntz in cooperation with his external supervisor who was the "customer" for this project. Kauntz used several techniques for the requirement analysis, including (see [Kauntz07] for details):

- ConcurTaskTrees (see sec. 4.1.2),
- detailed Use Cases (including preconditions, postconditions, basic course of events, etc.),
- Storyboards, and
- a simple user interface prototype (click-dummy) implemented with Flash.

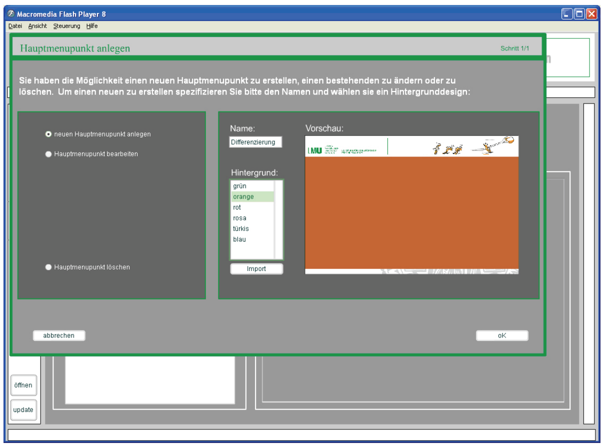
Subsequently, the application design was specified using MML models. The models were transformed into code skeletons for Flash/ActionScript which were then finalized as described in section 7. The authoring tool uses mainly Media Components which are generated at runtime. In these cases, the generated placeholders were just deleted and, instead, code was added to the generated ActionScript classes which creates or loads the Media Components at runtime. [Kauntz07] reports that the final implementation is very close to the MML models. Also, there was no need to change the generated code structure. Similar like Kraiker (see above), Kauntz had the feeling that the process with MML was significantly faster than if he would have had to plan and implement application structure manually.

CMD Tutorial for Siemens Finkenzeller [Finkenzeller08] developed in a tutorial for an existing application for customer relationship management at Siemens called *Corporate Master Data* (*CMD*) . The tutorial is called *CMD Tutorial*. It shows the different screens of CMD and explains them. The user can also click directly on user interface elements in the depicted screens which opens additional help for the selected element.



Neues Video 'Raupe_Nimmersatt.flv' fuer Hauptmenupunkt 'Differenzierung' synchronisiert

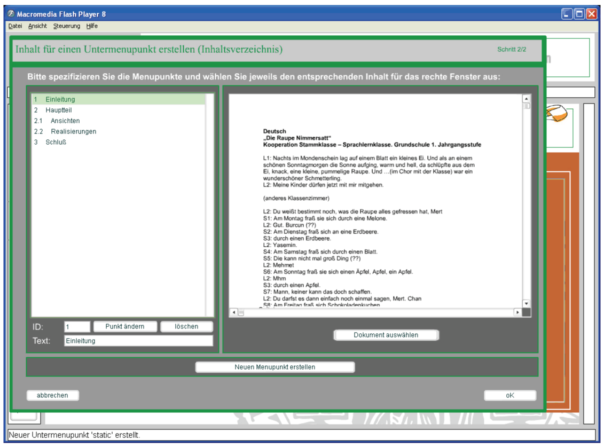
(a) Main screen


Macromedia Flash Player 8
Datei Ansicht Steuerung Hilfe
Hauptmenupunkt anlegen
Schritt 1/1
neuen Hauptmenupunkt anlegen
Hauptmenupunkt bearbeiten
Hauptmenupunkt löschen
Vorschau:
Import
abbrechen
oK
öffnen
update
Sie haben die Möglichkeit einen neuen Hauptmenupunkt zu erstellen, einen bestehenden zu ändern oder zu löschen. Um einen neuen zu erstellen spezifizieren Sie bitte den Namen und wählen sie ein Hintergrunddesign:

Name: Differenzierung

Hintergrund: grün, orange, rot, rosa, türkis, blau

(b) Creating a main menu item


Macromedia Flash Player 8
Datei Ansicht Steuerung Hilfe
Inhalt für einen Untermenupunkt erstellen (Inhaltsverzeichnis)
Schritt 2/2
Punkt ändern
löschen
Dokument auswählen
Neuen Menupunkt erstellen
abbrechen
oK
Bitte spezifizieren Sie die Menupunkte und wählen Sie jeweils den entsprechenden Inhalt für das rechte Fenster aus:

1 Einleitung
2 Hauptteil
2.1 Ansichten
2.2 Realisierungen
3 Schluß

ID: 1
Text: Einleitung

Deutsch
„Die Raupe Nimmersatt“
Kooperation Stammklasse – Sprachlernklasse. Grundschule 1. Jahrgangsstufe

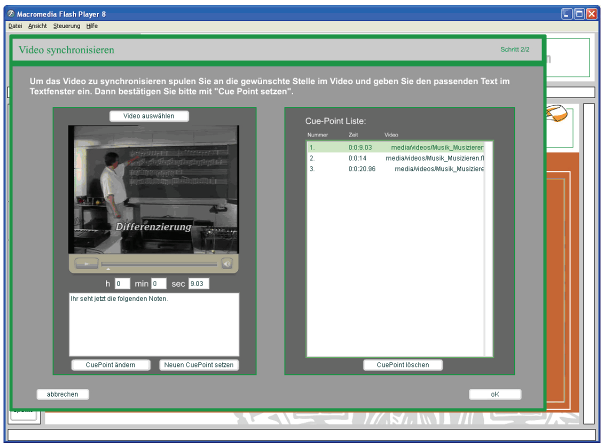
L1: Nachts im Mondenschein lag auf einem Blatt ein kleines Ei. Und als an einem schönen Sonntagmorgen die Sonne aufging, warm und hell, da schlüpfte aus dem Ei, knack, eine kleine, pummelige Raupe. Und …(im Chor mit der Klasse) war ein wunderschöner Schmetterling.
L2: Meine Kinder dürfen jetzt mit mir mitgehen.

(anderes Klassenzimmer)

L2: Du weißt bestimmt noch, was die Raupe alles gefressen hat, Mert
S1: Am Montag fraß sie sich durch eine Melone.
L2: Gut. Burcun (??)
S2: Am Dienstag fraß sich an eine Erdbeere.
S3: durch einen Erdbeere.
L2: Yasemin.
S4: Am Samstag fraß sich durch einen Blatt.
S5: Die kann nicht mal groß Ding (??)
L2: Mehmet
S6: Am Sonntag fraß sie sich einen Äpfel, Apfel, ein Apfel.
L2: Mhm
S3: durch einen Apfel.
S7: Mann, keiner kann das doch schaffen.
L2: Du darfst es dann einfach noch einmal sagen, Mert. Chan
S8: Am Freitag fraß sich Schokoladenkuchen

Neuer Untermenupunkt 'static' erstellt

(c) Importing a document



Um das Video zu synchronisieren spulen Sie an die gewünschte Stelle im Video und geben Sie den passenden Text im Textfenster ein. Dann bestätigen Sie bitte mit "Cue Point setzen".

h 0 min 0 sec 9.03

Ihr seht jetzt die folgenden Noten.

Cue-Point Liste:

Nummer	Zeit	Video
1.	0:0:9.03	media/videos/Musik_Musizierer
2.	0:0:14	media/videos/Musik_Musizieren.fl
3.	0:0:20.96	media/videos/Musik_Musiziere

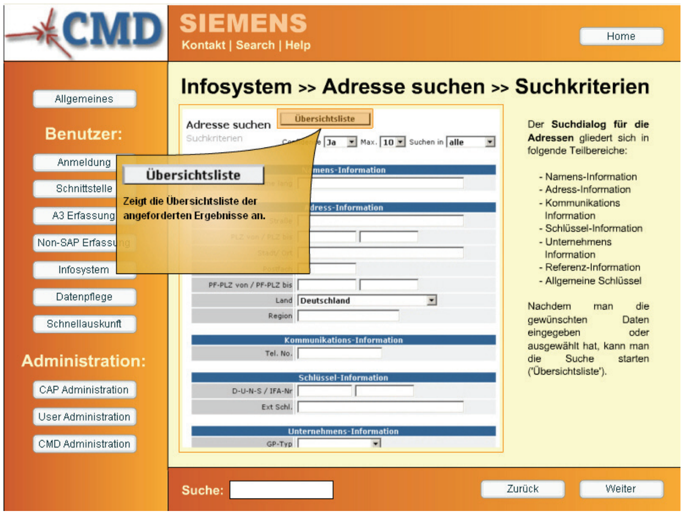
(d) Synchronizing a video

Figure 8.8: Screenshots from the authoring tool for the Unterrichtsmitschau.

Figure 8.9 shows a screenshot of the application. The buttons on the left hand side enable navigating between the different chapters. The center area shows a part of a screen from the CMD application while the right hand side shows the corresponding explanations.

This application provides only a small degree of interactivity and application logic; only navigation and the possibility to select elements in the images of CMD. It was specified as MML model from which code was generated. However, in such cases the resulting very modular code structure is unnecessary. Thus, Finkenzeller implemented two versions: one manual implementation for Siemens with a much more simplified structure (e.g. a single FLA file for the whole application instead of one FLA file for each Scene) and one prototype from the generated code to demonstrate that the generated structure is basically also sufficient for such a kind of application. Beside the complexity of the generated code, no other problems with MML occurred.

This case shows that the code structure generated from the models is not optimal for all kinds of applications. As described in section (sec. 7.2.1) the goal for the proposed Flash framework used as target structure for the transformation was a high degree of flexibility and modularity to support large and complex applications. It is possible, of course, to create an alternative transformation which maps



Übersichtsliste

Zeigt die Übersichtsliste der angeforderten Ergebnisse an.

Der **Suchdialog für die Adressen** gliedert sich in folgende Teilbereiche:

- Namens-Information
- Adress-Information
- Kommunikations Information
- Schlüssel-Information
- Unternehmens Information
- Referenz-Information
- Allgemeine Schlüssel

Nachdem man die gewünschten Daten eingegeben oder ausgewählt hat, kann man die Suche starten ('Übersichtsliste').

Figure 8.9: Example for a learning application with transcript.

an MML model into a Flash application with a simple structure. This can be useful for quite "simple" applications like the CMD Tutorial but also, for instance, for applications on target platforms with limited hardware resources. A concrete example for such an alternative transformation is presented in section 8.1.3 for the example platform Flash Lite (it was not available at the time when the CMD Tutorial was developed).

Nevertheless, the project also shows that MML itself can also be applied for applications like the CMD Tutorial.

8.2.3 Conclusions

The projects in multimedia programming enabled to gain practical experience on the development with MML. However, the experience from these projects confirms the expected problems explained in section 8: It is indeed very hard to validate a complex development approach like MML in academic context. Due to the large amount of influencing factors it is hardly possible to generalize the observed results.

Nevertheless, the observations made during the usage of MML by many different external people were considerably helpful to improve and refine MML. During the projects, no fundamental problems occurred with MML. Minor problems observed on single modeling concepts have been taken up and considered for the MML version presented here. The proposed framework for Flash applications was graded very well by all participants who used it. Also the transformation itself worked for all models it was applied on. Altogether, this provides at least a positive indication for the general feasibility of MML.

8.3 Internal Validation

In section 8, two feasible validation techniques have been identified for internal validation: validation against given criteria and validation compared to other approaches.

Evaluation against given criteria is a common technique for modeling languages. On the one hand, existing literature provides some quality criteria for modeling languages. Thus, section 8.3.1

examines whether they are fulfilled by MML. On the other hand, several goals and requirements to be fulfilled by the solution have been elaborated during this thesis. They are examined in section 8.3.2.

Validation compared to other approaches can only be performed here on a theoretical base by a brief comparison to existing modeling languages based on the related work from chapter 4. It is presented in section 8.3.3.

8.3.1 Comparison to General Criteria for Modeling Languages

The existing literature provides several lists or catalogs of criteria to be addressed by modeling languages. An often cited work is the *Cognitive Dimension Framework* by Green [Green and Petre96, Green00]. It provides a framework to evaluate the usability of visual artifacts in general and has been applied in particular to visual languages in Software Engineering [Green and Petre96, Green00, Blackwell et al.01].

While this work addresses mainly the visual aspects of a language, other work focuses more specifically on modeling languages and addresses in addition aspects like the language definition, ontological aspects, etc., [Paige et al.00, Shehory and Sturm01]. A very comprehensive "evaluation schema for visual modeling languages" is proposed in [Frank and Prasse97] (in German language). It orders the criteria into four classes: Language description, language concepts, language application, and general criteria. The authors discuss the criteria and how they should be addressed by a modeling language. They also provide some recommendations how to evaluate the criteria for a given modeling language.

The schema from [Frank and Prasse97] is used in the following to evaluate MML. Each issue in the list (denoted in italics) represents a criteria from [Frank and Prasse97] to be supported of fulfilled by a modeling language. The order of the list items corresponds to the table on pages 41-44 in [Frank and Prasse97][2]. Each issue is followed by a statement how it is addressed in MML. For a general discussion of each issue please refer to [Frank and Prasse97].

1. **Description of the modeling language**
   1. Definition
      1. *The type of the grammar or metamodel used to define the language (if any)*
         MML is defined by a standard-compliant metamodel. It is compliant to MOF and to Ecore. The latter is demonstrated by the EMF-based implementation
   2. Documentation:
      1. *Documentation of the language specification*
         The MML metamodel is shown in this thesis.
      2. *User documentation*
         MML is described in this thesis by a textual description.
2. **Language**
   1. **Language Structure**
      1. *The language can be monolithic or consist of different parts ("sub-languages"). A common practice in metamodel-based languages is to divide the metamodel into several sub-metamodels*
         MML uses the described technique for metamodels: The overall metamodel consists of several sub-models for the Structural Model, the Scene Model, the Presentation Model, and the Interaction Model.
   2. **Modeling Concepts:**

[2]The criteria have been translated into the English language here

      1. *Object-oriented concepts*
         The MML Structure Model is based on UML class diagrams and supports all basic object-oriented concepts. The concept of modeling Media Components is derived from UML Components.
      2. *Concepts for modularization of models*
         The Structure Model in MML supports the same modularization concepts like UML. It is possible to us packages to structure the Domain Classes if necessary. If necessary, Composite Scenes can be used to structure the Scene Model. The Presentation Model and the Interaction Models are structured by the Scenes.
      3. *Concepts for modeling the application's dynamic behavior*
         The coarse-grained behavior within a Scene is described in the Interaction Model. The detailed behavior of Domain Classes is explicitly not part of the model as it is intended to implement it manually for the target platform. Basically, as the MML reuses UML, it is possible to add UML concepts to model the behavior of Domain Classes
      4. *Concepts for modeling concurrency and parallel processing*
         Can be modeled in Interaction Models but is not specifically supported in MML yet.
      5. *Concepts for process-oriented modeling, like business processes*
         Basically supported by the Interaction Model.
      6. *Integration of modeling concepts*
         The integration of all modeling concepts is defined in the metamodel and in section 6.6.
   3. Notation:
      1. *Diagram types to visualize different aspects of the application*
         Each model in MML is supported by a diagram type of its own, i.e. Structure Diagram, Scene Diagram, Presentation Diagram, and Interaction Diagram. They provide different views on the system analogous like in other modeling approaches (like OMMMA, sec. 4.3.2).
      2. *Notation of diagram elements and resulting usability of the modeling language*
         MML reuses as much as possible established existing notations. New visual icons for media components have been evaluated in first user tests. The notation for inner properties of Media Components is partially taken from UML Components (different compartments, Media Artifacts, etc.) and from Scene Graphs (inner structure). A definitive evaluation of the notation for Media Components would be a useful task for future work.
      3. *Annotations*
         MML supports to add comments to any kind of model element, like UML.
3. **Application**
   1. Views
      1. *User's View – support for communication between developers and tool support*
         MML is designed in such a way that all information of a model can be visualized in the diagram and thus also in print-outs. It is possible to draw MML diagrams by hand (e.g. on a paper or whiteboard for discussions). MML is supported by a visual modeling tool described in section 6.7.
      2. *Meta view – extension and specialization of the language*
         MML is defined as explicit metamodel and is intended to be combined with other approaches. This issue is briefly discussed in section 10.3.
   2. Purpose
      1. *Purpose and intended coverage of the modeling language* The purpose and intended coverage of MML are defined in section 5.1.1.
   3. Development tasks:

1. *Analysis*
   MML aims not to provide support for an analysis phase. The relationships to typical artifacts from an analysis phase are explained in section 6.6.1.
2. *Design*
   As explained above, section 6.6.1 describes how MML models are created starting with the results from the analysis phase. MML enables to specify the application so that it is possible to derive the implementation from the models (see transformation in section 7 and transformations for other platforms in section 8.1).
3. *Implementation*
   MML supports the implementation by the transformation into code skeletons. Section 7.4 describes how to work with the code skeletons.
4. *Verification and Validation* Due to the usage of concepts from Model-Driven Engineering, it is possible to validate models using OCL constraints. [Shaykhit07] (a diploma thesis supervised by the author of this thesis) demonstrates this by applying an Eclipse OCL plugin [Ecla, Damus07].
5. *Integration of concepts for different phases*
   See sections 6.6.1 and 7.4.

4. **General criteria**
   1. User-related criteria:
      1. *Feasibility of the modeling language*
         The feasibility is demonstrated by the running example in this thesis and the projects in section 8.2.
      2. *Clearness and understandability of the language and the resulting models*
         Clearness and understandability are addressed by the extensive reuse of existing established modeling concepts. The student projects in section 8.2 have been used to test the clearness and understandability with external participants.
      3. *The language should be adequate compared to its purpose*
         A comparison to the goals of MML is provided in section 8.3.2.
      4. *Models can be validated compared to the reality to be modeled*
         The automatic transformations into code can help to find out whether a model specifies a system as intended.
      5. *Expressiveness of the modeling language*
         The examples in student projects (sec. 8.2), the coverage of different kinds of multimedia applications (sec. 8.3.3), and the possibility to generate code for different platforms (sec. 8.1) indicate that the language is expressive enough to meet the goals of this work.
   2. Model-related critera:
      1. *Abstract syntax and semantics must not be ambiguous*
         The transformations from MML models to code help to avoid ambiguousness as they indirectly define the meaning of a model element.
      2. *The language should support consistency of the models*
         The abstract syntax in the metamodel and the well-formedness rules aim to some extent to contribute that the models are consistent.
      3. *Adequate degree of formalization*
         The transformation into code indirectly provides to some degree a formal definition of MML's semantics.
      4. *Integration of the modeling concepts*
         The integration of models is described in the metamodel and in section 6.6.

3. Economical criteria:
   1. *Reusability of language elements and model elements*
      Language elements can be reused as they are defined in an explicit metamodel according to the concepts of Model Driven Engineering (see sec. 8.3.2). Within the models (and the resulting code) itself, MML provides explicit support to reuse elements from the Structural Model in multiple Scenes. Reuse of user interface elements is not supported yet. A possible extension would be to allow e.g. a UIContainer to be reused in multiple PresentationUnits (see sec. 6.4.2).
   2. *Extensibility of the modeling language*
      This issue is briefly discussed in section 10.3.

Altogether, the list shows that MML addresses all criteria. Two minor issues have been identified for possible future work on MML:

- Find possible alternative notations for Media Components (beyond the icons) and evaluate them in user tests (issue 2).
- Support for reuse of user interface elements like in other existing user interface modeling approaches (issue 1).

8.3.2 Comparison to the Goals of This Thesis

This section examines whether MML fulfills the goals and requirements defined throughout this thesis.

Fulfillment of Basic Goals Section3.3 defined three general goals for MML (page 34). They are fulfilled as follows:

1. *Integration of media design, user interface design, and software design into a single, consistent modeling approach.*
   The modeling language integrates these three areas. This is illustrated in figure 6.25.

2. *A level of abstraction which enables code generation but ensures a lightweight approach.*
   Section 7.3 demonstrates that the level of abstraction in MML is sufficient for code generation. In addition, section 8.1 provides transformations to other target platforms.
   The approach is 'lightweight' in that way that the models contain only information about the overall application structure while those application parts which are very tedious to model (like the detailed application logic, user interface layout, and the media design etc.) are not part of the models. They have to be implemented directly in the authoring tool.

3. *Advanced integration of authoring tools.*
   The integration of authoring tools is demonstrated in section 7.

Thus, all the basic goals are fulfilled by MML.

Coverage according to the Classification for Multimedia Applications Table 2.1 in section 2.4 provides a classification of multimedia applications. This section examines to which extent the identified spectrum of multimedia applications is covered by the examples presented in this thesis.

In table 8.1, the examples from the external validation (sec. 8.2) were added to the table (see the highlighted classes).

Interactivity / Domain	Business	Information	Communication	Edutainment	Education
Directive	Authoring Tool R G Authoring Tool Unterrichtsmitschau R G		CSCW System R	City-building Game D G	Electronic Circuit Simulation D G
Proactive	Car Configurator D	Navigation System G	Video Conference R	Car Racing Game D Games from „Multimedia-Programmierung“ D	Flight Simulator D G
Reactive	Online Shop D	Encyclopedia R StyleAdvisor D		Media Player R	Medical Course D CMD Tutorial D

Media Origin:		
	R	*Received*
	D	*Designed*
	G	*Generated*

Table 8.1: Coverage of the examples from section 8.2.

The table shows that the examples presented in this thesis cover all three degrees of interactivity and at least four of the five application domains. Moreover, they cover all three kinds of 'media origin'. Finally, they cover also all media types, except 3D animation:

- *Video* is covered by the authoring tool for the "Unterrichtsmitschau",
- *Sound* is covered by the various games from "Multimedia-Programmierung",
- *2D animation* is covered by the games from "Multimedia-Programmierung",
- *Image* is covered by the StyleAdvisor, the CMD Tutorial, and the games from "Multimedia-Programmierung", and
- *Graphics* is covered by the games from "Multimedia-Programmierung".

3D animation is not covered yet as it is not supported by Flash. However, the work in [Vitzthum08] has already demonstrated that it is feasible to generate useful code skeletons for 3D formats like VRML [ISO97a] or X3D [ISO04].

Altogether, the MML modeling examples from external evaluation cover all values (except 3D) of all four facets from the identified classification for multimedia applications. This indicates that MML is basically adequate to model any kind of multimedia application.

Comparison to Requirements of Model-Driven Engineering MML aims to comply to the concepts of Model-Driven Engineering (*MDE*; also Model-Driven Development, *MDD*; see sec. 3.4). The current working definition for the MDA (as a possible realization of MDE) from page 37 claims the following requirements:

- *Usage of at least one modeling language*
  This is addressed with MML.
- *MOF-compliance of the modeling languages*
  MML is complaint to MOF. This is demonstrated by the implementation as Ecore metamodel

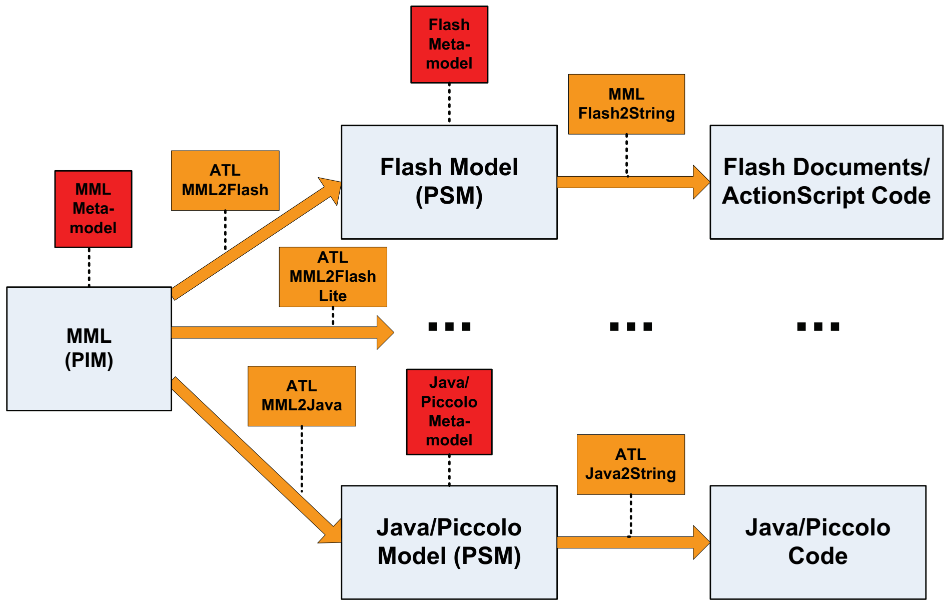


Figure 8.10: MML in terms of the Model Driven Architecture.

(using the with the Eclipse Modeling Framework, see sec. 6.7) which is an implementation of E-MOF (see [Gruhn et al.06]).

- *Automatic transformation.*
  The automatic transformations is implemented with ATL (sec. 7.3).

Figure 8.10 applies the models and transformations presented in this thesis to the MDA framework from figure 3.3[3].

Thus, MML is compliant to MDA and hence to MDE.

8.3.3 Comparison to Other Approaches

Section 4 has presented various modeling approaches which cover one or more aspects of interactive multimedia applications. It turned out that none of the existing approaches aims to cover multimedia *and* interactivity *and* user interface design. Moreover, during the discussion of MML in section 5.2 and chapter 6, several issues for advanced interactive multimedia applications were identified. This section first lists a collection of these issues. On this base, MML and the existing approaches are compared.

Requirements for Modeling Advanced Interactive Multimedia Applications The following lists summarizes requirements identified during the discussions in section 5.2 and chapter 6. They are formulated independent from MML.

1. *Media as First Class Entities:* It should be possible to define media as first class entities as the existence of specific (possibly complex) media can be direct requirement of the application.

[3]It should be mentioned that the *Computation Independent Model* (*CIM*) part of the MDA framework is outside the scope of MML as MML focuses on the design phase. However, it is basically possible to derive the MML Domain Model from a computation independent Domain Model or to create a transformation from a computation independent Task Model onto the Scene Models and the Presentation Model.

Moreover, media creation can be a complex and time-consuming process.
Example: The customer wants that a given video and a given 3D graphic is used for a learning application.

2. *Media properties and behavior:* In interactive applications it should be possible to specify manipulations on media.
Example: Set the rotation value of an animation in a racing game or pause a video if a Scene is canceled.
3. *Media can represent parts of the application logic:* It should be possible to model that a media component is directly associated with parts of the application logic, e.g. a Domain Class.
Example: The car animation in a racing game is associated with the Domain Class car. A direction in a car navigation system is associated with corresponding class or class property.
4. *Media as interaction objects:* It should be possible to use media objects for the user interaction
Example: The user clicks on video to trigger an action. The user drags an animation to input some value.
5. *Generic media:* Media can be generic as the concrete content can be loaded or generated dynamically at runtime.
Example: The user can choose between different types of cars loaded from a specific folder. A museum application shows a piece of art loaded from a database.
6. *Instantiation of media:* It should be possible to create multiple instances for media.
Example: A 3D component representing a rim in a 3D car configurator is instantiated four times. An image is used multiple times within an application.
7. *Inner structure:* It should be possible to specify inner parts of media to modify them independently or to use them for interaction.
Example: The wheels of a car animation should turn when the car drives through a corner. A click on specific region in a map triggers some action.
8. *Variations of media:* It should be possible to specify different variations for media.
Example: An application provides all videos in different resolutions. An animation containing text has different should have different sizes according to the text in different languages.
9. *Dynamic number of instances:* It should be possible to specify that the number of media on the user interface is calculated dynamically.
Example: The number of cars in a racing game depends on the selected number of players between 1 and 8. The number of videos in a videos in a video editing software is determined by the videos provided by the user.

These issues have been identified mainly due to analysis of existing multimedia applications and to personal experience during the external validation. As MML models aim to provide a certain level of abstraction there is no proof that this list is complete and it can be extended in the future based on further experience.

Comparison of Selected Approaches Table 8.2 compares some representative modeling approaches from section 4: UsiXML [Usi07] as representative for the User Interface Modeling Domain, UWE [Koch et al.07] from Web Engineering Domain, OMMMA [Engels and Sauer02] as one of the most influencing multimedia modeling approaches and two document-oriented multimedia modeling approaches, the one from Boll [Boll01] and Madeus [Villard et al.00]. They are introduced in chapter 4.

The comparison considers the support for application logic, user interface design, and interaction in general (first three rows in table 8.2). The support modeling for media is considered in more detail by the identified requirements from the list above.

	User Interface Modeling	**Web Engineering**	**Multimedia Modeling**			**MML**	
	UsiXML	**UWE**	**OMMMA**	**Boll**	**MADEUS**		
User Interface Design	✓	◎	▬	▬	▬	✓	Task Model, Presentation Model
Interaction	✓	✓	✓	◎	◎	✓	AIOs, Interaction Model
Application Logic	✓	✓	✓	▬	▬	✓	Domain Classes
Media:							
Media as First Class Entities	▬	▬	✓	✓	✓	✓	Media Components
Media properties and behavior:	✓	▬	▬	✓	✓	✓	Media Interfaces
Media can represent parts of the application logic:	✓	▬	✓	▬	▬	✓	Media Representation
Media as interaction objects:	✓	▬	▬	▬	▬	✓	Media Realization
Generic media:	▬	▬	▬	▬	▬	✓	Media Artifacts
Instantiation of media:	▬	▬	◎	✓	✓	✓	Media Instances
Inner structure:	▬	▬	▬	✓	✓	✓	Media Parts, Inner Properties, Part Artifacts
Variations of media:	✓	▬	▬	✓	✓	✓	Variations
Dynamic user interface:	▬	▬	▬	▬	▬	✓	Properties of AIOs (multiplicity and visibility); Entry Operations and parameters for Scenes

Table 8.2: Comparison of MML and selected other modeling languages.

As visible in the table, none of the approaches besides MML covers the user interface, interaction, application logic together with comprehensive media support.

It should also be kept in mind, that the integration of the listed requirements for advanced media usage is not trivial. This thesis has shown a detailed discussion of such a integration (see different abstraction layers in fig. 5.9 and 5.14) and provides a resulting consistent modeling concept for them. To the best knowledge of the author, MML is the only approach which provides such a concept.

The presented comparison regards only the features of the modeling language itself. Regarding model-driven development for interactive multimedia, including code generation for multimedia authoring tools like Flash, there is (to the knowledge of the author) no existing approach with similar goals besides MML.

Chapter 9

Outlook: Towards Better Integration of Software Engineering and Creative Design

This section gives an outlook on further work addressing a better integration of Software Engineering and creative design. It generalizes the idea presented in this thesis, to integrate models for systematic and well-structured software development and visual tools for creative design. Parts of this chapter are close to [Pleuß and Hußmann07].

9.1 Creative Design in Software Development

It is frequently emphasized during this thesis that multimedia authoring tools play an essential role for the creative design in multimedia development. However, multimedia is certainly not the only area where creative design is important. In fact, any kind of interactive application requires to consider user interface design [Dix et al.03, Shneiderman and Plaisant04] as the user interface often plays a key role for the application's success.

A common practice in user interface design is to adopt a *User Centered Design* [Vredenburg et al.02] process. It relies on frequent user feedback to interactively elaborate a user interface tailored to the user's needs. User feedback is obtained based on different kinds of prototypes like paper prototypes (in the early stage of the process), user interface mock-ups, and click-dummies (in a later stage). Such a process is usually performed by user interface specialists with knowledge in interaction design, graphics design, human perception, psychology, etc. Often, these specialists are not computer scientists and have only limited knowledge on Software Engineering methods and tools.

Instead, the typical tools in user interface design are drawing tools like *Adobe Illustrator* [Ill], image processing tools like *Adobe Photoshop* [Phoa], authoring tools like *Flash* (chapter 7), or 3D graphic tools like *3D Studio Max* [3DS]. Usually, many different tools are used throughout the design process. For instance, first user interface mock-ups are created with Photoshop and later on click-dummies with Flash.

In practice, the whole process can be quite complex involving different teams, different sub-processes, and a large number of different tools . Figure 9.1 shows a simplified high level sketch of the typical process and typical tools in a large user interface design department (about 50 people) of a very large company. It was elaborated during an analysis of the user interface design process in

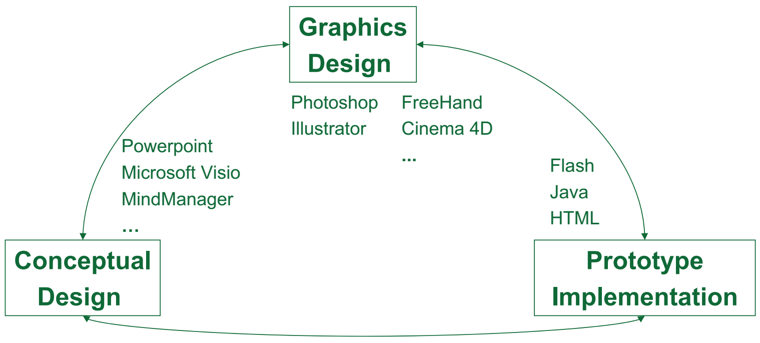


Figure 9.1: Simplified user interface design process and used tools in practice (based on [Ziegler08].

this department [Ziegler08][1], co-supervised by the author of this thesis. Ziegler was directly involved as a team member into the design process and gained her results by personal observation as well as by a large number of semi-structured interviews.

The advantage of using a large palette of different tools is that designers can select for each task the most efficient tool. Thereby, it is important that they are used to the tool so that they can quickly perform their tasks. On the other hand, the tool must be powerful enough to precisely realize the designer's creative ideas. However, the drawback is that the different tools are highly heterogeneous. They provide neither support for cooperative work nor for managing the created artifacts. Moreover, they use different file formats which are often incompatible.

Thus, handling the large amount of artifacts created over the time can become tedious. Possible problems are for instance [Ziegler08]:

- The different artifacts have to be managed manually, e.g. by storing them in a common folder structure.
- Changes in existing artifacts have to be propagated manually or by personal communication to other collaborators.
- Reuse of parts from existing artifacts is only possible by copy and paste, e.g. by searching for a specific part of a graphic in one document, cut it, and copy it into another screen.
- Often, results received from a previous step performed with a different tool have to be recreated again in the next step. A simple example is that user interface screens provided by the designer have to be manually recreated during prototype implementation.

One of the main ideas in this thesis is to use models and transformations for the integration of multimedia authoring tools and systematic development. The next section describes a proposal to generalize this idea like following: models and transformations can be used to integrate various heterogeneous tools with a more systematic development.

9.2 Vision: Models as Central Hub

Models are an excellent vehicle for integrating different stakeholders and different views on the system during the whole development process. Thus, in the vision described here, models are also used to integrate the different tools and the resulting artifacts. Thereby the concepts from model-driven development, like explicit transformations, are applied for computer-supported transitions between tools

[1]Unfortunately, this document in confidential.

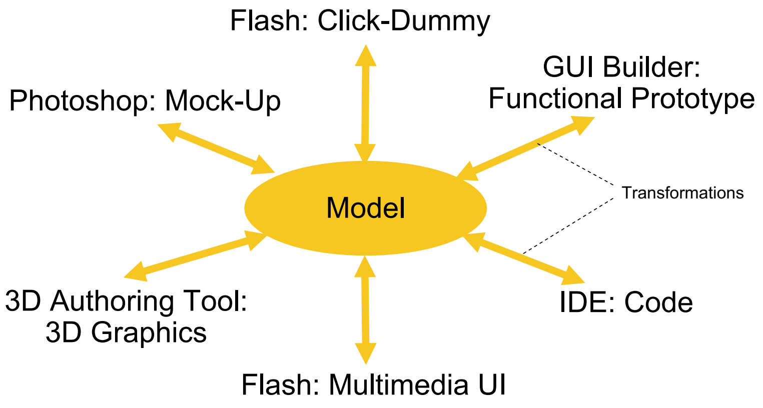


Figure 9.2: Models as central hub in the development integrating different specific development steps and tools.

and artifacts. This ensures consistency between the artifacts produced by heterogeneous tools and furthermore reduces effort. For instance, the results from previous development tasks can automatically be transformed into skeletons for subsequent tasks instead of taking them over manually.

Figure 9.2 visualizes this idea on models acting as a "central hub". The upper part shows examples for earlier development phases where prototypes play a central role in interactive systems development. For example, Photoshop mock-ups can be used to select first ideas about the system to be developed. When this step is finished, transformations are used to transmit the relevant information from the mock-ups into the model where it can be used for further development steps. A simple example for extracting information from Photoshop mock-ups is provided in the next section.

A possible subsequent step could be creating Flash click-dummies for gaining more specific user feedback. During this step, additional information about the system is added which should again be kept in a central place, i.e. in the model. Thus, it is important to allow transitions into both directions: extraction of relevant abstract information from the tools (kind of "reverse engineering") and generation of artifacts for the desired tools based on the existing model information. The ideal case would be seamless transitions between the model and different heterogeneous tools, like in Round-Trip Engineering (see sec. 7.3.1).

The lower part of figure 9.2 shows examples for later development steps, such as implementation of a final release. Here, the kinds of tools are more diverse and depend on the application domain and the target platforms. Models can be used to distribute the final implementation on different tools optimized for realizing different aspects of the system. For example, in multimedia development with Flash, it is a common practice to develop the code for system's application logic within an external programming IDE (like FDT [Powerflasher]), instead of using the Flash authoring tool's built-in code editor.

9.3 First Steps

An example for a transformation from a model into code skeletons for a specific tool (Flash) is already shown in this thesis (sec. 7). Another example for generating code skeletons for 3D authoring tools can be found in [Vitzthum08]. This section complements this with two additional examples. The first example sketches a transformation in the opposite direction: Extracting information from Photoshop mock-ups into a model. The second example is more elaborated and presents a partial application of the overall vision in a real-world user interface design project.

Extracting Information from Photoshop Photoshop is an image editing software which can be used for the very fast creation of user interface mock-ups, i.e. images to present possible ideas about the user interface to the customer or the target user group. Based on the mock-ups the most promising approaches are selected and can then be further refined using more advanced prototypes e.g. created with Flash or Java[2]

The information shown in the mock-up includes for instance the user interface elements which should be provided on a specific screen. It also contains information about the intended layout, the size of elements, the color scheme, etc. However, since the mock-ups are just images (raster graphics), this information is not stored within the mock-ups. Instead, it has to be recreated manually in later steps, e.g. when creating a click-dummy with Flash or Java.

Thus, the idea is to extract the relevant information from the mock-ups and store it in a model. For this purpose, the designers working with Photoshop have to obey a convention: Each part of the image which represents a user interface element is placed on a layer of its own. Indeed, it is a common practice to use a large number of layers when working with Photoshop, as otherwise it is not possible to move elements later. By convention, the designer has to specify a specific layer name to each layer containing a user interface element which indicates the element's name and type (for instance <name>_<type>). The possible types can be for instance taken from user interface modeling approaches.

Indeed, introducing an additional convention is to some extent a drawback. However, it is not necessarily a problem, as designers are basically used to consider conventions and the convention does not restrict the design or functionality of Photoshop.

Based on this convention, the information can be extracted as follows: Photoshop provides a built-in command "Save Layers" which causes all layers to be saved on disk in separate files. The resulting file names then correspond to the layer names which contain by convention the type and the name of the user interface element. A simple Java application then collects the file names, parses them, and creates a corresponding model, e.g. a simple kind of user interface model. Moreover, the graphics for the single user interface elements are stored separately in this way and can be directly reused.

The model is used as base for further development steps, which may include, for instance, transformations to code skeletons for a Flash click-dummy according to the mock-ups.

Application based on XAML The *Extensible Application Markup Language* (*XAML* [XAM]) from Microsoft (briefly introduced in section 4.1.1) is a specification language for concrete user interfaces. It is used for Microsoft technologies like the *Windows Presentation Foundation*. Applications with XAML user interfaces can be interpreted in a for instance in the *Silverlight* [Sil] plugin for web-browsers. It is also possible to edit and process them in *Microsoft Visual Studio* [Vis] and compile them for .NET applications.

Microsoft has recently developed two tools for user interface design which can be used to create XAML documents: *Expression Design* [Expb] is a drawing tool with similar coverage like Photoshop. *Expression Blend* [Expa] is an authoring tool for XAML user interfaces. Its functionality is, roughly spoken, similar to Flash.

Altogether, XAML and the related technologies realize already our vision to some extent: Expression Design and Expression Blend can be used for various steps in creative design. The results

[2]Please note that this is a very simple example for illustration purposes compared to a user interface design process like in [Ziegler08].

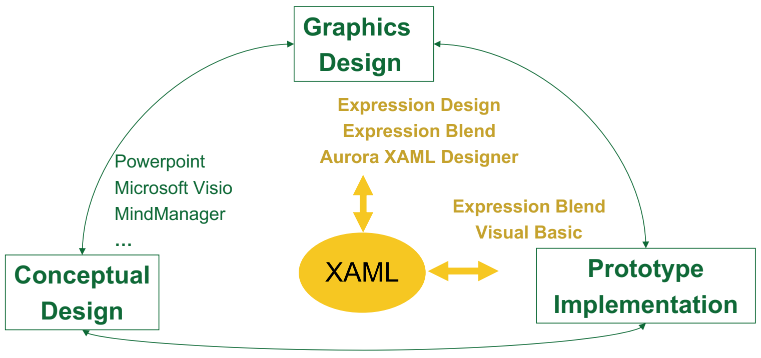


Figure 9.3: Using XAML as “central hub”.

are stored in XAML which can be used even for the final implementation with Visual Studio. This enables a seamless transition between different tools and different developers.

Such a process was applied in a real-world project in a large user interface design department in [Ziegler08]. Beside Expression Blend and Expression Design, Aurora XAML designer, a third-party tool for creating XAML user interfaces, was used. Figure 9.3 shows the resulting situation: The “Graphic Designers” perform their tasks using Expression Design. Their results are stored in XAML and can be directly used for the prototype implementation. Due to XAML as “central hub”, all subsequent changes can be automatically propagated back and forth between the different developer groups.

[Ziegler08] observed the resulting development process and interviewed the different developers about their experience. As result, she identified several problems. Many of these problems concern technical details of the tools, for instance missing functionality compared to Photoshop. As the tools are new, some of these problems might disappear in future versions.

More problematically was the observed problem, that only some the graphic designers were able to structure the XAML user interface in such a way that it can be reused for the prototype implementation. This regards for instance usage of hierarchically nested user interface container components. This means, that the advanced cooperation based on XAML would require also some learning effort on conceptual level for the graphic designers. At this point it is hard to predict how difficult this would be in practice and whether it is realistic or not.

Compared to the vision from section 9.2, XAML also has some drawbacks. First, the technology is proprietary and restricted to Windows. Second, XAML is only in the broadest sense a modeling language – it is much more a user interface specification languages. It provides only a low level of abstraction and covers, in terms of user interface modeling, at most the Concrete User Interface level. Consequently, [Ziegler08] found no way how to integrate e.g. the “Conceptual Design” into the XAML-based approach.

Summary Altogether there are still many open questions. The presented vision is certainly not much elaborated yet. Nevertheless, technologies like XAML show that this is indeed a relevant topic in industry.

XAML represents an already very comprehensive example. However, the vision as described in section 9.2 aims for a more general solution based on more profound concepts like standard-compliant modeling languages and different levels of abstraction. A proposal for a next step can be found in [Schweizer08] (supervised by the author of this thesis) which proposes to create transformations between UsiXML and XAML in both directions and provides a (very first) prototype. A transformation from UsiXML to XAML can also be found in [Martinez-Ruiz et al.06a].

Chapter 10

Summary and Conclusions

This section provides a summary, lists the contributions, discusses future work and finally draws some conclusions.

10.1 Summary

This work has presented a model-driven development approach for interactive multimedia applications. It consists of the *Multimedia Modeling Language* (*MML*) (sec. 5 and 6) and automatic transformations into code skeletons for different platforms (sec. 7 and 8.1). MML is platform independent and bases on existing modeling approaches like UML, user interface modeling approaches, and multimedia modeling (sec. 4 and 8.3.3). The resulting modeling language provides support for integration of software design, media design, *and* user interface design (sec. 6.6). In addition, it allows modeling advanced concepts of media objects, such as interactivity and dynamic alterations (sec. 5.2). Both, models and transformations, are defined in compliance to the concepts of Model-Driven Engineering (sec. 3.3 and 8.3.2).

Several model transformations exist for different target platforms. The most important target platforms addressed here are multimedia authoring tools. The authoring tool Flash was chosen as example because it is one of the most important professional platforms for multimedia application development (sec. 2.3.3). It is possible to automatically generate code skeletons from MML models which can be directly loaded and processed within the authoring tool (sec. 7.3). Thereby, the overall application structure and the relationships between its different parts are generated from the models (sec. 7.2). In contrast, for concrete media objects, user interface elements, and detailed application logic, only placeholders are generated. The placeholders are filled out in the authoring tool making use of its powerful support for visual creative design (sec. 7.4). In this way, the strengths of models (systematic structuring) and authoring tools (creative design) are both combined.

The approach presented here has been carefully validated. This includes demonstration by various implementations (sec. 6.7, 7.3, and 8.1), practical application in various projects (sec. 8.2), as well as theoretical examination from different points of view (sec. 8.3.2). In addition, it is shown that MML is the first modeling language which covers all three aspects to be modeled for an interactive multimedia application, which are application logic, interactivity, and Media Components (sec. 8.3.3). Moreover, MML is the first approach integrating existing well-established authoring tools into model-driven development.

10.2 Contributions

In summary, this thesis provides the following general conceptual research contributions:

- Demonstration of a model-driven approach for multimedia applications.
- A set of new requirements for modeling advanced interactive media objects.
- A platform independent modeling language for integrating software design, user interface design, and media design.
- Integration of multimedia authoring tools in a systematic model-driven approach.

Thereby, the thesis provides the following secondary (technical) contributions:

- A MOF-compliant and platform independent metamodel for interactive multimedia applications (sec. 5 and 6).
- An ATL transformation into Flash code skeletons (sec. 7.3 and app. C).
- A framework for structuring Flash applications (sec. 7.2).
- A MOF-compliant metamodel for Flash and ActionScript (sec. 7.1).
- An ATL transformation into Java/Piccolo code (sec. 8.1.1).
- A compact classification of multimedia applications from viewpoint of development (sec. 2.4).

10.3 Future Work

Section 9 shows some first steps into a possible area of future work: a better integration of heterogeneous tools and model-driven development. However, there are of course still many possible future steps on MML itself which are briefly discussed in the following.

Refinements on MML During the validation in section 8.3.1 two issues for future work have been identified:

- Find possible alternative notations for Media Components (beyond the icons) and evaluate them in user tests.
- Support for reuse of user interface elements like in other existing user interface modeling approaches.

In addition, several useful extensions on the different MML language parts have been discussed in section 6:

- Transformation from the Task Model to an initial Scene Model and Interaction Model ("Future Extensions" in sec. 6.3).
- Extending the Scene Model with support for Composite Scenes ("Composite Scenes" in sec. 6.3).
- Further work on the Interaction Model; its current tool support is still on a more prototypical stage compared to the other models and it has not been used extensively during the external validation yet.

Practical Application The most important future step for MML is obviously to gain more practical experience. Ideally, it should be applied in real-world projects in industrial practice. However, the most important problem is probably the missing professional tool support. Although the existing tools are sufficient to use MML in student projects, it is still far away from a really usable and stable professional tool which could be extensively used by other people on their own (i.e. without some support by the author of this thesis or good previous knowledge on the Eclipse tools). The same problems holds for many related approaches e.g. from user interface modeling domain [Clerckx et al.04].

Nevertheless, there is a rapid progress in tool support in Model-Driven Engineering in the last years. A very favorable fact is that projects like the *Eclipse Modeling Project* ([Ecld], see sec. 3.4.3 and 6.7) bundle the efforts on tool development for MDE. In the past, it often happened that research concepts were implemented by different research groups independently and from scratch. This led in the end to a variety of many incompatible alternative implementations where none of them was really elaborated enough to help in practice. In contrast, the various Eclipse projects have grown together and it has already emerged a very powerful collection of compatible and already very stable tools. The existing tools are continuously extended and refined and make development of custom modeling tools significantly easier than a few years before. Due to this pleasing situation it indeed becomes realistic that it is possible to extend and refine the tool support for MML in the next years as MML was, from start, developed in compliance to these tools.

Combination with other approaches The initial scope of MML (sec. 5.1.1) includes comprehensive support for interactive multimedia combined with the basic concepts from UML and from user interface modeling. It is thus an obvious opportunity to extend this scope towards additional concepts from user interface modeling. The MML metamodel was designed with this in mind. For instance, there is an abstract metaclass Abstract Interaction Object, analogous to other metamodels from user interface modeling area. It is possible, for instance, to extend or replace the current subclasses with those e.g. from UsiXML (sec. 4.1.2) or to add additional metaclasses for covering context-sensitiveness.

Another useful extension would be a combination with modeling approaches for Web and Rich Internet Applications, similar like e.g. the combination described in [Preciado et al.08]. Such a combination has already been discussed with one of the current contributors to UWE (sec. 4.2) and might be realized in the next time.

General Integration of Different Aspects in User Interface Modeling In general, there are various kinds of models in user interfaces modeling area, covering different aspects like multimedia, context-sensitivity, physical objects, etc. Even more aspects may arise in the future due to new interaction techniques or from the area of Ubiquitous Computing. As discussed for instance on the MDDAUI workshop in 2007 [Pleuß et al.07c], it raises the question how to manage and combine all the models for these different aspects in long-term.

One possibility is, to agree on a kind of "unified" modeling language, like the UML, which integrates all these aspects (i.e. supports multimedia *and* context *and* physical objects *and* any other possible kind of user interface). This would be mainly a matter of organization and is probably affected by many practical problems.

A second alternative is to omit a general integration and instead create over the time many different Domain Specific Languages (DSL) (see sec. 3.4.1). These DSLs can then be tailored and optimized for the kind of user interface required in a specific project or for a specific company. In such an approach, the existing (more general) metamodels from research would probably be used only as orientation or starting point but could not be directly reused.

A third alternative would be a kind of "construction kit" mechanism: metamodels for various aspects (like multimedia, context, etc.) are collected in a kind of library together with a common framework for combining them. Different researchers could contribute with metamodels supporting different or new aspects of user interfaces. The metamodels would have to follow a common structure or to provide a kind of interface so that they could be combined. Maybe the combination (which then would still be performed by modeling language specialists – not by end users) could be partially supported by predefined transformations or some kind of "metamodel weaving". A similar idea was for instance proposed for UML Profiles in their early days (see the OMG White Paper on UML Profiles in [Obj99]). Some first libraries for metamodels can be found for instance at [Atlb]. Certainly, such an approach is currently far away from a practical realization and would require a lot of future research on MDE and Language Engineering.

10.4 Conclusions

Based on the experiences and results mentioned in this thesis, Model Driven Engineering seems to be able to fulfill many of its expectations. In the author's experience, the explicit and declarative form of metamodels and transformations and the corresponding tools were helpful during the development of MML. For instance, maintenance of MML (with currently around 120 metaclasses) was relatively easy and convenient by just visually editing the metamodel and automatically generating an implementation and a tree-editor (which is sufficient for testing pruposes) with EMF (see sec. 3.4.3). Moreover, based on the existing ATL transformation for Flash, students without any previous knowledge on MDE were able to create additional transformations (Java/Piccolo, sec. 8.1.1; ActionScript 3, sec. 7.3) in short time. In the authors's opinion, models and MDE certainly have the power to become (as envisioned e.g. in [Bézivin05]) a new basic paradigm for Software Engineering.

Clearly, there are still some limitations. The external validation with students (see questionnaire on p. 180), as well as personal talks to different people from industry, has shown that – beside many positive findings – there are still software developers who are very critical against such systematic and quite "formal" methods like MDE. Such skepticism regarding new approaches from research have always occurred in Software Engineering – e.g. object-orientation was introduced. On the other hand, it can be learned the past, that it is certainly useful to prevent overrated expectations. New paradigms can never be the only "silver bullet" solving all existing problems – there are still some areas where e.g. object-orientation plays no role or where e.g. procedural programming is sufficient. Likewise, there will always be projects and situations where MDE is not sufficient or, at least, must be combined with other approaches.

Certainly, it is of great importance for MDE, that it is not applied in a too isolated and dogmatic way. Instead, bridges have to be build to other areas so that the strengths of MDE are emphasized while its limitations are compensated by other solutions. For instance, good user interface design is essential for an application's quality. It cannot be the solution for MDE to neglect creative design or just to discard the established existing methods and tools for user interface design. Instead, it is necessary to build a bridge to this area. This thesis has shown through very concrete examples a possible way for such a bridge – towards a better integration of Model-Driven Engineering and creative design.

Appendix A

Multimedia Taxonomy by Hannington and Reed

The next page shows the facets and their values of the taxonomy for multimedia applications taken from Hannington and Reed [Hannington and Reed02].

Listing of Multimedia Taxonomy by Hannington and Reed

Domain Facet
- MM Business Systems
  - Electronic - commerce
  - Marketing
  - Video Brochures
  - Virtual Shopping
- MM Communication Systems
  - Computer-supported - collaborative work
  - MM teleservices
- MM Educational Systems
  - Automatic testing
  - Distance learning
  - Flexible teaching - materials
  - Simulation systems
- MM Entertainment Systems
  - Infotainment
  - Multiplayer - network games
  - 3D computer - games
- MM Information Systems

Solution Space facet
- Databases
- Electronic books
- Electronic magazines
- Hypertexts
- Information kiosks
- Interactive art and performance
- Interactive music
- Multimedia expert system
- Multimedia presentation
- Streaming media
- Videoconferencing
- …

Delivery Platform Facet
- Online
  - Intranet
  - Internet
- Offline
  - CD-ROM
  - Hard-disk - installation
- Hybrid
  - Online/Offline

Security Facet
- Access levels
- Authorization
- Authentication
  - Digital signatures
  - Time stamping
- File privileges
- Firewall type
- Privacy
  - Algorithm
  - Encryption
  - Key system
- Password storage
- System managed locally
- System managed globally - (remotely)
- E-commerce
  - Transaction - security
  - Secure payment - processing
- …

Navigation Facet
- Linear
- Non-linear
- Hierarchical
- Composite
  - Non-linear/linear
  - Non-linear / hierarchical
  - Hierarchical/linear

Interactivity Facet (based on [44])
- Passive
- Reactive
- Proactive
- Directive

Interface Facet
- Widget
  - Menu
    - Level
  - Button
  - Check box
  - Text box
  - List box
  - Dialog box
  - Slider
  - Form
  - …

Programming Requirements Facet
- Static Web page
- Database
  - Retrieval/storage
  - Retrieval only - (data warehouse)
- Information processing
  - Forms (Web)
- Scripting
  - Client side
    - javascript
    - …
  - Server side
    - php
    - …
- Expert system
- Interface for pre-existing software
- Legacy system
- …

Media Facet
- Static
  - Text
  - Graphics
  - Photographs
- Temporal
  - Animation
  - Audio
    - Music
    - Voice
    - Sound effect
  - Video

Origin Facet
- Acquired
- Repurposed
- Created

State Facet
- Completed
- Demo voice
- Partially rendered
- Sample track
- Space filler
- …

Duration Facet – unit of measure and classification of particular durations into categories would need to be defined by the classifier
- Long
- Medium
- Short

Size Facet - unit of measure and classification of particular sizes into categories would need to be defined by the classifier. I.e. what might be regarded as small when working on a project with CD-ROM as the delivery medium, would be different from when developing for a hand-held.
- Large
- Medium
- Small

Format Facet
- Gif
- Jpeg
- Mpeg
- Pdf
- Plain text
- Post script
- Word
- …

Operations Facet – operations performed on media artefact – reflective of development phases
- Concept and planning
- Design
- Production
- Testing
- …

Design Technique/Artefact Facet
- Mind map
- Information hierarchy
- Content map
- Navigation chart
- Flowchart
- Prototype
- Storyboard
- Interactive storyboard
- Storybook
- Script
- HDM
- OOHDM
- RMDM
- …

Authoring Tools Facet
- Commercial
  - Adobe Photoshop
  - Authorware
  - Corel Draw
  - Dreamweaver
  - Flash
  - Macromedia - Director
  - Netscape - Composer
  - Pro Tools
  - Sound Designer
  - Toolbook
- …
- Research
  - DEMAIS
  - DENIM
  - …

Skills Facet
- Actor
- Animator
- Content expert
- Editor
- Graphic artist
- Musicians
- Photographer
- Project manager
- Programmer
- Researcher
- Sound/audio engineer
- Sound designer
- Tester
- Testing supervisor
- Videographer
- Video editor
- Writer
- …

Marginal Subjects
Instructional Design Facets
(based on [18])

Instructional Design Model Facet
- Tutorials
- Drills
- Practice programs
- Simulations
- Instructional games
- Didactic presentations
- Explorations
- Structured
- Observations
  - Simulated Personal - Interactions

Instruction Phase Facet
- Present
- Guide
- Practice
- Assess

Instructional Assessment Facet
- Demonstration /performance tests
- Problem solving tests
- Recall tests
  - Fill-in-the-blank
  - Short-answer
  - Essay
- Recognition tests
  - True-false
  - Multiple-choice

Figure A.1: factes and possible values for the taxonomy for multimedia applications taken from [Hannington and Reed02].

Appendix B

ActionScript Class MovieClip – Documentation

B.1 Properties

Property	**Description**
`_alpha:Number`	The alpha transparency value of the movie clip.
`blendMode:Object`	The blend mode for this movie clip.
`cacheAsBitmap:Boolean`	If set to true, Flash Player caches an internal bitmap representation of the movie clip.
`_currentframe:Number [read-only]`	Returns the number of the frame in which the playhead is located in the movie clip's timeline.
`_droptarget:String [read-only]`	Returns the absolute path in slash-syntax notation of the movie clip instance on which this movie clip was dropped.
`enabled:Boolean`	A Boolean value that indicates whether a movie clip is enabled.
`filters:Array`	An indexed array containing each filter object currently associated with the movie clip.
`focusEnabled:Boolean`	If the value is undefined or false, a movie clip cannot receive input focus unless it is a button.
`_focusrect:Boolean`	A Boolean value that specifies whether a movie clip has a yellow rectangle around it when it has keyboard focus.
`_framesloaded:Number [read-only]`	The number of frames that are loaded from a streaming SWF file.
`_height:Number`	The height of the movie clip, in pixels.
`_highquality:Number`	Deprecated since Flash Player 7. This property was deprecated in favor of MovieClip._quality.
	Specifies the level of anti-aliasing applied to the current SWF file.

`hitArea:Object`	Designates another movie clip to serve as the hit area for a movie clip.
`_lockroot:Boolean`	A Boolean value that specifies what _root refers to when a SWF file is loaded into a movie clip.
`menu:ContextMenu`	Associates the specified ContextMenu object with the movie clip.
`_name:String`	The instance name of the movie clip.
`opaqueBackground:Number`	The color of the movie clip's opaque (not transparent) background of the color specified by the number (an RGB hexadecimal value).
`_parent:MovieClip`	A reference to the movie clip or object that contains the current movie clip or object.
`_quality:String`	Sets or retrieves the rendering quality used for a SWF file.
`_rotation:Number`	Specifies the rotation of the movie clip, in degrees, from its original orientation.
`scale9Grid:Rectangle`	The rectangular region that defines the nine scaling regions for the movie clip.
`scrollRect:Object`	The scrollRect property allows you to quickly scroll movie clip content and have a window viewing larger content.
`_soundbuftime:Number`	Specifies the number of seconds a sound prebuffers before it starts to stream.
`tabChildren:Boolean`	Determines whether the children of a movie clip are included in the automatic tab ordering.
`tabEnabled:Boolean`	Specifies whether the movie clip is included in automatic tab ordering.
`tabIndex:Number`	Lets you customize the tab ordering of objects in a movie.
`_target:String [read-only]`	Returns the target path of the movie clip instance, in slash notation.
`_totalframes:Number [read-only]`	Returns the total number of frames in the movie clip instance specified in the MovieClip parameter.
`trackAsMenu:Boolean`	A Boolean value that indicates whether other buttons or movie clips can receive mouse release events.
`transform:Transform`	An object with properties pertaining to a movie clip's matrix, color transform, and pixel bounds.
`_url:String [read-only]`	Retrieves the URL of the SWF, JPEG, GIF, or PNG file from which the movie clip was downloaded.
`useHandCursor:Boolean`	A Boolean value that indicates whether the pointing hand (hand cursor) appears when the mouse rolls over a movie clip.
`_visible:Boolean`	A Boolean value that indicates whether the movie clip is visible.
`_width:Number`	The width of the movie clip, in pixels.

`_x:Number`	An integer that sets the x coordinate of a movie clip relative to the local coordinates of the parent movie clip.
`_xmouse:Number [read-only]`	Returns the x coordinate of the mouse position.
`_xscale:Number`	Determines the horizontal scale (percentage) of the movie clip as applied from the registration point of the movie clip.
`_y:Number`	Sets the y coordinate of a movie clip relative to the local coordinates of the parent movie clip.
`_ymouse:Number [read-only]`	Indicates the y coordinate of the mouse position.
`_yscale:Number`	Sets the vertical scale (percentage) of the movie clip as applied from the registration point of the movie clip.

B.2 Properties Inherited from Class Object

`constructor:Object`	Reference to the constructor function for a given object instance.
`__proto__:Object`	Refers to the prototype property of the class (ActionScript 2.0) or constructor function (ActionScript 1.0) used to create the object.
`prototype:Object [static]`	A reference to the superclass of a class or function object.
`__resolve:Object`	A reference to a user-defined function that is invoked if ActionScript code refers to an undefined property or method.

B.3 Operations

`Signature`	Description
`attachAudio(id:Object) :Void`	Specifies the audio source to be played.
`attachBitmap(bmp :BitmapData, depth:Number, [pixelSnapping:String], [smoothing:Boolean]):Void`	Attaches a bitmap image to a movie clip.
`attachMovie(id:String, name:String, depth:Number, [initObject:Object]) :MovieClip`	Takes a symbol from the library and attaches it to the movie clip.
`beginBitmapFill(bmp :BitmapData, [matrix :Matrix], [repeat :Boolean], [smoothing :Boolean]):Void`	Fills a drawing area with a bitmap image.

`beginFill(rgb:Number, [alpha:Number]):Void`	Indicates the beginning of a new drawing path.
`beginGradientFill(fillType :String, colors:Array, alphas:Array, ratios :Array, matrix:Object, [spreadMethod:String], [interpolationMethod :String], [focalPointRatio :Number]):Void`	Indicates the beginning of a new drawing path.
`clear():Void`	Removes all the graphics created during runtime by using the movie clip draw methods, including line styles specified with MovieClip.lineStyle().
`createEmptyMovieClip(name :String, depth:Number) :MovieClip`	Creates an empty movie clip as a child of an existing movie clip.
`createTextField (instanceName:String, depth:Number, x:Number, y:Number, width:Number, height:Number):TextField`	Creates a new, empty text field as a child of the movie clip on which you call this method.
`curveTo(controlX:Number, controlY:Number, anchorX :Number, anchorY:Number) :Void`	Draws a curve using the current line style from the current drawing position to (anchorX, anchorY) using the control point that ((controlX, controlY) specifies.
`duplicateMovieClip(name :String, depth:Number, [initObject:Object]) :MovieClip`	Creates an instance of the specified movie clip while the SWF file is playing.
`endFill():Void`	Applies a fill to the lines and curves that were since the last call to beginFill() or beginGradientFill().
`getBounds(bounds:Object) :Object`	Returns properties that are the minimum and maximum x and y coordinate values of the movie clip, based on the bounds parameter.
`getBytesLoaded():Number`	Returns the number of bytes that have already loaded (streamed) for the movie clip.
`getBytesTotal():Number`	Returns the size, in bytes, of the movie clip.
`getDepth():Number`	Returns the depth of the movie clip instance.
`getInstanceAtDepth(depth :Number):MovieClip`	Determines if a particular depth is already occupied by a movie clip.
`getNextHighestDepth() :Number`	Determines a depth value that you can pass to MovieClip.attachMovie(), MovieClip.duplicateMovieClip(), or MovieClip.createEmptyMovieClip() to ensure that Flash renders the movie clip in front of all other objects on the same level and layer in the current movie clip.

`getRect(bounds:Object):Object`	Returns properties that are the minimum and maximum x and y coordinate values of the movie clip, based on the bounds parameter, excluding any strokes on shapes.
`getSWFVersion():Number`	Returns an integer that indicates the Flash Player version for the movie clip was published.
`getTextSnapshot():TextSnapshot`	Returns a TextSnapshot object that contains the text in all the static text fields in the specified movie clip; text in child movie clips is not included.
`getURL(url:String, [window:String], [method:String]):Void`	Loads a document from the specified URL into the specified window.
`globalToLocal(pt:Object):Void`	Converts the pt object from Stage (global) coordinates to the movie clip's (local) coordinates.
`gotoAndPlay(frame:Object):Void`	Starts playing the SWF file at the specified frame.
`gotoAndStop(frame:Object):Void`	Brings the playhead to the specified frame of the movie clip and stops it there.
`hitTest():Boolean`	Evaluates the movie clip to see if it overlaps or intersects with the hit area that the target or x and y coordinate parameters identify.
`lineGradientStyle(fillType:String, colors:Array, alphas:Array, ratios:Array, matrix:Object, [spreadMethod:String], [interpolationMethod:String], [focalPointRatio:Number]):Void`	Specifies a line style that Flash uses for subsequent calls to the lineTo() and curveTo() methods until you call the lineStyle() method or the lineGradientStyle() method with different parameters.
`lineStyle(thickness:Number, rgb:Number, alpha:Number, pixelHinting:Boolean, noScale:String, capsStyle:String, jointStyle:String, miterLimit:Number):Void`	Specifies a line style that Flash uses for subsequent calls to the lineTo() and curveTo() methods until you call the lineStyle() method with different parameters.
`lineTo(x:Number, y:Number):Void`	Draws a line using the current line style from the current drawing position to (x, y); the current drawing position is then set to (x, y).
`loadMovie(url:String, [method:String]):Void`	Loads a SWF, JPEG, GIF, or PNG file into a movie clip in Flash Player while the original SWF file is playing.
`loadVariables(url:String, [method:String]):Void`	Reads data from an external file and sets the values for variables in the movie clip.
`localToGlobal(pt:Object):Void`	Converts the pt object from the movie clip's (local) coordinates to the Stage (global) coordinates.
`moveTo(x:Number, y:Number):Void`	Moves the current drawing position to (x, y).

`nextFrame():Void`	Sends the playhead to the next frame and stops it.
`play():Void`	Moves the playhead in the timeline of the movie clip.
`prevFrame():Void`	Sends the playhead to the previous frame and stops it.
`removeMovieClip():Void`	Removes a movie clip instance created with duplicateMovieClip(), MovieClip.duplicateMovieClip(), MovieClip.createEmptyMovieClip(), or MovieClip.attachMovie().
`setMask(mc:Object):Void`	Makes the movie clip in the parameter mc a mask that reveals the calling movie clip.
`startDrag([lockCenter :Boolean], [left:Number], [top:Number], [right :Number], [bottom:Number]) :Void`	Lets the user drag the specified movie clip.
`stop():Void`	Stops the movie clip that is currently playing.
`stopDrag():Void`	Ends a MovieClip.startDrag() method.
`swapDepths(target:Object) :Void`	Swaps the stacking, or depth level (z-order), of this movie clip with the movie clip that is specified by the target parameter, or with the movie clip that currently occupies the depth level that is specified in the target parameter.
`unloadMovie():Void`	Removes the contents of a movie clip instance.

B.4 Event Handling Operations

`Event`	Description
`onData():Void`	Invoked when a movie clip receives data from a MovieClip.loadVariables() call or a MovieClip.loadMovie() call.
`onDragOut():Void`	Invoked when the mouse button is pressed and the pointer rolls outside the object.
`onDragOver():Void`	Invoked when the pointer is dragged outside and then over the movie clip.
`onEnterFrame():Void`	Invoked repeatedly at the frame rate of the SWF file.
`onKeyDown():Void`	Invoked when a movie clip has input focus and user presses a key.
`onKeyUp():Void`	Invoked when a key is released.
`onKillFocus(newFocus :Object):Void`	Invoked when a movie clip loses keyboard focus.
`onLoad():Void`	Invoked when the movie clip is instantiated and appears in the timeline.
`onMouseDown():Void`	Invoked when the mouse button is pressed.
`onMouseMove():Void`	Invoked when the mouse moves.

`onMouseUp():Void`	Invoked when the mouse button is released.
`onPress():Void`	Invoked when the user clicks the mouse while the pointer is over a movie clip.
`onRelease():Void`	Invoked when a user releases the mouse button over a movie clip.
`onReleaseOutside():Void`	Invoked after a user presses the mouse button inside the movie clip area and then releases it outside the movie clip area.
`onRollOut():Void`	Invoked when a user moves the pointer outside a movie clip area.
`onRollOver():Void`	Invoked when user moves the pointer over a movie clip area.
`onSetFocus(oldFocus :Object):Void`	Invoked when a movie clip receives keyboard focus.
`onUnload():Void`	Invoked in the first frame after the movie clip is removed from the Timeline.

Appendix C

Transformation from MML to Flash Model

Main:

1. For each application, a new main folder is generated named by the application. It contains a folder media and a folder model. (For the ActionScript classes a folder corresponds to a package).

2. A folder util is generated where some library classes and interfaces are copied into.

3. A Flash Document is generated with the name of the application

4. An ActionScript class is generated

Domain Classes:

5. A Domain Class is mapped to an ActionScript class file in the folder model. For each class:

   - A property is mapped to an ActionScript class property. In the current implementation, associations are simply mapped to class attributes as well. But this could easily be extended with more advanced mappings analogous to mappings from UML class diagrams to Java code (see e.g. discussion in [Génova et al.03]).
   - An operation is mapped to an operation signature in ActionScript. The developer has to fill out the operation body manually.
   - Generalizations, etc. are mapped to the corresponding ActionScript constructs (analogous to mappings from UML to e.g. Java code)

6. All Domain Classes inherit from the library class Observable.

Media Components:

7. A Media Components is mapped to a file in the folder media containing a simple placeholder to be replaced by the media designer. The filename corresponds to the Media Component's name. Its type depends on the media type:

- A 2D animation or a graphic is mapped to a FLA file containing a placeholder MovieClip. The placeholder can be a simple rectangle shape with text inside showing the Media Component's name.
- An image is mapped to a *JPEG* file containing a dummy image.
- Sound is mapped to an *MP3* file containing a short dummy sound.
- A video is mapped to a *FLV* file (the Flash-specific video format) containing a short dummy video. For each instantiation an instance of a Flash video player component (FLVPlayback) is generated refering to the video.
- Text is mapped to a text file containing dummy text in a restricted HTML format supported by Flash. For each instantiation an instance of a Flash *Text Area* component is generated. Code is generated into the attached ActionScript class (rule 8) which automatically imports the text file into the Text Area when the Text Arae is loaded.
- 3D animations are currently not supported by Flash but basically a 3D animation can be mapped to e.g. a VRML or X3D file analogous to SSIML [Vitzthum08].

8. In addition, an ActionScript class is created which may provide general operations and properties of the media component. It is generated in the folder media and is automatically attached to the placeholder, e.g. the MovieClip in case of an animation.

9. Properties and Operations from Interfaces in MML are mapped to properties and operations in the ActionScript class from rule 8.

10. If a Media Component is manifested by Media Artifacts (sec. 5.2.5) then a subfolder of the folder media is generated named after the Media Component and containing a file for each Media Artifact.

11. Inner Structure of Media Components is defined by Media Parts, Part Artifacts, and Inner Properties. Their mapping depends on the media type. As only animations and graphics are created within Flash itself, the inner structure is only fully supported for these two media types as example:

    - For 2D animations and graphics a SubAnimation2D, Transformation2D or SubGraphics is mapped to a MovieClip in the library within the FLA file for the owning Media Component. Thereby, one MovieClip is generated for each Part Artifact. As Flash requires that each item in the library has a unique name (e.g. if different cars are imported into the same Scene) the name of the owning Media Component is added as prefix to the PartArtifact's name in the library (e.g. "FerrariFrontWheel" instead of "FrontWheel"). Each Inner Property is then mapped to an instance on the Stage of its parent MovieClip, i.e. the hierarchy of inner properties is mapped to a hierarchy of MovieClips.
    - Other media types are only supported partially:
      11.1. For Video and Images a VideoRegion or ImageRegion is mapped to a transparent MovieClip. Transparent MovieClips are visible in the authoring tool and can there easily be placed and reshaped by the developer to visually define a region. It is then possible e.g. to add a event listener to the transparent MovieClip which listens for instance for mouse clicks. In this way it is easy to define e.g. a specific region of an image which is sensitive to mouse clicks.
      11.2. Regarding Audio, Flash supports only basic functionality. Basically, an AudioChannel would be mapped to a Channel in a Midi Sequencer or to a track in a audio editor.

In Flash itself it is possible to distinguish between the two stereo channels and modify the panning. It is also possible to play synchronize multiple pieces of audio in the timeline.

11.3. 3D animation is not supported by Flash. Basically, the inner structure of 3D animations can be mapped to *VRML* or *X3D* code analogous to SSIML [Vitzthum and Pleuß05]. An Object3D is mapped to an external file. Each instance corresponds to an *Inline Node* referring to this file. Transformation3D, Camera3D, Light3D, and Viewpoint3D are mapped to corresponding VRML/X3D nodes.

11.4. A Text Region can be mapped to a text file. For each Inner Property code is generated which adds the Text Region at runtime to Text Areas.

12. If a Media Component has different variants (see sec. 5.2.10) then a file is generated for each value combination. The Variation Type values of each file are attached as suffixes to the filename, separated by '_' (e.g. video1_english_low, video1_english_high, video1_german_low, video1_german_high, etc.).

13. Note that Media Representations are not explicitly reflected in the generated code as the implementation proposed here uses only the Observer pattern between Domain Classes and AIOs to manipulate user interface elements.

Scenes:

14. A scene from the MML model is mapped to a FLA file (representing the Presentation Unit) and an ActionScript class. Both are located in the top-level folder. The ActionScript class for the Scene complies to the structure described in section 7.2.2, i.e. it applies the Singleton design pattern for accessing Scene, provides a loading mechanism, and is dynamically attached to the content of the FLA file (the Presentation Unit).

15. The Application Start is mapped to a static operation main in the main ActionScript class from rule 4 and to script code in the first frame of the timeline in the main Flash Document from rule 3 which triggers the operation main. (The operation main is provided for compatibility with the popular external third-party ActionScript compiler [Motion-Twin]).

16. An Entry Operation is mapped to an operation of its Scene's ActionScript class named with the prefix entry_.

17. An Exit Operation is mapped to an operation with the prefix exit_.

18. A Transition from the Scene Model is mapped to code in the operation body of the corresponding Exit Operation (resp. the operation main from rule 15). The code loads the target Scene and afterwards invokes the target Entry Operation analogous to listing 7.5.

19. For each Scene class an operation init is generated where code to initialize contained user interface elements is generated into. For example output components are initialized as observer (with addObserver()) by default (see rule 25)

20. For each Scene a folder (package) is generated which is used to store the ActionScript classes for contained user interface elements (see rule 22)

AIOs:

21. An AIO is mapped to a Movie Clip and an instance on the stage. The Movie Clip acts as container and has content depending on the AIO type:

    - For AIOs realized by Media Instances the container Movie Clip contains these Media Instances. Therefore, a library item is generated in the Scene's FLA file which refers to the Media Component from the folder media (see 7.2.3).
    - For AIOs not realized by a Media Instance the container Movie Clip contains one or more widget instances according to the AIO type. For instance, for an Action Component an instance of a button named with the name of the AIO while for an Input Component a text label showing the name of the AIO and a text input field next to it.

22. For each AIO an ActionScript class is generated in the Scene's folder which is attached to the container Movie Clip. This class is used for event handling operations ("Controller" in terms of MVC). The default code generated into this class initializes a default event listener depending on the widget type. For instance, for buttons a click event listener and for text input fields an enter event listener. The developer has to complete the body of the event handling operation manually, except for Action Components referring to an operation – in this case the operation call is generated automatically.

23. The instance on the stage is defined as property of the Scene class (to be directly accessible in the class, see 7.1.3).

24. A domain object represented by an AIO is mapped to a class property in the Scene class.

25. A UI Representation between a domain object and an AIO from type Output Component or Edit Component is mapped to code for the Observer pattern:

    - The ActionScript class for the AIO implements the interface observer and provides an operation update
    - A line of code is added to the operation init in the Scene class which adds the AIO class as observer to the domain object. (The domain object is available as property in the Scene class and all Domain classes are defined as Observable, see above).
    - Sensors are mapped to corresponding ActionScript code in the Scene class and attached to the Media Instance they belong to. If a Sensor is owned by a Media Component (instead of a single instance) then in Flash the sensor code is attached to all Media Instances. For example a Collision Sensor is mapped to an onEnterFrame operation in the Scene class. This operation is inherited from MovieClip and is called every time the Scene enters a new frame. Within onEnterFrame the operation hitTest from class MovieClip is used to find out whether a collision appeared.

Interaction:

26. The information from the Interaction Model is mapped to code within the corresponding Entry Operation. The mapping mainly corresponds to general rules for mapping UML Activity Diagrams to source code. A discussion and a first prototypical implementation based on existing algorithms has been discussed and implemented in the Diploma Thesis by Shaykhit [Shaykhit07] supervised by the author of this work.

Appendix D

Questionnaire

The next four page show the questionnaire on MML which has been filled out by 27 participants after the course “Multimedia-Programming” in 2006.

I. Programmier-Vorkenntnisse:

Mit Flash/ActionScript hatte ich vor Beginn der Lehrveranstaltung folgende Vorerfahrungen:

	Professionelle Projekte	Hobby-Projekte	Grundkenntnisse	Keine Kenntnisse
Java	O	O	O	O

Mit anderen Programmiersprachen habe ich folgende Erfahrungen:

	Professionelle Projekte	Hobby-Projekte	Grundkenntnisse	Keine Kenntnisse
Java	O	O	O	O
C/C++	O	O	O	O
JavaScript	O	O	O	O
PHP	O	O	O	O
__________	O	O	O	O
__________	O	O	O	O

Mit anderen Multimedia-Autorenwerkzeugen habe ich folgende Erfahrungen:

	Professionelle Projekte	Hobby-Projekte	Grundkenntnisse	Keine Kenntnisse
Director	O	O	O	O
Toolbook	O	O	O	O
__________	O	O	O	O
__________	O	O	O	O

Vor Beginn der Lehrveranstaltung hatte ich folgende Vorerfahrung mit graphischen Modellierungssprachen wie UML oder MML:

	Ich habe Modelle für eigene Projekte erstellt	Ich habe Beispiel-Modelle (z.B. aus Übungsaufgaben) erstellt.	Ich habe davon gehört oder gelesen	Keine Kenntnisse
UML	O	O	O	O
MML	O	O	O	O
__________	O	O	O	O

	Ja	Nein
Ich verfüge über grundlegende Kenntnisse im Bereich der Softwaretechnik (Software-Engineering).	O	O
Ich habe bereits im Rahmen des Softwarepraktikums ein Spiel programmiert.	O	O

Fachsemester: (Zahl)

Studiengang: (Bitte angeben)

II. Modellierung mit MML:

	Ich stimme voll zu				Ich stimme nicht zu
Bei der Entwicklung von Software ist eine Entwurfsphase (d.h. die Erstellung eines Entwurfs der Anwendung vor Beginn der Implementierung) generell wichtig.	○	○	○	○	○
Die Verwendung von MML ist für den Entwurf von Multimedia-Anwendungen sinnvoll.	○	○	○	○	○
Der Einarbeitungsaufwand in MML ist gering.	○	○	○	○	○
Für das MMP-Projekt wurde in unserem Team MML verwendet.	○	○	○	○	○
Ich habe mich persönlich mit MML beschäftigt.	○	○	○	○	○
Falls verwendet: Der Einsatz von MML war für unser MMP-Projekt hilfreich.	○	○	○	○	○
Falls verwendet: Die Modellierung hat nur einen kleinen Teil der Gesamtzeit unseres Projektes benötigt.	○	○	○	○	○
Der Einsatz von MML ist für große, professionelle Projekte sinnvoll.	○	○	○	○	○
Der Einsatz von MML ist für kleinere und mittelgroße Projekte sinnvoll.	○	○	○	○	○

	ja	nein
Ich würde MML zukünftig in eigenen Multimedia-Projekten einsetzen.	○	○

Ich habe ……… Prozent der Zeit des Gesamtprojekts zur Einarbeitung in MML benötigt.
(bitte Zahl einsetzen)

Ich habe …….. Prozent der Zeit des Gesamtprojekts für die Modellierung mit MML benötigt.
(bitte Zahl einsetzen)

Vorteile von MML:

Nachteile von MML:

Sonstige Kommentare zu MML:

III. Verwendung von MagicDraw für MML:

	Ich stimme voll zu				Ich stimme nicht zu
Ich habe bereits mit UML-Werkzeugen gearbeitet.	O	O	O	O	O
Der Einarbeitungsaufwand in MagicDraw war gering.	O	O	O	O	O
Für die gesamte Modellierung habe ich MagicDraw verwendet.	O	O	O	O	O
Falls verwendet: MagicDraw war gut benutzbar.	O	O	O	O	O

<u>Vorteile</u> von MagicDraw:

<u>Nachteile</u> von MagicDraw:

Sonstige Kommentare zu MagicDraw:

IV. Ableitung von Codegerüsten aus den MML-Modellen:

Anmerkung: im MMP-Projekt wurde ein Code-Gerüst zur Verfügung gestellt („ExampleProjekt"), zusammen mit einer Anleitung, wie man ein MML-Modell in ein Codegerüst (entsprechend der Vorlage) abbilden kann. Ein Codegenerator, der diese Abbildung des Modells in FLA-Dateien und ActionScript-Code auch automatisch ausführen kann, befindet sich in der Erstellung.

Bezogen auf das Codegerüst („ExampleProject"), das Ihnen im MMP-Projekt zur Verfügung gestellt wurde:

	Ich stimme voll zu				Ich stimme nicht zu
Der Einarbeitungsaufwand in das Codegerüst war gering.	O	O	O	O	O
Unser Team hat das Codegerüst verwendet.	O	O	O	O	O
Falls verwendet: Die Qualität des Codes im Codegerüst ist hoch.	O	O	O	O	O
Die Verwendung von (generierten) Codegerüsten ist in Multimedia-Projekten grundsätzlich sinnvoll.	O	O	O	O	O

Stellen Sie sich vor, der Code würde automatisch aus den MML-Modellen generiert. Wie schätzen Sie den Gesamtansatz (Verwendung von MML und automatische Codegenerierung aus den Modellen) ein **im Vergleich** zu „herkömmlichen“ Vorgehen (d.h. Vorgehen ohne Entwurfsphase)?

	viel besser		gleich		viel schlechter
Wartbarkeit, d.h. die Anwendung ist gut strukturiert und kann nachher einfach geändert oder erweitert werden	O	O	O	O	O
Plattformunabhängigkeit, d.h. die Anwendung kann einfach für verschiedene Plattformen (z.B. unterschiedliche Geräte) implementiert werden.	O	O	O	O	O
Aufwand für die Entwicklung der Anwendung.	O	O	O	O	O
Möglichkeit zur Einbindung von Expertenwissen in den Code (z.B. der Code entspricht den Regeln, um eine gute Performance der Anwendung zu erreichen).	O	O	O	O	O
Generelle Eignung zur Erstellung großer professioneller Anwendungen.	O	O	O	O	O
Generelle Eignung zur Erstellung kleinerer und mittelgroßer Anwendungen.	O	O	O	O	O
Ihre persönliche Gesamtwertung.	O	O	O	O	O

Sonstige wichtige <u>Vorteile</u> des Ansatzes (Modellierung + Codegenerierung):

Sonstige wichtige <u>Nachteile</u> des Ansatzes (Modellierung + Codegenerierung):

Sonstige Kommentare:

Bibliography

[3DS] "Autodesk 3ds Max," [Website]. URL: http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=5659302

[Abi-Antoun et al.08] M. Abi-Antoun, J. Aldrich, N. Nahas, B. Schmerl, and D. Garlan, "Differencing and merging of architectural views," *Automated Software Engg.*, vol. 15, no. 1, pp. 35–74, 2008.

[Abouzahra et al.05] A. Abouzahra, J. Bézivin, M. D. D. Fabro, and F. Jouault, "A Practical Approach to Bridging Domain Specific Languages with UML profiles," in *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA'05*, San Diego, California, USA, 2005. URL: http://www.softmetaware.com/oopsla2005/bezivin1.pdf

[Abrams and Helms04] M. Abrams and J. Helms, *User Interface Markup Language (UIML) Specification*, 3rd ed., UIML.org, March 2004, wd-UIML-UIMLspecification-3.1. URL: http://www.oasis-open.org/committees/download.php/5937/uiml-core-3.1-dra%20ft-01-20040311.pdf

[Abrams et al.99] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster, "UIML: an appliance-independent XML user interface language," in *WWW '99: Proceedings of the eighth international conference on World Wide Web*. New York, NY, USA: Elsevier North-Holland, Inc., 1999, pp. 1695–1708.

[Acerbis et al.07] R. Acerbis, A. Bongio, M. Brambilla, and S. Butti, "WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications," in *Web Engineering, 7th International Conference, ICWE 2007, Como, Italy, July 16-20, 2007, Proceedings*, ser. Lecture Notes in Computer Science, L. Baresi, P. Fraternali, and G.-J. Houben, Eds., vol. 4607. Springer, 2007, pp. 501–505.

[Ado] "Adobe," [Website]. URL: www.adobe.com

[Ado05a] *Flash 8 – ActionScript 2.0 Language Reference*, Adobe, San Francisco, CA, 9 2005. URL: http://download.macromedia.com/pub/documentation/en/flash/fl8/fl8_as2lr.pdf

[Ado05b] *Flash 8 – Components Language Reference*, Adobe, San Francisco, CA, 9 2005. URL: http://download.macromedia.com/pub/documentation/en/flash/fl8/fl8_clr.pdf

[Ado05c] *Flash 8 – Extending Flash*, Adobe, San Francisco, CA, 9 2005. URL: http://download.macromedia.com/pub/documentation/en/flash/fl8/fl8_extending.pdf

[Ado08] *SWF File Format Specification*, 10th ed., Adobe, 11 2008. URL: http://www.adobe.com/devnet/swf/pdf/swf_file_format_spec_v10.pdf

[Adobea] Adobe, “Cairngorm,” [Website]. URL: http://opensource.adobe.com/wiki/display/cairngorm/Cairngorm;jsessionid=6CABDBBBE2CB65747462A07D0CBF0E05

[Adobeb] Adobe, “Flash – Supported Devices,” [Website]. URL: http://www.adobe.com/mobile/supported_devices/

[Adobec] Adobe, “Flash 8 Livedocs [Complete Online Documentation],” [Website]. URL: http://livedocs.adobe.com/flash/8/main/wwhelp/wwhimpl/js/html/wwhelp.htm

[Adobed] Adobe, “Flash Player Software,” [Website]. URL: http://www.adobe.com/de/products/flashplayer/

[Adobee] Adobe, “Flash Player Statistics,” [Website]. URL: http://www.adobe.com/products/player_census/flashplayer/

[Aedo and Díaz01] I. Aedo and P. Díaz, “Applying software engineering methods for hypermedia systems,” in *ITiCSE ’01: Proceedings of the 6th annual conference on Innovation and technology in computer science education*. New York, NY, USA: ACM, 2001, pp. 5–8.

[AGG] “The AGG Homepage,” [Website]. URL: http://tfs.cs.tu-berlin.de/agg/

[Alanen and Porres08] M. Alanen and I. Porres, “A metamodeling language supporting subset and union properties,” *Software and System Modeling*, vol. 7, no. 1, pp. 103–124, 2008.

[Aleem98] T. A. Aleem, “A Taxonomy of Multimedia Interactivity,” Ph.D. dissertation, The Union Institute, USA, 1998.

[Allen83] J. F. Allen, “Maintaining knowledge about temporal intervals,” *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[Amelunxen and Schürr06] C. Amelunxen and A. Schürr, “Vervollständigung des Constraint-basierten Assoziationskonzeptes von UML 2.0,” in *Modellierung*, ser. LNI, H. C. Mayr and R. Breu, Eds., vol. 82. GI, 2006, pp. 163–172.

[Amelunxen et al.04] C. Amelunxen, A. Schürr, and L. Bichler, “Codegenerierung für Assoziationen in MOF 2.0,” in *Modellierung*, ser. LNI, B. Rumpe and W. Hesse, Eds., vol. 45. GI, 2004, pp. 149–168.

[AMM] “The AMMA Homepage,” [Website]. URL: http://www.sciences.univ-nantes.fr/lina/atl/

[AMW] “Atlas Model Weaver (AMW),” [Website]. URL: http://www.eclipse.org/gmt/amw/

[André95] E. André, *Ein planbasierter Ansatz zur Generierung multimedialer Präsentationen*, ser. DISKI. Infix Verlag, St. Augustin, Germany, 1995, vol. 108.

[André00] E. André, “The Generation of Multimedia Presentations,” in *A Handbook of Natural Language Processing: techniques and applications for the processing of language as text*, R. Dale, H. Moisl, and H. Somers, Eds. Marcel Dekker Inc., 2000, pp. 305–327.

[App08] *Apple Human Interface Guidelines*, Apple Inc., 3 2008. URL: http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/OSXHIGuidelines.pdf

[Apple88] Apple, Ed., *Hypercard Script Language Guide: The Hypertalk Language*. Boston, MA, USA: Addison Wesley, 8 1988. URL: http://amazon.com/o/ASIN/0201176327/

[Arens and Hovy95] Y. Arens and E. Hovy, "The design of a model-based multimedia interaction manager," *Artif. Intell. Rev.*, vol. 9, no. 2-3, pp. 167–188, 1995.

[Arndt99] T. Arndt, "The evolving role of software engineering in the production of multimedia applications," in *Multimedia Computing and Systems, 1999. IEEE International Conference on*, vol. 1, 7-11 June 1999, pp. 79–84vol.1.

[Arteaga et al.06] J. M. Arteaga, F. J. Martínez-Ruiz, J. Vanderdonckt, and A. Ochoa, "Categorization of Rich Internet Applications based on Similitude Criteria," in *Proc. of XI Simpósio de Informática e VI Mostra de Software Acadêmico SIMS'2006 (Uruguaiana, 8-10 November 2006).* Brazilian Computer Society, 2006.

[ASD] "Action Script Development Tools (ASDT) and AXDT," [Website]. URL: http://www.asdt.org/

[ATE] "ATEM 2007: 4th International Workshop on (Software) Language Engineering," [Website]. URL: http://planet-mde.org/atem2007/

[Atkinson and Kühne01] C. Atkinson and T. Kühne, "The Essence of Multilevel Metamodeling," in *UML 2001 - The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1-5, 2001, Proceedings*, ser. Lecture Notes in Computer Science, M. Gogolla and C. Kobryn, Eds., vol. 2185. Springer, 2001, pp. 19–33.

[Atkinson and Kühne02] C. Atkinson and T. Kühne, "Profiles in a strict metamodeling framework," *Sci. Comput. Program.*, vol. 44, no. 1, pp. 5–22, 2002.

[Atkinson and Kühne03] C. Atkinson and T. Kühne, "Model-Driven Development: A Metamodeling Foundation," *IEEE Software*, vol. 20, no. 5, pp. 36–41, 2003.

[ATLa] "ATL Transformations Zoo," [Website]. URL: http://www.eclipse.org/m2m/atl/atlTransformations/

[Atlb] "AtlanMod – Metamodel Zoos," [Website]. URL: http://www.emn.fr/x-info/atlanmod/index.php/Zoos

[ATo] "AToM3 A Tool for Multi-formalism Meta-Modelling," [Website]. URL: http://atom3.cs.mcgill.ca/

[Aut] "Adobe Authorware," [Website]. URL: http://www.adobe.com/products/authorware/

[Bailey and Konstan03] B. P. Bailey and J. A. Konstan, "Are informal tools better?: comparing DEMAIS, pencil and paper, and authorware for early multimedia design," in *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2003, pp. 313–320.

[Balzert and Weidauer98] H. Balzert and C. Weidauer, “Multimedia-Systeme: Ein neues Anwendungsgebiet für die Software-Technik,” *Softwaretechnik-Trends*, vol. 18, no. 4, pp. 4–9, November 1998.

[Balzert et al.96] H. Balzert, F. Hofmann, V. Kruschinski, and C. Niemann, “The JANUS Application Development Environment - Generating More than the User Interface,” in *Computer-Aided Design of User Interfaces I, Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces, June 5-7, 1996, Namur, Belgium*, J. Vanderdonckt, Ed. Presses Universitaires de Namur, 1996, pp. 183–208.

[Balzert95] H. Balzert, “From OOA to GUIs - the JANUS System,” in *Human-Computer Interaction, INTERACT ’95, IFIP TC13 Interantional Conference on Human-Computer Interaction, 27-29 June 1995, Lillehammer, Norway*, ser. IFIP Conference Proceedings, K. Nordby, P. H. Helmersen, D. J. Gilmore, and S. A. Arnesen, Eds. Chapman & Hall, 1995, pp. 319–324.

[Balzert98] H. Balzert, *Lehrbuch der Softwaretechnik (Bd. II). Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Heidelberg: Spektrum Akademischer Verlag, 1998.

[Bandelloni and Paternò04] R. Bandelloni and F. Paternò, “Flexible interface migration,” in *Proceedings of the 2004 International Conference on Intelligent User Interfaces, January 13-16, 2004, Funchal, Madeira, Portugal*, J. Vanderdonckt, N. J. Nunes, and C. Rich, Eds. ACM, 2004, pp. 148–155.

[Bandelloni et al.04] R. Bandelloni, S. Berti, and F. Paternò, “Mixed-Initiative, Trans-modal Interface Migration,” in *Mobile Human-Computer Interaction - Mobile HCI 2004, 6th International Symposium, Glasgow, UK, September 13-16, 2004, Proceedings*, ser. Lecture Notes in Computer Science, S. A. Brewster and M. D. Dunlop, Eds., vol. 3160. Springer, 2004, pp. 216–227.

[Baranauskas et al.07] M. C. C. Baranauskas, P. A. Palanque, J. Abascal, and S. D. J. Barbosa, Eds., *Human-Computer Interaction - INTERACT 2007, 11th IFIP TC 13 International Conference, Rio de Janeiro, Brazil, September 10-14, 2007, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 4662. Springer, 2007.

[Baresi et al.01] L. Baresi, F. Garzotto, and P. Paolini, “Extending UML for Modeling Web Applications,” in *HICSS ’01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 3*. Washington, DC, USA: IEEE Computer Society, 2001, p. 3055.

[Barry and Lang01] C. Barry and M. Lang, “A Survey of Multimedia and Web Development Techniques and Methodology Usage,” *IEEE MultiMedia*, vol. 8, no. 2, pp. 52–60, 2001.

[Basnyat et al.05] S. Basnyat, J. D. Boeck, E. Cuppens, L. Nóbrega, F. Montero, F. Paternò, and K. Schneider, “Future Challenges of Model-Based Design,” in *Interactive Systems, Design, Specification, and Verification, 12th International Workshop, DSVIS 2005, Newcastle upon Tyne, UK, July 13-15, 2005, Revised Papers*, ser. Lecture Notes in Computer Science, S. W. Gilroy and M. D. Harrison, Eds., vol. 3941. Springer, 2005, p. 261.

[Bastide and Basnyat06] R. Bastide and S. Basnyat, “Error Patterns: Systematic Investigation of Deviations in Task Models,” in *Task Models and Diagrams for Users Interface Design, 5th International Workshop, TAMODIA 2006, Hasselt, Belgium, October 23-24, 2006. Revised Papers*, ser. Lecture Notes in Computer Science, K. Coninx, K. Luyten, and K. A. Schneider, Eds., vol. 4385. Springer, 2006, pp. 109–121.

[Bechhofer et al.04] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, *OWL Web Ontology Language Reference*, W3C, 2004. URL: http://www.w3.org/TR/2004/REC-owl-ref-20040210/

[Beck and Andres04] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2nd ed. Addison-Wesley Professional, 2004.

[Bederson et al.04] B. B. Bederson, J. Grosjean, and J. Meyer, “Toolkit Design for Interactive Structured Graphics,” *IEEE Trans. Softw. Eng.*, vol. 30, no. 8, pp. 535–546, 2004.

[Berti et al.05] S. Berti, F. Paternò, and C. Santoro, “A Taxonomy for Migratory User Interfaces,” in *Interactive Systems, Design, Specification, and Verification, 12th International Workshop, DSVIS 2005, Newcastle upon Tyne, UK, July 13-15, 2005, Revised Papers*, ser. Lecture Notes in Computer Science, S. W. Gilroy and M. D. Harrison, Eds., vol. 3941. Springer, 2005, pp. 149–160.

[Bertino and Ferrari98] E. Bertino and E. Ferrari, “Temporal Synchronization Models for Multimedia Data,” *IEEE Trans. on Knowl. and Data Eng.*, vol. 10, no. 4, pp. 612–631, 1998.

[Bertram et al.99] J. Bertram, C. Kemper, M. Klotz, and S. Nabbefeld, “Abschlussbericht zur Projektgruppe OMMMA,” University of Paderborn, December 1999. URL: http://wwwcs.uni-paderborn.de/cs/ag-engels/ag_dt/Courses/Lehrveranstaltungen/SS98/OMMMA/Homepage/Zwischenbericht/Folien.ps

[Besley et al.03] K. Besley, S. Bhangal, G. Rhodes, B. Monnone, S. Young, K. Peters, A. Eden, and B. Ferguson, *Macromedia Flash MX 2004 Games Most Wanted*, 1st ed. Berkeley, CA, USA: friends of ED, 11 2003. URL: http://amazon.com/o/ASIN/1590592360/

[Bézivin and Heckel05] J. Bézivin and R. Heckel, Eds., *Language Engineering for Model-Driven Software Development, 29. February - 5. March 2004*, ser. Dagstuhl Seminar Proceedings, vol. 04101. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[Bézivin et al.04] J. Bézivin, F. Jouault, and P. Valduriez, “On the Need for Megamodels,” in *OOPSLA Workshop on Best Practices for Model-Driven Software Development*, 2004. URL: www.sciences.univ-nantes.fr/lina/atl/www/papers/OOPSLA04bezivin-megamodel.pdf

[Bézivin05] J. Bézivin, “On the unification power of models,” *Software and System Modeling*, vol. 4, no. 2, pp. 171–188, 2005.

[Bilas05] S. Bilas, “What About Flash? Can You Really Make Games With It?” in *Game Developers Conference 2005*, 2005. URL: http://www.drizzle.com/~scottb/gdc/flash-paper.htm

[Blackwell et al.01] A. F. Blackwell, K. N. Whitley, J. Good, and M. Petre, "Cognitive Factors in Programming with Diagrams," *Artif. Intell. Rev.*, vol. 15, no. 1-2, pp. 95–114, 2001.

[Blair et al.97] G. Blair, L. Blair, H. Bowman, and A. Chetwynd, *Formal Specification of Distributed Multimedia Systems*. University College London Press, September 1997. URL: http://www.cs.kent.ac.uk/pubs/1997/339

[Bock03a] C. Bock, "UML 2 Activity and Action Models," *Journal of Object Technology*, vol. 2, no. 4, pp. 43–53, 2003.

[Bock03b] C. Bock, "UML 2 Activity and Action Models, Part 2," *Journal of Object Technology*, vol. 2, no. 5, pp. 41–56, 2003.

[Bock03c] C. Bock, "UML 2 Activity and Action Models, Part 3: Control Nodes," *Journal of Object Technology*, vol. 2, no. 6, pp. 7–23, 2003.

[Bock04a] C. Bock, "UML 2 Activity and Action Models, Part 4: Object Nodes," *Journal of Object Technology*, vol. 3, no. 1, pp. 27–41, 2004.

[Bock04b] C. Bock, "UML 2 Activity and Action Models, Part 5: Partitions," *Journal of Object Technology*, vol. 3, no. 7, pp. 37–56, 2004.

[Bodart et al.95] F. Bodart, A.-M. Hennebert, J.-M. Leheureux, I. Provot, B. Sacré, and J. Vanderdonckt, "Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide," in *Design, Specification and Verification of Interactive Systems '95, Proceedings of the Eurographics Workshop in Toulouse, France June 7-9, 1995*, P. A. Palanque and R. Bastide, Eds. Springer, 1995, pp. 262–278.

[Bodoff et al.05] D. Bodoff, M. Ben-Menachem, and P. C. K. Hung, "Web Metadata Standards: Observations and Prescriptions," *IEEE Softw.*, vol. 22, no. 1, pp. 78–85, 2005.

[Boles and Schlattmann98] D. Boles and M. Schlattmann, "Multimedia-Autorensysteme - Grafisch-interaktive Werkzeuge zur Erstellung multimedialer Anwendungen," *LOG IN*, vol. 18, no. 1, pp. 10–, 1998.

[Boles et al.98] D. Boles, P. Dawabi, M. Schlattmann, E. Boles, C. Trunk, and F. Wigger, "Objektorientierte Multimedia-Softwareentwicklung: Vom UML-Modell zur Director-Anwendung am Beispiel virtueller naturwissenschaftlich-technischer Labore," in *Workshop Multimedia-Systeme, GI Jahrestagung*, 1998, pp. 33–51.

[Boll01] S. Boll, "Zyx – Towards flexible multimedia document models for reuse and adaptation," Phd, Vienna University of Technology, Vienna, Austria, August 2001. URL: http://medien.informatik.uni-oldenburg.de/index.php?id=31

[Bolt80] R. A. Bolt, "P̈ut-that-there:̈ Voice and gesture at the graphics interface," in *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1980, pp. 262–270.

[Booch et al.07] G. Booch, R. A. Maksimchuk, M. W. Engel, B. J. Young, J. Conallen, and K. A. Houston, *Object-Oriented Analysis and Design with Applications*, 3rd ed. Addison-Wesley Professional, 2007.

[Botterweck06] G. Botterweck, “A Model-Driven Approach to the Engineering of Multiple User Interfaces,” in *Models in Software Engineering, Workshops and Symposia at MoDELS 2006, Genoa, Italy, October 1-6, 2006, Reports and Revised Selected Papers*, ser. Lecture Notes in Computer Science, T. Kühne, Ed., vol. 4364. Springer, 2006, pp. 106–115.

[Bouillon et al.04] L. Bouillon, J. Vanderdonckt, and K. C. Chow, “Flexible re-engineering of web sites,” in *Proceedings of the 2004 International Conference on Intelligent User Interfaces, January 13-16, 2004, Funchal, Madeira, Portugal*, J. Vanderdonckt, N. J. Nunes, and C. Rich, Eds. ACM, 2004, pp. 132–139.

[Bourguin et al.07] G. Bourguin, A. Lewandowski, and J.-C. Tarby, “Defining Task Oriented Components,” in *Task Models and Diagrams for User Interface Design, 6th International Workshop, TAMODIA 2007, Toulouse, France, November 7-9, 2007, Proceedings*, ser. Lecture Notes in Computer Science, M. Winckler, H. Johnson, and P. A. Palanque, Eds., vol. 4849. Springer, 2007, pp. 170–183.

[Box98] D. Box, *Essential COM*. Boston, MA, USA: Addison-Wesley Professional, 1 1998. URL: http://amazon.com/o/ASIN/0201634465/

[Bozzon et al.06] A. Bozzon, S. Comai, P. Fraternali, and G. T. Carughi, “Conceptual modeling and code generation for rich internet applications,” in *ICWE ’06: Proceedings of the 6th international conference on Web engineering*. New York, NY, USA: ACM, 2006, pp. 353–360.

[Bra] “Fluidum – BrainStorm.” URL: http://www.fluidum.org/projects_brainstorm.shtml

[Braun and Mühlhäuser05] E. Braun and M. Mühlhäuser, “Automatically Generating User Interfaces for Device Federations,” in *Seventh IEEE International Symposium on Multimedia (ISM 2005), 12-14 December 2005, Irvine, CA, USA*. IEEE Computer Society, 2005, pp. 261–268.

[Briand and Williams05] L. C. Briand and C. Williams, Eds., *Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3713. Springer, 2005.

[Britton et al.97] C. Britton, S. Jones, M. Myers, and M. Sharif, “A survey of current practice in the development of multimedia systems,” *Information & Software Technology*, vol. 39, no. 10, pp. 695–705, 1997.

[Broy et al.93] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hussmann, D. Nazareth, F. Regensburger, O. Slotosch, and K. Stølen, “The Requirement and Design Specification Language SPECTRUM. An Informal Introduction. Version 1.0,” Technische Universität München, Tech. Rep. TUM-I9312, May 1993.

[Broy et al.05] M. Broy, J. Dingel, A. Hartman, B. Rumpe, and B. Selic, Eds., *A Formal Semantics for UML, Workshop on European Conference on Model Driven Architecture ECMDA 2005*, 2005. URL: http://www.cs.queensu.ca/~stl/internal/uml2/ECMDA2005/index.html

[Bruck and Hussey07] J. Bruck and K. Hussey, “Customizing UML: Which Technique is Right for You?” 2007. URL: http://www.eclipse.org/modeling/mdt/uml2/docs/articles/Customizing_UML2_Which_Technique_is_Right_For_You/article.html

[Bruel06] J.-M. Bruel, Ed., *Satellite Events at the MoDELS 2005 Conference, MoDELS 2005 International Workshops, Doctoral Symposium, Educators Symposium, Montego Bay, Jamaica, October 2-7, 2005, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 3844. Springer, 2006.

[Buchanan and Zellweger05] M. C. Buchanan and P. T. Zellweger, “Automatic temporal layout mechanisms revisited,” *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 1, no. 1, pp. 60–88, 2005.

[Budinsky et al.03] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose, *Eclipse Modeling Framework*. Boston, MA, USA: Addison-Wesley, 8 2003. URL: http://amazon.com/o/ASIN/0131425420/

[Bulterman and Hardman05] D. C. A. Bulterman and L. Hardman, “Structured multimedia authoring,” *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 1, no. 1, pp. 89–109, 2005.

[Bulterman and Rutledge04] D. C. Bulterman and L. W. Rutledge, *SMIL 2.0*. Heidelberg, Germany: Springer, 2004.

[Bulterman et al.91] D. C. A. Bulterman, G. van Rossum, and R. van Liere, “A Structure for Transportable, Dynamic Multimedia Documents,” in *Proceedings of the Summer 1991 USENIX Conference, Nashville, TN*. USENIX Association, 1991, pp. 137–155.

[Bulterman et al.98] D. C. A. Bulterman, L. Hardman, J. Jansen, K. S. Mullender, and L. Rutledge, “GRiNS: a graphical interface for creating and playing SMIL documents,” in *WWW7: Proceedings of the seventh international conference on World Wide Web 7*. Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V., 1998, pp. 519–529.

[Bulterman et al.05] D. Bulterman, G. Grassel, J. Jansen, A. Koivisto, N. Layaïda, T. Michel, S. Mullender, and D. Zucker, *Synchronized Multimedia Integration Language (SMIL 2.1)*, W3C, December 2005, [Website]. URL: http://www.w3.org/TR/2005/REC-SMIL2-20051213/

[Burmester et al.05] S. Burmester, H. Giese, and S. Henkler, “Visual Model-Driven Development of Software Intensive Systems: A Survey of available Techniques and Tools,” in *Proc. of the Workshop on Visual Modeling for Software Intensive Systems (VMSIS) at the the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC’05), Dallas, Texas, USA*, September 2005, pp. 11–18.

[CAD] “Computer-Aided Design of User Interfaces,” [Website]. URL: http://www.isys.ucl.ac.be/bchi/cadui/

[CAD07] *Computer-Aided Design of User Interfaces V: Proceedings of the Sixth International Conference on Computer-Aided Design of User Interfaces CADUI ’06 (6-8 June 2006, Bucharest, Romania)*. Springer, 2007.

[Calvary et al.02] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, N. Souchon, L. Bouillon, M. Florins, and J. Vanderdonckt, “Plasticity of User Interfaces: A Revised Reference Framework,” in *Task Models and Diagrams for User Interface Design: Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2002, 18-19 July 2002, Bucharest, Romania*, C. Pribeanu and J. Vanderdonckt, Eds. INFOREC Publishing House Bucharest, 2002, pp. 127–134.

[Calvary et al.03] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, “A Unifying Reference Framework for multi-target user interfaces,” *Interacting with Computers*, vol. 15, no. 3, pp. 289–308, 2003.

[Can] “University of Madeira – CanonSketch,” [Website]. URL: http://dme.uma.pt/projects/canonsketch/

[Capraro et al.04] M. Capraro, D. McAlester, E. Bianchi, C. Corbin, D. Design, A. Danika, A. Heim, R. Hoekman, T. Marks, B. Spencer, and J. Williamson, *Macromedia Flash MX 2004 Magic*. Indianapolis, IN, USA: New Riders Press, 2 2004. URL: http://amazon.com/o/ASIN/0735713774/

[Carughi et al.07] G. T. Carughi, S. Comai, A. Bozzon, and P. Fraternali, “Modeling Distributed Events in Data-Intensive Rich Internet Applications,” in *Web Information Systems Engineering - WISE 2007, 8th International Conference on Web Information Systems Engineering, Nancy, France, December 3-7, 2007, Proceedings*, ser. Lecture Notes in Computer Science, B. Benatallah, F. Casati, D. Georgakopoulos, C. Bartolini, W. Sadiq, and C. Godart, Eds., vol. 4831. Springer, 2007, pp. 593–602.

[Ceri et al.02] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera, *Designing Data-Intensive Web Applications*, 1st ed. Morgan Kaufmann, 12 2002. URL: http://amazon.com/o/ASIN/1558608435/

[Chang99] S.-K. Chang, “Perspectives in Multimedia Software Engineering,” in *ICMCS, Vol. 1*, 1999, pp. 74–78.

[Chen76] P. P.-S. Chen, “The entity-relationship model–toward a unified view of data,” *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, 1976.

[CHI] “CHI 2008 Conference Website,” [Website]. URL: http://www.chi2008.org/

[Clerckx and Coninx05] T. Clerckx and K. Coninx, “Towards an Integrated Development Environment for Context-Aware User Interfaces,” in *Mobile Computing and Ambient Intelligence: The Challenge of Multimedia, 1.-4. May 2005*, ser. Dagstuhl Seminar Proceedings, N. Davies, T. Kirste, and H. Schumann, Eds., vol. 05181. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany IBFI, Schloss Dagstuhl, Germany, 2005.

[Clerckx et al.04] T. Clerckx, K. Luyten, and K. Coninx, “The mapping problem back and forth: customizing dynamic models while preserving consistency,” in *Task Models and Diagrams for User Interface Design: Proceedings of the Third International Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2004, November 15 - 16, 2004, Prague, Czech Republic*, P. Slavík and P. A. Palanque, Eds. ACM, 2004, pp. 33–42.

[Clerckx et al.05a] T. Clerckx, K. Luyten, and K. Coninx, “Designing Interactive Systems in Context: From Prototype to Deployment,” in *People and Computers XIX Ů The Bigger Picture - Proceedings of HCI 2005*. London: Springer, 2005, pp. 85–100.

[Clerckx et al.05b] T. Clerckx, F. Winters, and K. Coninx, “Tool support for designing context-sensitive user interfaces using a model-based approach,” in *Task Models and Diagrams for User Interface Design: Proceedings of the Forth International Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2005, Gdansk, Poland, September 26-27, 2005*, M. Sikorski, Ed. ACM, 2005, pp. 11–18.

[Cockton87] G. Cockton, “Interaction ergonomics, control and separation: open problems in user interface management,” *Inf. Softw. Technol.*, vol. 29, no. 4, pp. 176–191, 1987.

[Cohen et al.97] P. R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow, “QuickSet: multimodal interaction for distributed applications,” in *MULTIMEDIA ’97: Proceedings of the fifth ACM international conference on Multimedia*. New York, NY, USA: ACM, 1997, pp. 31–40.

[Conallen00] J. Conallen, *Building Web applications with UML*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[Conallen02] J. Conallen, *Building Web Applications with Uml*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[Coninx et al.03] K. Coninx, K. Luyten, C. Vandervelpen, J. V. den Bergh, and B. Creemers, “Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems,” in *Human-Computer Interaction with Mobile Devices and Services, 5th International Symposium, Mobile HCI 2003, Udine, Italy, September 8-11, 2003, Proceedings*, ser. Lecture Notes in Computer Science, L. Chittaro, Ed., vol. 2795. Springer, 2003, pp. 256–270.

[Coninx et al.07] K. Coninx, K. Luyten, and K. A. Schneider, Eds., *Task Models and Diagrams for Users Interface Design, 5th International Workshop, TAMODIA 2006, Hasselt, Belgium, October 23-24, 2006. Revised Papers*, ser. Lecture Notes in Computer Science, vol. 4385. Springer, 2007.

[Constantine and Lockwood99] L. L. Constantine and L. A. Lockwood, *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design (ACM Press)*. Addison-Wesley, 1999.

[Constantine and Lockwood01] L. L. Constantine and L. A. D. Lockwood, *Structure and style in use cases for user interface design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 245–279.

[Constantine03] L. L. Constantine, “Canonical Abstract Prototypes for Abstract Visual and Interaction,” in *Interactive Systems. Design, Specification, and Verification, 10th International Workshop, DSV-IS 2003, Funchal, Madeira Island, Portugal, June 11-13, 2003, Revised Papers*, ser. Lecture Notes in Computer Science, J. A. Jorge, N. J. Nunes, and J. ao Falcão e Cunha, Eds., vol. 2844. Springer, 2003, pp. 1–15.

[Cook et al.07] S. Cook, G. Jones, S. Kent, and A. C. Wills, *Domain-Specific Development with Visual Studio DSL Tools*. Boston, MA, USA: Addison Wesley, 6 2007. URL: http://amazon.com/o/ASIN/0321398203/

[Cor] "CorelDRAW Graphics Suite," [Website]. URL: http://www.corel.com/servlet/Satellite/us/en/Product/1191272117978#tabview=tab0

[Coutaz and Caelen91] J. Coutaz and J. Caelen, "A Taxonomy for Multimedia and Multimodal User Interfaces," in *Proceedings of the ERCIM Workshop on User Interfaces and Multimedia*, 1991, pp. 143–147. URL: http://iihm.imag.fr/publs/1991/ERCIM91_PoleIHMM.ps.gz

[Coutaz and Rey02] J. Coutaz and G. Rey, "Foundations for a Theory of Contextors," in *Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, May, 15-17, 2002, Valenciennes, France*, C. Kolski and J. Vanderdonckt, Eds. Kluwer, 2002, pp. 13–34.

[Coutaz et al.07] J. Coutaz, L. Balme, X. Alvaro, G. Calvary, A. Demeure, and J.-S. Sottet, "An MDE-SOA Approach to Support Plastic User Interfaces in Ambient Spaces," in *Universal Access in Human-Computer Interaction. Ambient Interaction, 4th International Conference on Universal Access in Human-Computer Interaction, UAHCI 2007 Held as Part of HCI International 2007 Beijing, China, July 22-27, 2007 Proceedings, Part II*, ser. Lecture Notes in Computer Science, C. Stephanidis, Ed., vol. 4555. Springer, 2007, pp. 63–72.

[Coutaz06] J. Coutaz, "Meta-User Interfaces for Ambient Spaces," in *Task Models and Diagrams for Users Interface Design, 5th International Workshop, TAMODIA 2006, Hasselt, Belgium, October 23-24, 2006. Revised Papers*, ser. Lecture Notes in Computer Science, K. Coninx, K. Luyten, and K. A. Schneider, Eds., vol. 4385. Springer, 2006, pp. 1–15.

[Coyette et al.07] A. Coyette, S. Kieffer, and J. Vanderdonckt, "Multi-fidelity Prototyping of User Interfaces," in *Human-Computer Interaction - INTERACT 2007, 11th IFIP TC 13 International Conference, Rio de Janeiro, Brazil, September 10-14, 2007, Proceedings, Part I*, ser. Lecture Notes in Computer Science, M. C. C. Baranauskas, P. A. Palanque, J. Abascal, and S. D. J. Barbosa, Eds., vol. 4662. Springer, 2007, pp. 150–164.

[Csertán et al.02] G. Csertán, G. Huszerl, I. Majzik, Z. Pap, A. Pataricza, and D. Varró, "VIATRA – Visual Automated Transformations for Formal Verification and Validation of UML Models," in *ASE '02: Proceedings of the 17th IEEE international conference on Automated software engineering*. Washington, DC, USA: IEEE Computer Society, 2002, p. 267.

[Cub] "Steinberg Cubase," [Website]. URL: http://www.steinberg.net/en/products/musicproduction/cubase4_product.html

[CWI] "Centrum Wiskunde & Informatica (CWI), Netherlands," [Website]. URL: http://www.cwi.nl/

[Czarnecki and Helsen06] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Syst. J.*, vol. 45, no. 3, pp. 621–645, 2006.

[da Silva and Paton00] P. P. da Silva and N. W. Paton, “UMLi: The Unified Modeling Language for Interactive Applications,” in *UML 2000 - The Unified Modeling Language, Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000, Proceedings*, ser. Lecture Notes in Computer Science, A. Evans, S. Kent, and B. Selic, Eds., vol. 1939. Springer, 2000, pp. 117–132.

[da Silva and Paton03] P. P. da Silva and N. W. Paton, “User Interface Modeling in UMLi,” *IEEE Software*, vol. 20, no. 4, pp. 62–69, 2003.

[da Silva00] P. P. da Silva, “User Interface Declarative Models and Development Environments: A Survey,” in *DSV-IS*, 2000, pp. 207–226.

[Damus07] C. W. Damus, *Implementing Model Integrity in EMF with MDT OCL*, Eclipse Corner Articles, 2007. URL: http://www.eclipse.org/articles/Article-EMF-Codegen-with-OCL/

[Dawes01] B. Dawes, *Drag, Slide, Fade – Flash ActionScript for Designers*. Indianapolis, IN, USA: New Riders, 11 2001. URL: http://amazon.de/o/ASIN/0735710473/

[DBL01] *2002 IEEE CS International Symposium on Human-Centric Computing Languages and Environments (HCC 2001), September 5-7, 2001 Stresa, Italy*. IEEE Computer Society, 2001.

[DBL05] *Seventh IEEE International Symposium on Multimedia (ISM 2005), 12-14 December 2005, Irvine, CA, USA*. IEEE Computer Society, 2005.

[deHaan06] J. deHaan, “Flash 8 Best Practices,” 2006. URL: http://www.adobe.com/devnet/flash/articles/flash8_bestpractices.html

[Depke et al.99] R. Depke, G. Engels, K. Mehner, S. Sauer, and A. Wagner, “Ein Vorgehensmodell für die Multimedia-Entwicklung mit Autorensystemen,” *Inform., Forsch. Entwickl.*, vol. 14, no. 2, pp. 83–94, 1999.

[Desfray00] P. Desfray, “UML Profiles versus Metamodel extensions : An ongoing debate,” 2000, talk at OMG Workshop on UML In The .com Enterprise. URL: http://www.omg.org/news/meetings/workshops/presentations/uml_presentations/5-3%20Desfray%20-%20UMLWorkshop.pdf

[Dix et al.03] A. Dix, J. E. Finlay, G. D. Abowd, and R. Beale, *Human-Computer Interaction*, 3rd ed. Prentice Hall, 2003.

[Doherty and Blandford07] G. J. Doherty and A. Blandford, Eds., *Interactive Systems. Design, Specification, and Verification, 13th International Workshop, DSVIS 2006, Dublin, Ireland, July 26-28, 2006. Revised Papers*, ser. Lecture Notes in Computer Science, vol. 4323. Springer, 2007.

[Dospisil and Polgar94] J. Dospisil and T. Polgar, “Conceptual modelling in the hypermedia development process,” in *SIGCPR ’94: Proceedings of the 1994 computer personnel research conference on Reinventing IS : managing information technology in changing organizations*. New York, NY, USA: ACM, 1994, pp. 97–104.

[DSV] "DSV-IS 2008 – The XVth International Workshop on Design, Specification and Verification of Interactive Systems," [Website]. URL: http://www.cs.queensu.ca/dsvis2008/

[Duyne et al.02] D. K. V. Duyne, J. Landay, and J. I. Hong, *The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[Eckstein04] J. Eckstein, *Agile Softwareentwicklung im Groſsen. Ein Eintauchen in die Untiefen erfolgreicher Projekte.* Heidelberg: Dpunkt Verlag, 2004.

[Ecla] "Eclipse – Model Development Tools (MDT) – OCL," [Website]. URL: http://www.eclipse.org/modeling/mdt/?project=ocl

[Eclb] "Eclipse - an open development platform," [Website]. URL: http://www.eclipse.org/

[Eclc] "Eclipse M2M Project," [Website]. URL: http://www.eclipse.org/m2m/

[Ecld] "Eclipse Modeling Project," [Website]. URL: http://www.eclipse.org/modeling/

[Ecm99] *Standard ECMA-262 – ECMAScript Language Specification*, Ecma International, Geneva, Switzerland, 1999, [Website]. URL: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[Eicher05] C. Eicher, "Konzeption und prototypische Realisierung eines Graphischen Editors zur Unterstützung von Präsentationsdiagrammen," Project Thesis, University of Munich, Munich, August 2005.

[EIS] "EIS 2008 – Engineering Interactive Systems," [Website]. URL: http://www.se-hci.org/ehci-hcse-dsvis07/

[Elwert and Schlungbaum95] T. Elwert and E. Schlungbaum, "Modelling and Generation of Graphical User Interfaces in the TADEUS Approach," in *Design, Specification and Verification of Interactive Systems '95, Proceedings of the Eurographics Workshop in Toulouse, France June 7-9, 1995*, P. A. Palanque and R. Bastide, Eds. Springer, 1995, pp. 193–208.

[EMFa] "Eclipse Modeling - MDT - EMF UML2," [Website]. URL: http://www.eclipse.org/modeling/mdt/?project=uml2

[EMFb] "Eclipse Modeling Framework Project (EMF)," [Website]. URL: http://www.eclipse.org/modeling/emf/

[Engels and Sauer02] G. Engels and S. Sauer, "Object-oriented Modeling of Multimedia Applications," in *Handbook of Software Engineering and Knowledge Engineering*, S. K. Chang, Ed. Singapore: World Scientific, 2002, vol. 2, pp. 21–53.

[Engels et al.00] G. Engels, J. H. Hausmann, R. Heckel, and S. Sauer, "Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML," in *UML 2000 - The Unified Modeling Language, Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000, Proceedings*, ser. Lecture Notes in Computer Science, A. Evans, S. Kent, and B. Selic, Eds., vol. 1939. Springer, 2000, pp. 323–337.

[Eun et al.94] S. Eun, E. S. No, H. C. Kim, H. Yoon, and S. R. Maeng, "Eventor: an authoring system for interactive multimedia applications," *Multimedia Syst.*, vol. 2, no. 3, pp. 129–140, 1994.

[Evans et al.00] A. Evans, S. Kent, and B. Selic, Eds., *UML 2000 - The Unified Modeling Language, Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000, Proceedings*, ser. Lecture Notes in Computer Science, vol. 1939. Springer, 2000.

[Expa] "Microsoft Expression Blend," [Website]. URL: http://www.microsoft.com/expression/products/Overview.aspx?key=blend

[Expb] "Microsoft Expression Design," [Website]. URL: http://www.microsoft.com/expression/products/Overview.aspx?key=design

[Fabro et al.06] M. D. D. Fabro, J. Bézivin, and P. Valduriez, "Weaving Models with the Eclipse AMW plugin," in *Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany*, 2006. URL: http://www.eclipsecon.org/summiteurope2006/presentations/ESE2006-EclipseModelingSymposium2_WeavingModels.pdf

[Favre and Nguyen05] J.-M. Favre and T. Nguyen, "Towards a Megamodel to Model Software Evolution Through Transformations," *Electr. Notes Theor. Comput. Sci.*, vol. 127, no. 3, pp. 59–74, 2005.

[Favre04a] J.-M. Favre, "Foundations of Meta-Pyramids: Languages vs. Metamodels - Episode II: Story of Thotus the Baboon1," in *Language Engineering for Model-Driven Software Development, 29. February - 5. March 2004*, ser. Dagstuhl Seminar Proceedings, J. Bézivin and R. Heckel, Eds., vol. 04101. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2004.

[Favre04b] J.-M. Favre, "Foundations of Model (Driven) (Reverse) Engineering : Models - Episode I: Stories of The Fidus Papyrus and of The Solarus," in *Language Engineering for Model-Driven Software Development, 29. February - 5. March 2004*, ser. Dagstuhl Seminar Proceedings, J. Bézivin and R. Heckel, Eds., vol. 04101. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2004.

[Favre04c] J.-M. Favre, "Towards a Basic Theory to Model Model Driven Engineering," in *Workshop on Software Model Engineering (WISME 2004), joint event with UML2004*, 2004. URL: http://www-adele.imag.fr/users/Jean-Marie.Favre/

[Felipe et al.05] J. C. Felipe, J. B. Olioti, A. J. M. Traina, M. X. Ribeiro, E. P. M. de Sousa, and C. T. Jr., "A Low-cost Approach for Effective Shape-based Retrieval and Classification of Medical Images," in *Seventh IEEE International Symposium on Multimedia (ISM 2005), 12-14 December 2005, Irvine, CA, USA*. IEEE Computer Society, 2005, pp. 565–570.

[Fetterman and Gupta93] R. L. Fetterman and S. K. Gupta, *Mainstream Multimedia: Applying Multimedia in Business*. New York: Van Nostrand Reinhold, 6 1993. URL: http://amazon.com/o/ASIN/0442011814/

[Feuerstack et al.08] S. Feuerstack, M. Blumendorf, V. Schwartze, and S. Albayrak, “Model-based layout generation,” in *AVI ’08: Proceedings of the working conference on Advanced visual interfaces*. New York, NY, USA: ACM, 2008, pp. 217–224.

[FFm] “FFmpeg,” [Website]. URL: http://ffmpeg.org/

[Fin] “Apple Final Cut,” [Website]. URL: http://www.apple.com/finalcutstudio/finalcutpro/

[Finkenzeller08] S. Finkenzeller, “Untersuchung und Verbesserung der visuellen Modellierungssprache MML,” Project Thesis, University of Munich, Munich, February 2008.

[Flaa] “Adobe Flash CS4 Professional,” [Website]. URL: http://www.adobe.com/products/flash/

[Flab] “Adobe Flash Lite,” [Website]. URL: http://www.adobe.com/products/flashlite/

[Foley et al.91] J. Foley, W. C. Kim, S. Kovacevic, and K. Murray, “UIDE–an intelligent user interface design environment,” in *Intelligent user interfaces*. New York, NY, USA: ACM, 1991, pp. 339–384.

[Frank and Prasse97] U. Frank and M. Prasse, “Ein Bezugsrahmen zur Beurteilung Objektorientierter Modellierungssprachen – Veranschaulicht Am Beispiel von OML und UML,” Universität Koblenz-Landau, Tech. Rep. 6, 1997. URL: http://www.wi-inf.uni-duisburg-essen.de/MobisPortal/upload/Nr6.pdf

[Franklin and Makar03] D. Franklin and J. Makar, *Macromedia Flash MX 2004 ActionScript: Training from the Source*. Berkeley, CA, USA: Macromedia Press, 11 2003. URL: http://amazon.com/o/ASIN/0321213432/

[Fraternali and Paolini98] P. Fraternali and P. Paolini, “A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications,” in *EDBT ’98: Proceedings of the 6th International Conference on Extending Database Technology*. London, UK: Springer-Verlag, 1998, pp. 421–435.

[Gajos and Weld04] K. Gajos and D. S. Weld, “SUPPLE: automatically generating user interfaces,” in *Proceedings of the 2004 International Conference on Intelligent User Interfaces, January 13-16, 2004, Funchal, Madeira, Portugal*, J. Vanderdonckt, N. J. Nunes, and C. Rich, Eds. ACM, 2004, pp. 93–100.

[Gallagher and Webb97] S. Gallagher and B. Webb, “Competing Paradigms in Multimedia Systems Development: Who Shall Be The Aristocracy,” in *Proceedings of the Fifth European Conference on Information Systems*. Cork, Ireland: Cork Publishing Ltd, 1997, pp. 1113–1120.

[Gamma et al.95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[Gartner] Gartner, “Understanding Hype Cycles,” [website].

[Garzotto et al.95] F. Garzotto, L. Mainetti, and P. Paolini, "Hypermedia Design, Analysis, and Evaluation Issues," *Commun. ACM*, vol. 38, no. 8, pp. 74–86, 1995.

[Gasevic et al.07] D. Gasevic, N. Kaviani, and M. Hatala, "On Metamodeling in Megamodels," in *Model Driven Engineering Languages and Systems, 10th International Conference, MoDELS 2007, Nashville, USA, September 30 - October 5, 2007, Proceedings*, ser. Lecture Notes in Computer Science, G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, Eds., vol. 4735. Springer, 2007, pp. 91–105.

[Gauffre et al.07] G. Gauffre, E. Dubois, and R. Bastide, "Domain Specific Methods and Tools for the Design of Advanced Interactive Techniques," in *Proceedings of the MoDELS 2007 Workshop on Model Driven Development of Advanced User Interfaces, Nashville, Tennessee, USA, October 1, 2007*, ser. CEUR Workshop Proceedings, A. Pleuß, J. V. den Bergh, H. Hußmann, S. Sauer, and D. Görlich, Eds., vol. 297. CEUR-WS.org, 2007.

[GEF] "Graphical Editing Framework (GEF)," [Website]. URL: http://www.eclipse.org/gef/

[Gellersen et al.97] H.-W. Gellersen, R. Wicke, and M. Gaedke, "WebComposition: an object-oriented support system for the Web engineering lifecycle," in *Selected papers from the sixth international conference on World Wide Web*. Essex, UK: Elsevier Science Publishers Ltd., 1997, pp. 1429–1437.

[Génova et al.03] G. Génova, C. R. del Castillo, and J. Lloréns, "Mapping UML Associations into Java Code," *Journal of Object Technology*, vol. 2, no. 5, pp. 135–162, 2003.

[Gentleware] Gentleware, "Poseidon for UML," [Website]. URL: http://www.gentleware.com/products.html

[Gibbs and Tsichritzis95] S. J. Gibbs and D. Tsichritzis, *Multimedia programming: objects, environments and frameworks*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995.

[Gibbs et al.94] S. J. Gibbs, C. Breiteneder, and D. Tsichritzis, "Data Modeling of Time-Based Media," in *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994*, R. T. Snodgrass and M. Winslett, Eds. ACM Press, 1994, pp. 91–102.

[Gilroy and Harrison06] S. W. Gilroy and M. D. Harrison, Eds., *Interactive Systems, Design, Specification, and Verification, 12th International Workshop, DSVIS 2005, Newcastle upon Tyne, UK, July 13-15, 2005, Revised Papers*, ser. Lecture Notes in Computer Science, vol. 3941. Springer, 2006.

[GME] "GME: The Generic Modeling Environment," [Website]. URL: http://w3.isis.vanderbilt.edu/projects/gme/

[GMF] "The Eclipse Graphical Modeling Framework (GMF)," [Website]. URL: http://www.eclipse.org/modeling/gmf/

[GNO04] *GNOME Human Interface Guidelines 2.0*, The GNOME Usability Project, 2004. URL: http://library.gnome.org/devel/hig-book/stable/

[Gomaa et al.05] M. Gomaa, A. Salah, and S. Rahman, “Towards A Better Model Based User Interface Development Environment: A Comprehensive Survey,” in *Midwest Instruction and Computing Symposium, April 8 - 9, 2005, Eau Claire, Wisconsin, USA*, April 2005. URL: http://www.micsymposium.org/mics_2005/papers/paper72.pdf

[Gómez et al.01] J. Gómez, C. Cachero, and O. Pastor, “Conceptual Modeling of Device-Independent Web Applications,” *IEEE MultiMedia*, vol. 8, no. 2, pp. 26–39, 2001.

[Gonzalez00] R. Gonzalez, “Disciplining Multimedia,” *IEEE MultiMedia*, vol. 7, no. 3, pp. 72–78, 2000.

[Goo] “Google Maps,” [Website]. URL: http://maps.google.com/

[Görlich and Breiner07] D. Görlich and K. Breiner, “Useware Modeling for Ambient Intelligent Production Environments,” in *Proceedings of the MoDELS 2007 Workshop on Model Driven Development of Advanced User Interfaces, Nashville, Tennessee, USA, October 1, 2007*, ser. CEUR Workshop Proceedings, A. Pleuß, J. V. den Bergh, H. Hußmann, S. Sauer, and D. Görlich, Eds., vol. 297. CEUR-WS.org, 2007.

[Gra] “UsiXML – GrafiXML,” [Website]. URL: http://www.usixml.org/index.php?mod=pages&id=12

[Grana et al.05] C. Grana, G. Tardini, and R. Cucchiara, “MPEG-7 Compliant Shot Detection in Sport Videos,” in *Seventh IEEE International Symposium on Multimedia (ISM 2005), 12-14 December 2005, Irvine, CA, USA*. IEEE Computer Society, 2005, pp. 395–402.

[Green and Petre96] T. R. G. Green and M. Petre, “Usability Analysis of Visual Programming Environments: A ’Cognitive Dimensions’ Framework,” *Journal of Visual Languages and Computing*, vol. 7, no. 2, pp. 131–174, 1996.

[Green00] T. R. G. Green, “Instructions and descriptions: some cognitive aspects of programming and similar activities,” in *AVI ’00: Proceedings of the working conference on Advanced visual interfaces*. New York, NY, USA: ACM, 2000, pp. 21–28.

[Griffiths et al.98] T. Griffiths, J. McKirdy, G. Forrester, N. W. Paton, J. B. Kennedy, P. J. Barclay, R. Cooper, C. A. Goble, and P. D. Gray, “Exploiting Model-based Techniques for User Interfaces to Databases,” in *Visual Database Systems 4 (VDB4), IFIP TC2/WG 2.6 Fourth Working Conference on Visual Database Systems, L’Aquila, Italy, 27-29 May 1998*, ser. IFIP Conference Proceedings, Y. E. Ioannidis and W. Klas, Eds., vol. 126. Chapman & Hall, 1998, pp. 21–46.

[Griffiths et al.99] T. Griffiths, P. J. Barclay, J. McKirdy, N. W. Paton, P. D. Gray, J. B. Kennedy, R. Cooper, C. A. Goble, A. West, and M. Smyth, “Teallach: A Model-Based User Interface Development Environment for Object Databases,” in *UIDIS*, 1999, pp. 86–96.

[Gruhn et al.06] V. Gruhn, D. Pieper, and C. Röttgers, *MDA: Effektives Softwareengineering mit UML2 und Eclipse*, 1st ed. Springer, Berlin, 7 2006. URL: http://amazon.de/o/ASIN/3540287442/

[Gruhn07] R. Gruhn, “Entwicklung von Rich Internet Applications mit dem Adobe Flex Framework,” Project Thesis, University of Munich, Munich, December 2007.

[Halasz and Schwartz94] F. Halasz and M. Schwartz, "The Dexter hypertext reference model," *Commun. ACM*, vol. 37, no. 2, pp. 30–39, 1994.

[Hall and Wan02] B. Hall and S. Wan, *Object-Oriented Programming with Actionscript*, 1st ed. Indianapolis, IN, USA: New Riders, 12 2002. URL: http://amazon.de/o/ASIN/0735711836/

[Hannington and Reed02] A. Hannington and K. Reed, "Towards a Taxonomy for Guiding Multimedia Application Development," in *APSEC '02: Proceedings of the Ninth Asia-Pacific Software Engineering Conference*. Washington, DC, USA: IEEE Computer Society, 2002, p. 97.

[Hannington and Reed06] A. Hannington and K. Reed, "Preliminary Results from a Survey of Multimedia Development Practices in Australia," in *Product-Focused Software Process Improvement, 7th International Conference, PROFES 2006, Amsterdam, The Netherlands, June 12-14, 2006, Proceedings*, ser. Lecture Notes in Computer Science, J. Münch and M. Vierimaa, Eds., vol. 4034. Springer, 2006, pp. 192–207.

[Hannington and Reed07] A. Hannington and K. Reed, "Factors in Multimedia Project and Process Management–Australian Survey Findings," in *ASWEC '07: Proceedings of the 2007 Australian Software Engineering Conference*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 379–388.

[Hardman et al.94] L. Hardman, D. C. A. Bulterman, and G. van Rossum, "The Amsterdam hypermedia model: adding time and context to the Dexter model," *Commun. ACM*, vol. 37, no. 2, pp. 50–62, 1994.

[Hardman et al.97] L. Hardman, M. Worring, and D. C. A. Bulterman, "Integrating the Amsterdam hypermedia model with the standard reference model for intelligent multimedia presentation systems," *Comput. Stand. Interfaces*, vol. 18, no. 6-7, pp. 497–507, 1997.

[Harel and Naamad96] D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts," *ACM Trans. Softw. Eng. Methodol.*, vol. 5, no. 4, pp. 293–333, 1996.

[Harel and Rumpe00] D. Harel and B. Rumpe, "Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff," Weizmann Institute Of Science, Jerusalem, Israel, Israel, Tech. Rep. MCS00-16, 2000.

[Harel87] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, 1987.

[Hausmann et al.01] J. H. Hausmann, R. Heckel, and S. Sauer, "Towards Dynamic Meta Modeling of UML Extensions: An Extensible Semantics for UML Sequence Diagrams," in *2002 IEEE CS International Symposium on Human-Centric Computing Languages and Environments (HCC 2001), September 5-7, 2001 Stresa, Italy*. IEEE Computer Society, 2001, pp. 80–87.

[Hausmann et al.04] J. H. Hausmann, R. Heckel, and S. Sauer, "Dynamic Meta Modeling with time: Specifying the semantics of multimedia sequence diagrams," *Software and System Modeling*, vol. 3, no. 3, pp. 181–193, 2004.

[Hayes et al.85] P. J. Hayes, P. A. Szekely, and R. A. Lerner, “Design alternatives for user interface management sytems based on experience with COUSIN,” in *CHI ’85: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 1985, pp. 169–175.

[Hayward et al.04] V. Hayward, O. R. Astley, M. Cruz-Hernandez, D. Grant, and G. Robles-De-La-Torre, “Haptic interfaces and devices,” *Sensor Review*, vol. 24, no. 1, pp. 16–29, 2004.

[Heller et al.01] R. S. Heller, C. D. Martin, N. Haneef, and S. Gievska-Krliu, “Using a theoretical multimedia taxonomy framework,” *J. Educ. Resour. Comput.*, vol. 1, p. 6, 2001.

[Hennicker and Koch01] R. Hennicker and N. Koch, “Modeling the User Interface of Web Applications with UML,” in *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*, ser. LNI, A. Evans, R. B. France, A. M. D. Moreira, and B. Rumpe, Eds., vol. 7. GI, 2001, pp. 158–172.

[Henning01] P. A. Henning, *Taschenbuch Multimedia.* Fachbuchverlag Leipzig, 6 2001. URL: http://amazon.com/o/ASIN/3446217517/

[Hettel et al.08] T. Hettel, M. Lawley, and K. Raymond, “Model Synchronisation: Definitions for Round-Trip Engineering,” in *Theory and Practice of Model Transformations, First International Conference, ICMT 2008, Zürich, Switzerland, July 1-2, 2008, Proceedings*, ser. Lecture Notes in Computer Science, A. Vallecillo, J. Gray, and A. Pierantonio, Eds., vol. 5063. Springer, 2008, pp. 31–45.

[Hewett et al.92] T. T. Hewett, R. Baecker, S. Card, T. Carey, J. Gasen, M. Mantei, G. Perlman, G. Strong, and W. Verplank, “ACM SIGCHI curricula for human-computer interaction,” ACM, New York, NY, USA, Tech. Rep., 1992, chairman-Thomas T. Hewett.

[Hilliges et al.06] O. Hilliges, P. Holzer, R. Klüber, and A. Butz, “AudioRadar: A metaphorical visualization for the navigation of large music collections,” in *Proceedings of the International Symposium on Smart Graphics 2006, Vancouver Canada*, 2006.

[Hirakawa99] M. Hirakawa, “Do software engineers like multimedia?” in *Multimedia Computing and Systems, 1999. IEEE International Conference on*, vol. 1, 7-11 June 1999, pp. 85–90vol.1.

[Hitz et al.05] M. Hitz, G. Kappel, E. Kapsammer, and W. Retschitzegger, *UML @ Work*, 3rd ed. Heidelberg: Dpunkt Verlag, 2005.

[Hoogeveen97] M. Hoogeveen, “Towards a Theory of the Effectiveness of Multimedia Systems,” *International Journal of Human Computer Interaction*, vol. 9, no. 2, pp. 151–168, 1997. URL: http://www.cyber-ventures.com/mh/paper/mmtheory.htm

[Hußmann and Pleuß04] H. Hußmann and A. Pleuß, “Model-Driven Development of Multimedia Applications,” in *The Monterey Workshop 2004 – Workshop on Software Engineering Tools: Compatibility and Integration*, 2004. URL: http://www.medien.ifi.lmu.de/pubdb/publications/pub/hussmann2004monterey/hussmann2004monterey.pdf

[Hussmann07] H. Hussmann, “Vorlesung Multimedia-Programmierung,” pp. 2–50, 6 2007. URL: http://www.medien.ifi.lmu.de/lehre/ss07/mmp/vorlesung/mmp3b.pdf

[Hyp] "Apple HyperCard," [Website]. URL: http://www.apple.com/hypercard/

[IBM] IBM, "Rational Software Modeler," [Website]. URL: http://www-01.ibm.com/software/awdtools/modeler/swmodeler/index.html

[ICS] "ACM/IEEE International Conference on Software Engineering (ICSE)," [Website]. URL: http://www.icse-conferences.org/

[Ide] "UsiXML – IdealXML," [Website]. URL: http://www.usixml.org/index.php?mod=pages&id=15

[Ill] "Adobe Illustrator CS4," [Website]. URL: http://www.adobe.com/products/illustrator/

[Inf] "Automotive Infotainment Systems," [Website]. URL: http://www.smsc-ais.com/AIS/

[INR05] *ATL UML to Java*, INRIA, March 2005. URL: http://www.eclipse.org/m2m/atl/atlTransformations/UML2Java/ExampleUML2Java%5Bv00.01%5D.pdf

[INT] "The INTERACT Conference Series," [Website]. URL: http://www.ifip-hci.org/INTERACT.html

[Isakowitz et al.98] T. Isakowitz, A. Kamis, and M. Koufaris, "Reconciling Top-Down and Bottom-Up Design Approaches in RMM," *DATA BASE*, vol. 29, no. 4, pp. 58–67, 1998.

[Isazadeh and Lamb97] H. Isazadeh and D. A. Lamb, "CASE Environments and MetaCASE Tools," Queen's University, Canada, Tech. Rep. 1997-403, 1997. URL: http://ftp.qucis.queensu.ca/TechReports/Reports/1997-403.pdf

[ISO88] *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*, ISO, 1988, iSO/IS 8807.

[ISO97a] *The Virtual Reality Modeling Language 1.0*, ISO, 1997, iSO/IEC 14772-1:1997. URL: http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/

[ISO97b] *Information technology – Coding of multimedia and hypermedia information – Part 1: MHEG object representation – Base notation (ASN.1)*, ISO/IEC, Geneva, Switzerland, 1997, iSO/IEC 13522-1:1997.

[ISO98] *Ergonomics of Human System Interaction – Part 11: Guidance on usability*, ISO, 1998, iSO 9241-11.

[ISO04] *ISO/IEC 19775:2004 – Extensible 3D (X3D)*, ISO, 2004, iSO/IEC 19775:2004. URL: http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/

[Jacobson et al.99] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Addison-Wesley Professional, 2 1999. URL: http://amazon.com/o/ASIN/0201571692/

[Janssen et al.93] C. Janssen, A. Weisbecker, and J. Ziegler, "Generating user interfaces from data models and dialogue net specifications," in *Human-Computer Interaction, INTERACT '93, IFIP TC13 International Conference on Human-Computer Interaction, 24-29*

April 1993, Amsterdam, The Netherlands, jointly organised with ACM Conference on Human Aspects in Computing Systems CHI'93, S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. N. White, Eds. ACM, 1993, pp. 418–423.

[Jav] "JavaCC," [Website]. URL: https://javacc.dev.java.net/

[JDT] "Eclipse Java development tools (JDT)," [Website]. URL: http://www.eclipse.org/jdt/

[Jeckle et al.04] M. Jeckle, C. Rupp, J. Hahn, B. Zengler, and S. Queins, *UML 2 glasklar*. München, Wien: Carl Hanser Verlag, 2004.

[Jeckle04] M. Jeckle, "Unified Modeling Language (UML) Tools," 2004, [Website]. URL: http://www.jeckle.de/umltools.htm

[JET] "Eclipse – Model to Text (M2T) project – JET," [Website]. URL: http://www.eclipse.org/modeling/m2t/?project=jet

[JJT] "JavaCC: JJTree Reference Documentation," [Website]. URL: https://javacc.dev.java.net/doc/JJTree.html

[JMe] "What is JMerge," [Website]. URL: http://wiki.eclipse.org/JET_FAQ_What_is_JMerge%3F

[Johnson and Johnson89] P. Johnson and H. Johnson, "Knowledge analysis of task : Task analysis and specification for human-computer systems," in *Engineering the human-computer interface*, A. Downton, Ed. New York, NY, USA: McGraw-Hill, Inc., 1989. URL: http://portal.acm.org/citation.cfm?id=141687

[Johnson91] P. Johnson, *Human Computer Interaction*. McGraw-Hill Publishing Co., 12 1991. URL: http://amazon.com/o/ASIN/0077072359/

[Jorge et al.03] J. A. Jorge, N. J. Nunes, and J. ao Falcão e Cunha, Eds., *Interactive Systems. Design, Specification, and Verification, 10th International Workshop, DSV-IS 2003, Funchal, Madeira Island, Portugal, June 11-13, 2003, Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2844. Springer, 2003.

[Jouault and Bézivin06] F. Jouault and J. Bézivin, "KM3: A DSL for Metamodel Specification," in *Formal Methods for Open Object-Based Distributed Systems, 8th IFIP WG 6.1 International Conference, FMOODS 2006, Bologna, Italy, June 14-16, 2006, Proceedings*, ser. Lecture Notes in Computer Science, R. Gorrieri and H. Wehrheim, Eds., vol. 4037. Springer, 2006, pp. 171–185.

[Jouault and Kurtev05] F. Jouault and I. Kurtev, "Transforming Models with ATL," in *Satellite Events at the MoDELS 2005 Conference, MoDELS 2005 International Workshops, Doctoral Symposium, Educators Symposium, Montego Bay, Jamaica, October 2-7, 2005, Revised Selected Papers*, ser. Lecture Notes in Computer Science, J.-M. Bruel, Ed., vol. 3844. Springer, 2005, pp. 128–138.

[Jouault and Kurtev06] F. Jouault and I. Kurtev, "On the architectural alignment of ATL and QVT," in *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2006, pp. 1188–1195.

[Jungwirth and Stadler03] H. Jungwirth and H. Stadler, *Ansichten - Videoanalysen zur Lehrer/-innenbildung (CD-ROM).* Innsbruck, Austria: Studienverlag, 2003.

[Kaczkowski07] M. Kaczkowski, "Realisierung einer Transformation von MML-Modellen nach Java-Code," Project Thesis, University of Munich, Munich, July 2007.

[Kannengiesser and Kannengiesser06] C. Kannengiesser and M. Kannengiesser, *Flash 8*, 2nd ed. Poing, Germany: Franzis Verlag GmbH, 5 2006. URL: http://amazon.de/o/ASIN/3772370950/

[Kappel et al.03] G. Kappel, B. Pröll, S. Reich, and W. Retschitzegger, *Web Engineering. Systematische Entwicklung von Webanwendungen.* Heidelberg: Dpunkt Verlag, 2003.

[Kauntz07] G. Kauntz, "MMLbasierte Entwicklung eines Autorenwerkzeugs für Lernanwendungen der Unterrichtsmitschau," Project Thesis, University of Munich, Munich, Mai 2007.

[Kazoun and Lott07] C. Kazoun and J. Lott, *Programming Flex 2: The comprehensive guide to creating rich media applications with Adobe Flex (Programming).* Sebastopol, CA, USA: OŠReilly, 4 2007. URL: http://amazon.com/o/ASIN/059652689X/

[Kelly and Tolvanen08] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation.* New York, NY, USA: John Wiley & Sons, Inc., 3 2008. URL: http://amazon.com/o/ASIN/0470036664/

[Kent05] S. Kent, "A DSL or UML Profile. Which Would You Use? - Panel Discussion," in *Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005, Proceedings*, ser. Lecture Notes in Computer Science, L. C. Briand and C. Williams, Eds., vol. 3713. Springer, 2005, p. 719.

[Kieburtz et al.96] R. B. Kieburtz, L. McKinney, J. M. Bell, J. Hook, A. Kotov, J. Lewis, D. P. Oliva, T. Sheard, I. Smith, and L. Walton, "A software engineering experiment in software component generation," in *ICSE '96: Proceedings of the 18th international conference on Software engineering*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 542–552.

[Kitchenham et al.02] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 721–734, 2002.

[Kleppe et al.03] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional, 5 2003. URL: http://amazon.com/o/ASIN/032119442X/

[Klußmann01] N. Klußmann, *Lexikon der Kommunikations- und Informationstechnik*, 3rd ed. Heidelberg, Germany: Hüthig, 10 2001. URL: http://amazon.de/o/ASIN/3778539515/

[Koch et al.07] N. Koch, A. Knapp, G. Zhang, and H. Baumeister, "Uml-Based Web Engineering: An Approach Based on Standards," in *Web Engineering: Modelling and Implementing Web Applications*, G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, Eds. London, UK: Springer, 2007, pp. 157–191.

[Kolski and Vanderdonckt02] C. Kolski and J. Vanderdonckt, Eds., *Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, May, 15-17, 2002, Valenciennes, France*. Kluwer, 2002.

[Kozel96] K. Kozel, "The Interactive Killing Fields," *Multimedia Producer*, no. 5, May 1996.

[Kraiker07] S. Kraiker, "Klassifikation von Multimedia-Anwendungen bezüglich der Modellierung mit MML," Project Thesis, University of Munich, Munich, February 2007.

[Krasner and Pope88] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80," *J. Object Oriented Program.*, vol. 1, no. 3, pp. 26–49, 1988.

[Kühne07] T. Kühne, Ed., *Models in Software Engineering, Workshops and Symposia at MoDELS 2006, Genoa, Italy, October 1-6, 2006, Reports and Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 4364. Springer, 2007.

[Kurtev et al.02] I. Kurtev, J. Bézivin, and M. Aksit, "Technological Spaces: An Initial Appraisal," in *CoopIS, DOA'2002 Federated Conferences, Industrial track*, Irvine, 2002. URL: http://fparreiras/papers/TechnologicalSpaces.pdf

[Kurtev et al.06] I. Kurtev, J. Bézivin, F. Jouault, and P. Valduriez, "Model-based DSL frameworks," in *Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, Oregon, USA*, P. L. Tarr and W. R. Cook, Eds. ACM, 2006, pp. 602–616.

[Lang and Fitzgerald05] M. Lang and B. Fitzgerald, "Hypermedia Systems Development Practices: A Survey," *IEEE Softw.*, vol. 22, no. 2, pp. 68–75, 2005.

[Lang and Fitzgerald06] M. Lang and B. Fitzgerald, "New Branches, Old Roots: A Study of Methods and Techniques in Web/Hypermedia Systems Design," *Information Systems Management*, vol. 23, no. 3, pp. 62–74, June 2006.

[Lang01a] M. Lang, "Issues and Challenges in the Development of Hypermedia Information Systems," in *Proceedings of 11th Annual Business Information Technology Conference (BIT 2001)*, R. Hackney, Ed. Machester, UK: Manchester Metropolitan University, 2001.

[Lang01b] M. Lang, "A Study of Practice in Hypermedia Systems Design," in *Doctoral Consortium, European Conference on Information Systems (ECIS)*, 2001. URL: http://ecis2001.fov.uni-mb.si/doctoral/Students/ECIS-DC_Lang.pdf

[Leichtenstern04] K. Leichtenstern, "Automatische Generierung von SVG/JavaScript-Code für Multimedia-Anwendungen," Project Thesis, University of Munich, Munich, December 2004.

[Lienhart et al.07] R. Lienhart, A. R. Prasad, A. Hanjalic, S. Choi, B. P. Bailey, and N. Sebe, Eds., *Proceedings of the 15th International Conference on Multimedia 2007, Augsburg, Germany, September 24-29, 2007*. ACM, 2007.

[Limbourg et al.00] Q. Limbourg, J. Vanderdonckt, and N. Souchon, "The Task-Dialog and Task-Presentation Mapping Problem: Some Preliminary Results," in *DSV-IS*, 2000, pp. 227–246.

[Limbourg et al.01] Q. Limbourg, C. Pribeanu, and J. Vanderdonckt, "Towards Uniformed Task Models in a Model-Based Approach," in *Interactive Systems: Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001, Glasgow, Scotland, UK, June 13-15, 2001, Revised Papers*, ser. Lecture Notes in Computer Science, C. Johnson, Ed., vol. 2220. Springer, 2001, pp. 164–182.

[Limbourg et al.04] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and V. López-Jaquero, "USIXML: A Language Supporting Multi-path Development of User Interfaces," in *Engineering Human Computer Interaction and Interactive Systems, Joint Working Conferences EHCI-DSVIS 2004, Hamburg, Germany, July 11-13, 2004, Revised Selected Papers*, ser. Lecture Notes in Computer Science, R. Bastide, P. A. Palanque, and J. Roth, Eds., vol. 3425. Springer, 2004, pp. 200–220.

[Limbourg04] Q. Limbourg, "Multi-Path Development of User Interfaces," Phd, Université catholique de Louvain, Louvain-La-Neuve, 2004.

[Linaje et al.07] M. Linaje, J. C. Preciado, and F. Sánchez-Figueroa, "Engineering Rich Internet Application User Interfaces over Legacy Web Models," *IEEE Internet Computing*, vol. 11, no. 6, pp. 53–59, 2007.

[Lonczewski and Schreiber96] F. Lonczewski and S. Schreiber, "The FUSE-System: an Integrated User Interface Design Environment," in *Computer-Aided Design of User Interfaces I, Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces, June 5-7, 1996, Namur, Belgium*, J. Vanderdonckt, Ed. Presses Universitaires de Namur, 1996, pp. 37–56.

[Long et al.98] E. Long, A. Misra, and J. Sztipanovits, "Increasing productivity at Saturn," *Computer*, vol. 31, no. 8, pp. 35–43, 1998.

[Lord07] R. Lord, "Polling the keyboard in Actionscript 3," September 2007, [Website]. URL: http://www.bigroom.co.uk/blog/polling-the-keyboard-in-actionscript-3

[Lorenz and Schmalfuß98] M. Lorenz and R. Schmalfuß, "Multimedia Authoring Systems: A Proposal for a Reference Model," in *The Sixth International Conference in Central Europe on Computer Graphics and Visualization (WSCG'98)*, 1998. URL: http://wscg.zcu.cz/WSCG1998/papers98/Lorenz_98.ps.gz

[Lowe and Hall99] D. Lowe and W. Hall, *Hypermedia and the Web: An Engineering Approach*. New York, NY, USA: John Wiley & Sons, Inc., 1999.

[Luna08] F. D. Luna, *Introduction to 3D Game Programming with DirectX 10*. Plexo, TX, USA: Wordware Publishing, Inc., 10 2008. URL: http://amazon.com/o/ASIN/1598220535/

[Luyten and Coninx05] K. Luyten and K. Coninx, "Distributed User Interface Elements to support Smart Interaction Spaces," in *Seventh IEEE International Symposium on Multimedia (ISM 2005), 12-14 December 2005, Irvine, CA, USA*. IEEE Computer Society, 2005, pp. 277–286.

[Luyten et al.03] K. Luyten, T. Clerckx, K. Coninx, and J. Vanderdonckt, “Derivation of a Dialog Model from a Task Model by Activity Chain Extraction,” in *Interactive Systems. Design, Specification, and Verification, 10th International Workshop, DSV-IS 2003, Funchal, Madeira Island, Portugal, June 11-13, 2003, Revised Papers*, ser. Lecture Notes in Computer Science, J. A. Jorge, N. J. Nunes, and J. ao Falcão e Cunha, Eds., vol. 2844. Springer, 2003, pp. 203–217.

[Luyten04] K. Luyten, “Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development,” Phd, Transnationale Universiteit Limburg, Diepenbeek, Belgium, October 2004. URL: http://research.edm.uhasselt.be/~kris/research/phd/phd-luyten.pdf

[Macromedia03] Macromedia, Ed., *Macromedia Flash MX 2004 ActionScript 2.0 Dictionary*. Berkeley, CA, USA: Macromedia Press, 11 2003. URL: http://amazon.com/o/ASIN/0321228413/

[Mallon95] A. Mallon, “The Multimedia Development Process,” 1995. URL: http://ourworld.compuserve.com/homepages/adrian_mallon_multimedia/devmtpro.htm

[Marculescu et al.04] R. Marculescu, M. Pedram, and J. Henkel, “Distributed Multimedia System Design: A Holistic Perspective,” in *DATE ’04: Proceedings of the conference on Design, automation and test in Europe*. Washington, DC, USA: IEEE Computer Society, 2004, p. 21342.

[Markopoulos and Marijnissen00] P. Markopoulos and P. Marijnissen, “UML as a Representation for Interaction Designs,” in *Proc. Australian Conf. Computer-Human Interaction (OZCHI)*, 2000, pp. 240–249. URL: http://www.idemployee.id.tue.nl/p.markopoulos/downloadablePapers/P.MarkopoulosOZCHI2000.pdf

[Markopoulos et al.92] P. Markopoulos, J. Pycock, S. Wilson, and P. Johnson, “Adept-a task based design environment,” in *Proc. Twenty-Fifth Hawaii International Conference on System Sciences*, J. Pycock, Ed., vol. ii, 1992, pp. 587–596 vol.2.

[Marshall08] D. Marshall, “Multimedia – Online Course Notes – Multimedia and Hypermedia Information Encoding Expert Group (MHEG),” Cardiff University, 2008. URL: http://www.cs.cf.ac.uk/Dave/Multimedia/node297.html#SECTION04360000000000000000

[Märtin96] C. Märtin, “Software Life Cycle Automation for Interactive Applications: The AME Design Environment,” in *Computer-Aided Design of User Interfaces I, Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces, June 5-7, 1996, Namur, Belgium*, J. Vanderdonckt, Ed. Presses Universitaires de Namur, 1996, pp. 57–76.

[Martinez-Ruiz et al.06a] F. J. Martinez-Ruiz, J. M. Arteaga, and J. Vanderdonckt, “Transformation of XAML schema for RIA using XSLT,” in *Proc. of XlX Congreso Nacional y V Congreso Internacional de Informática y Computación de la ANIEI, Avances en Tecnologías de la Información CNCIIC’2006*, 2006. URL: http://www.isys.ucl.ac.be/bchi/publications/2006/Martinez-CNCIIC2006.pdf

[Martinez-Ruiz et al.06b] F. J. Martinez-Ruiz, J. M. Arteaga, J. Vanderdonckt, J. M. Gonzalez-Calleros, and R. Mendoza, "A first draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications," in *LA-WEB '06: Proceedings of the Fourth Latin American Web Congress*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 32–38.

[May] "Autodesk Maya," [Website]. URL: http://usa.autodesk.com/adsk/servlet/index?id=7635018&siteID=123112

[Mbakop06] X. Mbakop, "Erstellung eines MML Editors mittels der openArchitectureWare," Project Thesis, University of Munich, Munich, January 2006.

[Mbakop07] X. Mbakop, "Reverse-Engineering von MML-Modellen aus Flash-Anwendungen," Diploma Thesis, University of Munich, Munich, March 2007.

[Mehra et al.05] A. Mehra, J. Grundy, and J. Hosking, "A generic approach to supporting diagram differencing and merging for collaborative design," in *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. New York, NY, USA: ACM, 2005, pp. 204–213.

[Meta] "MetaCase - Domain-Specific Modeling with MetaEdit+," [Website]. URL: http://www.metacase.com/

[Metb] "Metacase - Examples on Domain-Specific Modeling," january 30, 2008. URL: http://www.metacase.com/cases/dsm_examples.html

[MetaCase99] MetaCase, "Nokia Case Study," 1999. URL: http://www.metacase.com/papers/MetaEdit_in_Nokia.pdf

[Meyer-Boudnik and Effelsberg95] T. Meyer-Boudnik and W. Effelsberg, "MHEG Explained," *IEEE MultiMedia*, vol. 2, no. 1, pp. 26–38, 1995.

[Meyer08] R. Meyer, "MML Code Generation for ActionScript 3," Project Thesis, University of Munich, Munich, 11 2008.

[MHE] "MHEG-5 Class Hierarchy," [Website]. URL: http://user.chol.com/~mheg5/down/class.pdf

[Mic] "Microsoft Corporation," [Website]. URL: www.microsoft.com/

[Michotte and Vanderdonckt08] B. Michotte and J. Vanderdonckt, "GrafiXML, A Multi-Target User Interface Builder based on UsiXML," in *Proc. of 4th International Conference on Autonomic and Autonomous Systems ICASŠ2008*. IEEE Computer Society Press, 2008.

[Microsofta] Microsoft, "DirectX Developer Center," [Website]. URL: http://msdn.microsoft.com/en-us/directx/default.aspx

[Microsoftb] Microsoft, "Microsoft MSDN Library – IDirectInputDevice8 Interface," [Website]. URL: http://msdn.microsoft.com/en-us/library/bb205956(VS.85).aspx

[Microsoft03] Microsoft, Ed., *Computerlexikon mit Fachwörterbuch*, 7th ed. UnterschleiSSheim, Germany: Microsoft Press, 2003.

[Microsystems] S. Microsystems, “Java Media APIs,” [Website]. URL: http://java.sun.com/javase/technologies/desktop/media/

[Miller and (Eds.)03] J. Miller and J. M. (Eds.), *MDA Guide Version 1.0.1*, Object Management Group, June 2003, omg/2003-06-01.

[Misanchuk et al.00] E. R. Misanchuk, R. A. Schwier, and E. Boling, *Visual design for instructional multimedia*. CD-ROM hypertextbook, 2000. URL: http://www.indiana.edu/~vdim/Start.HTM

[MMM] “International Multimedia Modeling Conference 2008.” URL: http://research.nii.ac.jp/mmm2008/

[MMPa] “University of Munich – Multimedia-Programmierung Summer Semester 2004,” january 28, 2008. URL: http://www.medien.ifi.lmu.de/lehre/ss04/mmp/

[MMPb] “University of Munich – Multimedia-Programmierung Summer Semester 2004 – Results,” january 28, 2008. URL: http://www.medien.ifi.lmu.de/fileadmin/mimuc/mmp_ss04/results/mmp_results.swf

[MMPc] “University of Munich – Multimedia-Programmierung Summer Semester 2005.” URL: http://www.medien.ifi.lmu.de/lehre/ss05/mmp/

[MMPd] “University of Munich – Multimedia-Programmierung Summer Semester 2006.” URL: http://www.medien.ifi.lmu.de/lehre/ss06/mmp/

[MMPe] “University of Munich – Multimedia-Programmierung Summer Semester 2006 – Results.” URL: http://www.ifi.lmu.de/studium-medieninformatik/galerie/mmp-ss06/

[MMPf] “University of Munich – Multimedia-Programmierung Summer Semester 2007.” URL: http://www.medien.ifi.lmu.de/lehre/ss07m/mmp/

[Montero et al.05] F. Montero, V. López-Jaquero, J. Vanderdonckt, P. González, M. D. Lozano, and Q. Limbourg, “Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML,” in *Interactive Systems, Design, Specification, and Verification, 12th International Workshop, DSVIS 2005, Newcastle upon Tyne, UK, July 13-15, 2005, Revised Papers*, ser. Lecture Notes in Computer Science, S. W. Gilroy and M. D. Harrison, Eds., vol. 3941. Springer, 2005, pp. 161–172.

[Montero05] F. Montero, “Quality and experience integration in the model-driven user interfaces development process (Slides),” 2005. URL: http://www.isys.ucl.ac.be/bchi/publications/Ph.D.Theses/Montero-PhD2005.ppt

[Moock04] C. Moock, *Essential ActionScript 2.0*, 1st ed. Sebastopol, CA, USA: O’Reilly Media, 6 2004. URL: http://amazon.de/o/ASIN/0596006527/

[Moock07] C. Moock, *Essential ActionScript 3.0*, 1st ed. Sebastopol, CA, USA: O’Reilly Media, 7 2007. URL: http://amazon.de/o/ASIN/0596526946/

[Mori et al.04] G. Mori, F. Paternò, and C. Santoro, “Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions,” *IEEE Trans. Software Eng.*, vol. 30, no. 8, pp. 507–520, 2004.

[Morris and Finkelstein96] S. J. Morris and A. C. W. Finkelstein, "Integrating Design and Development in the Production of Multimedia Documents," in *MMSD '96: Proceedings of the 1996 International Workshop on Multimedia Software Development (MMSD '96)*. Washington, DC, USA: IEEE Computer Society, 1996, p. 98.

[Motion-Twin] Motion-Twin, "MTASC – Motion-Twin ActionScript 2 Compiler," [Website]. URL: http://www.mtasc.org/

[Moyes and Jordan93] J. Moyes and P. W. Jordan, "Icon design and its effect on guessability, learnability, and experienced user performance," in *People and Computers VIII*, L. Alty, D. Diaper, and S. Guest, Eds. Cambridge, England: Cambridge University Press, 1993. URL: http://books.google.de/books?hl=en&lr=&id=Jzwbj9B1vpQC&oi=fnd&pg=PA49&dq=icon+design&ots=1t0zrEbkIp&sig=3YWMa_bs0BwcXxDy-4YDhyq0jh4

[Moz] "Mozilla Project," [Website]. URL: http://www.mozilla.org/

[Mühlhäuser and Gecsei96] M. Mühlhäuser and J. Gecsei, "Services, Frameworks, and Paradigms for Distributed Multimedia Applications," *IEEE MultiMedia*, vol. 3, no. 3, pp. 48–61, 1996.

[Mühlhäuser96] M. Mühlhäuser, "Issues in Multimedia Software Development," in *MMSD '96: Proceedings of the 1996 International Workshop on Multimedia Software Development (MMSD '96)*. Washington, DC, USA: IEEE Computer Society, 1996, p. 2.

[Murata and Murata89] T. Murata and T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[Myers et al.00] B. A. Myers, S. E. Hudson, and R. F. Pausch, "Past, present, and future of user interface software tools," *ACM Trans. Comput.-Hum. Interact.*, vol. 7, no. 1, pp. 3–28, 2000.

[Narasimhalu96] A. D. Narasimhalu, "Multimedia databases," *Multimedia Syst.*, vol. 4, no. 5, pp. 226–249, 1996.

[Nielsen93] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[No Magic] I. No Magic, "MagicDraw UML," [Website]. URL: http://www.magicdraw.com/

[Noble et al.99] J. Noble, A. Taivalsaari, and I. Moore, Eds., *Prototype-Based Programming: Concepts, Languages and Applications*, 1st ed. Singapore: Springer, 2 1999.

[Nóbrega et al.05] L. Nóbrega, N. J. Nunes, and H. Coelho, "Mapping ConcurTaskTrees into UML 2.0," in *Interactive Systems, Design, Specification, and Verification, 12th International Workshop, DSVIS 2005, Newcastle upon Tyne, UK, July 13-15, 2005, Revised Papers*, ser. Lecture Notes in Computer Science, S. W. Gilroy and M. D. Harrison, Eds., vol. 3941. Springer, 2005, pp. 237–248.

[Nunes and Falcão e Cunha00] N. J. Nunes and J. Falcão e Cunha, "Towards a UML profile for interaction design: the Wisdom approach," in *UML 2000 - The Unified Modeling Language,* *Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000, Proceedings*, ser. Lecture Notes in Computer Science, A. Evans, S. Kent, and B. Selic, Eds., vol. 1939. Springer, 2000, pp. 101–116.

[Nunes01] N. J. Nunes, "Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach," Ph.D. dissertation, University of Madrid, Madrid, Spain, 2001.

[Oatrix] Oatrix, "GRiNS," [Website]. URL: http://www.oratrix.com/

[oAW] "openArchitectureWare," [Website]. URL: http://www.openarchitectureware.org/

[Obj99] *Requirements for UML Profiles*, Object Management Group (OMG), 1999, oMG document ad/99-12-32.

[Obj03] *Common Warehouse Metamodel (CWM) Specification, Version 1.1*, Object Management Group, February 2003, formal/2003-03-02. URL: http://www.omg.org/cgi-bin/apps/doc?formal/03-03-02.pdf

[Obj04a] *Metmodel and UML Profile for Java and EJB Specification*, Object Management Group (OMG), February 2004, formal/04-02-02. URL: http://www.omg.org/docs/formal/04-02-02.pdf

[Obj04b] *An ORMSC Definition of MDA*, Object Management Group (OMG), August 2004, ormsc/04-08-02. URL: http://www.omg.org/docs/ormsc/04-08-02.pdf

[Obj05] *Meta Object Facility (MOF) Specification, Version 1.4.1*, Object Management Group, July 2005, formal/05-05-05. URL: http://www.omg.org/docs/formal/05-05-05.pdf

[Obj06a] *Meta Object Facility (MOF) Core Specification, Version 2.0*, Object Management Group, January 2006, formal/06-01-01. URL: http://www.omg.org/docs/formal/06-01-01.pdf

[Obj06b] *Object Constraint Language OMG Available Specification Version 2.0*, Object Management Group, May 2006, formal/2006-05-01. URL: http://www.omg.org/docs/formal/06-05-01.pdf

[Obj07a] *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*, Object Management Group, July 2007, ptc/07-07-07. URL: http://www.omg.org/docs/ptc/07-07-07.pdf

[Obj07b] *MOF 2.0/XMI Mapping, Version 2.1.1*, Object Management Group, December 2007, formal/2007-12-01. URL: http://www.omg.org/docs/formal/07-12-01.pdf

[Obj07c] *OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2*, Object Management Group, November 2007, formal/2007-11-04. URL: http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF

[Obj07d] *OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2*, Object Management Group, November 2007, formal/2007-11-02. URL: http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF

[Obj07e] *Ontology Definition Metamodel*, Object Management Group, 2007, ptc/2007-09-09. URL: http://www.omg.org/docs/ptc/07-09-09.pdf

[of Maryland] U. of Maryland, “Piccolo Home Page,” [Website]. URL: http://www.cs.umd.edu/hcil/jazz/

[Oldevik et al.08] J. Oldevik, G. K. Olsen, T. Neple, and R. Paige, Eds., *ECMDA Traceability Workshop Proceedings*, 2008. URL: http://modelbased.net/ecmda-traceability/images/papers/2008/ecmda-tw-proceedings08.pdf

[Olsen07] D. R. Olsen, Jr., “Evaluating user interface systems research,” in *UIST ’07: Proceedings of the 20th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM, 2007, pp. 251–258.

[OMGa] “Object Management Group (OMG) – Object and Reference Model Subcommittee (ORMSC) – MDA Guide Working Page,” [Website]. URL: http://ormsc.omg.org/mda_guide_working_page.htm

[OMGb] “Object Management Group (OMG) Website,” january 23, 2008. URL: http://www.omg.org/

[Osswald03] K. Osswald, *Konzeptmanagement - Interaktive Medien - Interdisziplinäre Projekte*. Berlin Heidelberg: Springer, 2003.

[Otr] “Otris Software AG – JANUS,” [Website]. URL: http://www.otris.de/cms/JANUS_Softwarefabrik_MDA_otris.html

[Oviatt99] S. Oviatt, “Ten myths of multimodal interaction,” *Commun. ACM*, vol. 42, no. 11, pp. 74–81, 1999.

[Ozcan et al.05] G. Ozcan, C. Isikhan, and A. Alpkocak, “Melody Extraction on MIDI Music Files,” in *Seventh IEEE International Symposium on Multimedia (ISM 2005), 12-14 December 2005, Irvine, CA, USA*. IEEE Computer Society, 2005, pp. 414–422.

[Pai] “Corel Paint Shop Pro,” [Website]. URL: http://www.corel.com/servlet/Satellite/us/en/Product/1184951547051#versionTabview=tab0&tabview=tab0

[Paige et al.00] R. F. Paige, J. S. Ostroff, and P. J. Brooke, “Principles for modeling language design,” *Information & Software Technology*, vol. 42, no. 10, pp. 665–675, 2000.

[Palanque and Bastide95] P. A. Palanque and R. Bastide, Eds., *Design, Specification and Verification of Interactive Systems ’95, Proceedings of the Eurographics Workshop in Toulouse, France June 7-9, 1995*. Springer, 1995.

[Palanque et al.93] P. A. Palanque, R. Bastide, L. Dourte, and C. Sibertin-Blanc, “Design of User-Driven Interfaces Using Petri Nets and Objects,” in *CAiSE ’93: Proceedings of Advanced Information Systems Engineering*. London, UK: Springer-Verlag, 1993, pp. 569–585.

[Paternò and Sansone06] F. Paternò and S. Sansone, “Model-based Generation of Interactive Digital TV Applications,” in *Proceedings of the MoDELS’06 Workshop on Model Driven Development of Advanced User Interfaces*, A. Pleuß, J. V. den Bergh, S. Sauer, H. Hußmann, and A. Bödcher, Eds., vol. 214. CEUR-WS.org, 2006.

[Paternò and Santoro02] F. Paternò and C. Santoro, "One Model, Many Interfaces," in *Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, May, 15-17, 2002, Valenciennes, France*, C. Kolski and J. Vanderdonckt, Eds. Kluwer, 2002, pp. 143–154.

[Paternò et al.97] F. Paternò, C. Mancini, and S. Meniconi, "ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models," in *Human-Computer Interaction, INTERACT '97, IFIP TC13 Interantional Conference on Human-Computer Interaction, 14th-18th July 1997, Sydney, Australia*, ser. IFIP Conference Proceedings, S. Howard, J. Hammond, and G. Lindgaard, Eds., vol. 96. Chapman & Hall, 1997, pp. 362–369.

[Paternò99] F. Paternò, *Model-Based Design and Evaluation of Interactive Applications*. London, UK: Springer-Verlag, 1999.

[Paternò01] F. Paternò, "Towards a UML for Interactive Systems," in *Engineering for Human-Computer Interaction, 8th IFIP International Conference, EHCI 2001, Toronto, Canada, May 11-13, 2001, Revised Papers*, ser. Lecture Notes in Computer Science, M. R. Little and L. Nigay, Eds., vol. 2254. Springer, 2001, pp. 7–18.

[Pauen and Voss98] P. Pauen and J. Voss, "The HyDev Approach to model-based Development of Hypermedia Applications," in *Proceedings Hypertext '98 Workshop, 1st International Workshop on Hypermedia Development: Processes, Methods and Models*, Pittsburgh, Juni 1998. URL: http://voss.fernuni-hagen.de/pi3/hydev/hypertext98/index.htm

[Pauen et al.98a] P. Pauen, J. Voss, and H.-W. Six, "Modeling Hypermedia Applications with HyDev," in *Designing Effective and Usable Multimedia Systems: Proceedings of the IFIP Working Group 13.2 Conference on Designing Effective and Usable Multimedia Systems, Stuttgart, Germany, September 1998*, ser. IFIP Conference Proceedings, A. G. Sutcliffe, J. Ziegler, and P. Johnson, Eds., vol. 133. Kluwer, 1998.

[Pauen et al.98b] P. Pauen, J. Voss, and H.-W. Six, "Modeling of Hypermedia Applications with HyDev," in *Proceedings CHI-98 Workshop Hyped-Media to Hyper-Media: Toward Theoretical Foundations of Design, Use and Evaluation*, Los Angeles, April 1998. URL: http://www.informatik.fernuni-hagen.de/import/pi3/PDFs/chi98.pdf

[Phanouriou00] C. Phanouriou, "UIML: A Device-Independent User Interface Markup Language," Phd, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA, September 2000. URL: http://scholar.lib.vt.edu/theses/available/etd-08122000-19510051/unrestricted/PhanouriouETD.pdf

[Phoa] "Adobe Photoshop," [Website]. URL: http://www.adobe.com/products/photoshop/photoshop/

[Phob] "Photoplay Games," january 25, 2008. URL: http://www.photoplay.com/

[Pla] "Planet MDE," january 30, 2008. URL: http://planetmde.org/

[Pleuß and Hußmann07] A. Pleuß and H. Hußmann, "Integrating Authoring Tools into Model-Driven Development of Interactive Multimedia Applications," in *Human-Computer Interaction. Interaction Design and Usability, 12th International Conference, HCI International 2007, Beijing, China, July 22-27, 2007, Proceedings, Part I*, ser. Lecture Notes in Computer Science, J. A. Jacko, Ed., vol. 4550. Springer, 2007, pp. 1168–1177.

[Pleuß et al.05a] A. Pleuß, J. V. den Bergh, H. Hußmann, and S. Sauer, Eds., *MDDAUI '05, Model Driven Development of Advanced User Interfaces 2005, Proceedings of the MoDELS'05 Workshop on Model Driven Development of Advanced User Interfaces, Montego Bay, Jamaica, October 2, 2005*, ser. CEUR Workshop Proceedings, vol. 159. CEUR-WS.org, 2005.

[Pleuß et al.05b] A. Pleuß, J. V. den Bergh, S. Sauer, and H. Hußmann, "Workshop Report: Model Driven Development of Advanced User Interfaces (MDDAUI)," in *Satellite Events at the MoDELS 2005 Conference, MoDELS 2005 International Workshops, Doctoral Symposium, Educators Symposium, Montego Bay, Jamaica, October 2-7, 2005, Revised Selected Papers*, ser. Lecture Notes in Computer Science, J.-M. Bruel, Ed., vol. 3844. Springer, 2005, pp. 182–190.

[Pleuß et al.06a] A. Pleuß, J. V. den Bergh, H. Hußmann, S. Sauer, and A. Bödcher, Eds., *MDDAUI '06, Model Driven Development of Advanced User Interfaces 2007, Proceedings of the MoDELS'06 Workshop on Model Driven Development of Advanced User Interfaces, Genova, Italy, October 2, 2006*, ser. CEUR Workshop Proceedings, vol. 214. CEUR-WS.org, 2006.

[Pleuß et al.06b] A. Pleuß, J. V. den Bergh, S. Sauer, H. Hußmann, and A. Bödcher, "Model Driven Development of Advanced User Interfaces (MDDAUI) - MDDAUI'06 Workshop Report," in *Models in Software Engineering, Workshops and Symposia at MoDELS 2006, Genoa, Italy, October 1-6, 2006, Reports and Revised Selected Papers*, ser. Lecture Notes in Computer Science, T. Kühne, Ed., vol. 4364. Springer, 2006, pp. 101–105.

[Pleuß et al.07a] A. Pleuß, J. V. den Bergh, H. Hußmann, S. Sauer, and D. Görlich, Eds., *MDDAUI '07, Model Driven Development of Advanced User Interfaces 2007, Proceedings of the MoDELS'07 Workshop on Model Driven Development of Advanced User Interfaces, Nashville, Tennessee, USA, October 1, 2007*, ser. CEUR Workshop Proceedings, vol. 297. CEUR-WS.org, 2007.

[Pleuß et al.07b] A. Pleuß, J. V. den Bergh, H. Hußmann, S. Sauer, and D. Görlich, Eds., *Proceedings of the MoDELS 2007 Workshop on Model Driven Development of Advanced User Interfaces, Nashville, Tennessee, USA, October 1, 2007*, ser. CEUR Workshop Proceedings, vol. 297. CEUR-WS.org, 2007.

[Pleuß et al.07c] A. Pleuß, J. V. den Bergh, S. Sauer, D. Görlich, and H. Hußmann, "Third International Workshop on Model Driven Development of Advanced User Interfaces," in *Models in Software Engineering, Workshops and Symposia at MoDELS 2007, Nashville, TN, USA, September 30 - October 5, 2007, Reports and Revised Selected Papers*, ser. Lecture Notes in Computer Science, H. Giese, Ed., vol. 5002. Springer, 2007, pp. 59–64.

[Pleuß02] A. Pleuß, "Werkzeugunterstützung für UML Profiles," Diploma Thesis, Technische Universität Dresden, Dresden; Germany, October 2002.

[Pleuß05a] A. Pleuß, "MML: A Language for Modeling Interactive Multimedia Applications," in *Seventh IEEE International Symposium on Multimedia (ISM 2005), 12-14 December 2005, Irvine, CA, USA*. IEEE Computer Society, 2005, pp. 465–473.

[Pleuß05b] A. Pleuß, "Modeling the User Interface of Multimedia Applications," in *Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005, Proceedings*, ser. Lecture Notes in Computer Science, L. C. Briand and C. Williams, Eds., vol. 3713. Springer, 2005, pp. 676–690.

[Pol08] "Game Programming Snippets – Keyboard Input: Polling System In Java," March 2008, [Website]. URL: http://gpsnippets.blogspot.com/2008/03/keyboard-input-polling-system-in-java.html

[Poppendieck and Poppendieck03] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional, 2003.

[Potter et al.96] B. Potter, J. Sinclair, and D. Till, *An introduction to formal specification and Z*. London, New York: Prentice-Hall, Inc., 1996.

[Powerflasher] Powerflasher, "FDT Development Tool for Flash," [Website]. URL: http://fdt.powerflasher.com/

[Pre] "Adobe Premiere," [Website]. URL: http://www.adobe.com/products/premiere/

[Preciado et al.05] J. C. Preciado, M. L. Trigueros, F. Sanchez, and S. Comai, "Necessity of methodologies to model Rich Internet Applications," in *Seventh IEEE International Workshop on Web Site Evolution (WSE 2005), 26 September 2005, Budapest, Hungary*. IEEE Computer Society, 2005, pp. 7–13.

[Preciado et al.07] J. C. Preciado, M. L. Trigueros, and F. Sánchez-Figueroa, "An approach to support the Web User Interfaces evolution," in *Proceedings of the 2nd International Workshop on Adaptation and Evolution in Web Systems Engineering AEWSE'07*, ser. CEUR-WS-Proc., S. Casteleyn, F. Daniel, P. Dolog, M. Matera, G.-J. Houben, and O. D. Troyer, Eds., vol. 267, 2007. URL: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-267/

[Preciado et al.08] J. Preciado, M. Linaje, R. Morales-Chaparro, F. Sanchez-Figueroa, G. Zhang, C. Kroiss, and N. Koch, "Designing Rich Internet Applications Combining UWE and RUX-Method," in *Proc. Eighth International Conference on Web Engineering ICWE '08*, 2008, pp. 148–154.

[Preece et al.94] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey, *Human-Computer Interaction: Concepts And Design (ICS)*. Addison Wesley, 1994.

[Puerta and Eisenstein99] A. R. Puerta and J. Eisenstein, "Towards a General Computational Framework for Model-Based Interface Development Systems," in *Intelligent User Interfaces*, 1999, pp. 171–178.

[Puerta and Maulsby97] A. R. Puerta and D. Maulsby, "Management of Interface Design Knowledge with MOBI-D," in *Intelligent User Interfaces*, 1997, pp. 249–252.

[Puerta96] A. R. Puerta, "The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development," in *Computer-Aided Design of User Interfaces I, Proceedings of the Second International Workshop on Computer-Aided Design of*

User Interfaces, June 5-7, 1996, Namur, Belgium, J. Vanderdonckt, Ed. Presses Universitaires de Namur, 1996, pp. 19–36.

[Radeke and Forbrig07] F. Radeke and P. Forbrig, "Patterns in Task-Based Modeling of User Interfaces," in *Task Models and Diagrams for User Interface Design, 6th International Workshop, TAMODIA 2007, Toulouse, France, November 7-9, 2007, Proceedings*, ser. Lecture Notes in Computer Science, M. Winckler, H. Johnson, and P. A. Palanque, Eds., vol. 4849. Springer, 2007, pp. 184–197.

[Radeke et al.06] F. Radeke, P. Forbrig, A. Seffah, and D. Sinnig, "PIM Tool: Support for Pattern-Driven and Model-Based UI Development," in *Task Models and Diagrams for Users Interface Design, 5th International Workshop, TAMODIA 2006, Hasselt, Belgium, October 23-24, 2006. Revised Papers*, ser. Lecture Notes in Computer Science, K. Coninx, K. Luyten, and K. A. Schneider, Eds., vol. 4385. Springer, 2006, pp. 82–96.

[Rahardja95] A. Rahardja, "Multimedia Systems Design: A Software Engineering Perspective," in *International Conference on Computers and Education (ICCE) 95 Proceedings*. IEEE Computer Society, 1995.

[Rea] "Propellerhead Reason," [Website]. URL: http://www.propellerheads.se/products/reason/

[Reggio et al.01] G. Reggio, M. Cerioli, and E. Astesiano, "Towards a Rigorous Semantics of UML Supporting Its Multiview Approach," in *Fundamental Approaches to Software Engineering*, vol. Volume 2029/2001. Springer Berlin / Heidelberg, 2001, pp. 171–186. URL: http://www.springerlink.com/content/y4abenbqhc9pq9q4/

[Reisman98] S. Reisman, "Multimedia Is Dead," *IEEE MultiMedia*, vol. 5, no. 1, pp. 4–5, 1998.

[Rev] "UsiXML – ReversiXML," [Website]. URL: http://www.usixml.org/index.php?mod=pages&id=32

[Rosson and Carroll02] M. B. Rosson and J. M. Carroll, *Usability Engineering: Scenario-Based Development of Human Computer Interaction (Interactive Technologies)*, 1st ed. San Francisco, USA: Morgan Kaufmann, 10 2002. URL: http://amazon.com/o/ASIN/1558607129/

[Rosson and Gilmore07] M. B. Rosson and D. J. Gilmore, Eds., *Proceedings of the 2007 Conference on Human Factors in Computing Systems, CHI 2007, San Jose, California, USA, April 28 - May 3, 2007*. ACM, 2007.

[Rothenberg89] J. Rothenberg, "The nature of modeling," in *Artificial intelligence, simulation & modeling*, L. E. Widman, K. A. Loparo, and N. R. Nielson, Eds. New York, NY, USA: John Wiley & Sons, Inc., 1989, pp. 75–92.

[Rout and Sherwood99] T. Rout and C. Sherwood, "Software engineering standards and the development of multimedia-based systems," in *Software Engineering Standards, 1999. Proceedings. Fourth IEEE International Symposium and Forum on*, 17-21 May 1999, pp. 192–198.

[Rozenberg97] G. Rozenberg, Ed., *Handbook of graph grammars and computing by graph transformation: volume I. foundations*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1997.

[Rumbaugh et al.91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-oriented modeling and design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1991.

[Rumpe04] B. Rumpe, *Modellierung mit UML : Sprache, Konzepte und Methodik (Xpert.press)*. Berlin Heidelberg: Springer, June 2004.

[Rumpe06] B. Rumpe, *Agile Modellierung mit UML: Codegenerierung, Testfälle, Refactoring (Xpert.press)*. Berlin Heidelberg: Springer-Verlag, 2006.

[Rupp et al.07] C. Rupp, S. Queins, and B. Zengler, *UML 2 glasklar. Praxiswissen für die UML-Modellierung*, 3rd ed. München Wien: Carl Hanser Verlag, 2007.

[Salembier and Sikora02] P. Salembier and T. Sikora, *Introduction to MPEG-7: Multimedia Content Description Interface*, B. Manjunath, Ed. New York, NY, USA: John Wiley & Sons, Inc., 2002.

[Sampaio et al.97] P. N. M. Sampaio, C. Y. Shiga, and W. L. de Souza, "Modelling Multimedia and Hypermedia Applications using an E-LOTOS/MHEG-5 Approach," in *Workshop on Conceptual Modelling in Multimedia Information Seeking*, 1997. URL: http://osm7.cs.byu.edu/ER97/workshop1/

[Satoh et al.08] S. Satoh, F. Nack, and M. Etoh, Eds., *Advances in Multimedia Modeling, 14th International Multimedia Modeling Conference, MMM 2008, Kyoto, Japan, January 9-11, 2008, Proceedings*, ser. Lecture Notes in Computer Science, vol. 4903. Springer, 2008.

[Sauer and Engels99a] S. Sauer and G. Engels, "Extending UML for Modeling of Multimedia Applications," in *VL '99: Proceedings of the IEEE Symposium on Visual Languages*, M. Hirakawa and P. Mussio, Eds. Washington, DC, USA: IEEE Computer Society, 1999, p. 80.

[Sauer and Engels99b] S. Sauer and G. Engels, "UML-basierte Modellierung von Multimediaanwendungen," in *Modellierung 1999, Workshop der Gesellschaft für Informatik e. V. (GI), März 1999 in Karlsruhe*, J. Desel, K. Pohl, and A. Schürr, Eds. Teubner, 1999, pp. 155–170.

[Sauer and Engels01] S. Sauer and G. Engels, "UML-based Behavior Specification of Interactive Multimedia Applications," in *2002 IEEE CS International Symposium on Human-Centric Computing Languages and Environments (HCC 2001), September 5-7, 2001 Stresa, Italy*. IEEE Computer Society, 2001, pp. 248–255.

[Schlungbaum96] E. Schlungbaum, "Model-based User Interface Software Tools Current state of declarative models," Georgia Institute of Technology, Tech. Rep. GIT-GVU-96-30, November 1996.

[Schmidt et al.99] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, and W. V. de Velde, “Advanced Interaction in Context,” in *HUC ’99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. London, UK: Springer-Verlag, 1999, pp. 89–101.

[Schulert et al.85] A. J. Schulert, G. T. Rogers, and J. A. Hamilton, “ADM – a dialog manager,” in *CHI ’85: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 1985, pp. 177–183.

[Schulmeister03] R. Schulmeister, “Taxonomy of Multimedia Component Interactivity,” *Studies in Communication Sciences.*, vol. 3, no. 3, pp. 61–80, March 2003.

[Schwabe et al.02] D. Schwabe, a. Robson Mattos Guimar and G. Rossi, “Cohesive Design of Personalized Web Applications,” *IEEE Internet Computing*, vol. 6, no. 2, pp. 34–43, 2002.

[Schweizer08] E. Schweizer, “Integration modell-getriebener Software-Entwicklung mit zusätzlichen heterogenen Werkzeugen durch Transformationen,” Project Thesis, University of Munich, Munich, March 2008.

[Schwinger and Koch03] W. Schwinger and N. Koch, “Modellierung von Web-Anwendungen,” in *Web Engineering: Systematische Entwicklung von Web-Anwendungen*. Heidelberg: Dpunkt Verlag, 2003, pp. 49–76.

[Seffah and Metzker04] A. Seffah and E. Metzker, “The obstacles and myths of usability and software engineering,” *Commun. ACM*, vol. 47, no. 12, pp. 71–76, 2004.

[Seffah et al.05] A. Seffah, J. Gulliksen, and M. C. Desmarais, *Human-Centered Software Engineering, Integrating Usability in the Software Development Lifecycle*, 1st ed. Springer Netherlands, 12 2005. URL: http://amazon.de/o/ASIN/140204027X/

[Seidewitz03] E. Seidewitz, “What Models Mean,” *IEEE Softw.*, vol. 20, no. 5, pp. 26–32, 2003.

[Sendall and Küster04] S. Sendall and J. M. Küster, “Taming Model Round-Trip Engineering,” in *Proceedings of Workshop on Best Practices for Model-Driven Software Development on OOPSLA 2004, Vancouver, Canada*, 2004. URL: http://www.zurich.ibm.com/pdf/csc/position-paper-mdsd04_sendall.pdf

[Shaw01] M. Shaw, “The coming-of-age of software architecture research,” in *ICSE ’01: Proceedings of the 23rd International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2001, p. 656.

[Shaykhit06] D. Shaykhit, “Definition and implementation of an UML Profile and a transformation for the graphical specification of MML Models in UML-Tool MagicDraw,” Project Thesis, University of Munich, Munich, December 2006.

[Shaykhit07] D. Shaykhit, “A Flexible Code Generator for the Model-Driven Development of Multimedia Applications,” Diploma Thesis, University of Munich, Munich, October 2007.

[Shehory and Sturm01] O. Shehory and A. Sturm, “Evaluation of modeling techniques for agent-based systems,” in *AGENTS ’01: Proceedings of the fifth international conference on Autonomous agents*. New York, NY, USA: ACM, 2001, pp. 624–631.

[Sherrod06] A. Sherrod, *Ultimate Game Programming With DirectX*, 1st ed. Boston, MA, USA: Charles River Media, 5 2006. URL: http://amazon.com/o/ASIN/1584504587/

[Shneiderman and Plaisant04] B. Shneiderman and C. Plaisant, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 4th ed. Addison Wesley, 2004.

[Sikorski05] M. Sikorski, Ed., *Task Models and Diagrams for User Interface Design: Proceedings of the Forth International Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2005, Gdansk, Poland, September 26-27, 2005*. ACM, 2005.

[Sil] “Microsoft Silverlight,” [Website]. URL: http://silverlight.net/

[Sinnig et al.07] D. Sinnig, P. Chalin, and F. Khendek, “Common Semantics for Use Cases and Task Models,” in *Integrated Formal Methods, 6th International Conference, IFM 2007, Oxford, UK, July 2-5, 2007, Proceedings*, ser. Lecture Notes in Computer Science, J. Davies and J. Gibbons, Eds., vol. 4591. Springer, 2007, pp. 579–598.

[Ske] “UsiXML – SketchiXML,” [Website]. URL: http://www.usixml.org/index.php?mod=pages&id=14

[Softwarea] E. Software, “Flash Decompiler Trillix,” [Website]. URL: http://www.flash-decompiler.com/

[Softwareb] S. Software, “Sothink SWF Decompiler,” [Website]. URL: http://www.sothink.com/

[Sommerville06] I. Sommerville, *Software Engineering: (Update) (8th Edition) (International Computer Science Series)*, 8th ed. Addison Wesley, 6 2006. URL: http://amazon.com/o/ASIN/0321313798/

[Sottet et al.06] J.-S. Sottet, G. Calvary, and J.-M. Favre, “Mapping Model: A First Step to Ensure Usability for sustaining User Interface Plasticity,” in *MDDAUI’06*, ser. CEUR Workshop Proceedings, A. Pleuß, J. V. den Bergh, H. Hußmann, S. Sauer, and A. Bödcher, Eds., vol. 214. CEUR-WS.org, 2006.

[Sottet et al.07a] J.-S. Sottet, G. Calvary, J. Coutaz, and J.-M. Favre, “A Model-Driven Engineering Approach for the Usability of User Interfaces,” in *Proc. Engineering Interactive Systems 2007*. Springer (to appear), 2007. URL: http://iihm.imag.fr/publs/2007/EIS07-sottet.pdf

[Sottet et al.07b] J.-S. Sottet, V. Ganneau, G. Calvary, J. Coutaz, A. Demeure, J.-M. Favre, and R. Demumieux, “Model-Driven Adaptation for Plastic User Interfaces,” in *Human-Computer Interaction - INTERACT 2007, 11th IFIP TC 13 International Conference, Rio de Janeiro, Brazil, September 10-14, 2007, Proceedings, Part I*, ser. Lecture Notes in Computer Science, M. C. C. Baranauskas, P. A. Palanque, J. Abascal, and S. D. J. Barbosa, Eds., vol. 4662. Springer, 2007, pp. 397–410.

[Specht and Zoller00] G. Specht and P. Zoller, “HMT: Modeling Temporal Aspects in Hypermedia Applications,” in *Web-Age Information Management, First International Conference, WAIM 2000, Shanghai, China, June 21-23, 2000, Proceedings*, ser. Lecture Notes in Computer Science, H. Lu and A. Zhou, Eds., vol. 1846. Springer, 2000, pp. 259–270.

[Staas04] D. Staas, "Standardanwendungen," in *Taschenbuch Informatik*, 5th ed., U. Schneider and D. Werner, Eds. Munich, Germany: Carl Hanser Verlag, 2004, ch. 23.

[Stahl et al.07] T. Stahl, M. Völter, and S. Efftinge, *Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management*, 2nd ed. Dpunkt Verlag, 5 2007. URL: http://amazon.de/o/ASIN/3898644480/

[Stankowski and Duschek94] A. Stankowski and K. Duschek, Eds., *Visuelle Kommunikation: Ein Design-Handbuch*, 2nd ed. Berlin: Dietrich Reimer Verlag, 12 1994. URL: http://amazon.de/o/ASIN/3496011068/

[Steinberg et al.08] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework (2nd Edition) (The Eclipse Series)*, 2nd ed. Addison-Wesley Professional, 4 2008. URL: http://amazon.com/o/ASIN/0321331885/

[Steinmetz and Nahrstedt04] R. Steinmetz and K. Nahrstedt, *Multimedia Applications*, 1st ed. Berlin: Springer, 1 2004. URL: http://amazon.de/o/ASIN/3540408495/

[Steinmetz00] R. Steinmetz, *Multimedia-Technologie. Grundlagen, Komponenten und Systeme.*, 3rd ed. Berlin: Springer, 7 2000. URL: http://amazon.de/o/ASIN/3540673326/

[Stephens and Rosenberg03] M. Stephens and D. Rosenberg, *Extreme Programming Refactored: The Case Against XP*. Apress, 2003.

[Störrle04] H. Störrle, "Semantics of Structured Nodes in UML 2.0 Activities," in *Nordic Workshop on UML (NWUML) 2004*, 2004. URL: http://crest.abo.fi/nwuml04/

[Syn] "SyncATL," [Website]. URL: http://www.ipl.t.u-tokyo.ac.jp/~xiong/modelSynchronization.html

[Szekely et al.92] P. Szekely, P. Luo, and R. Neches, "Facilitating the exploration of interface design alternatives: the HUMANOID model of interface design," in *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 1992, pp. 507–515.

[Szekely et al.95] P. A. Szekely, P. N. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher, "Declarative interface models for user interface construction tools: the MASTER-MIND approach," in *Engineering for Human-Computer Interaction, Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, Yellowstone Park, USA, August 1995*, ser. IFIP Conference Proceedings, L. J. Bass and C. Unger, Eds., vol. 45. Chapman & Hall, 1995, pp. 120–150.

[Szekely96] P. A. Szekely, "Retrospective and Challenges for Model-Based Interface Development," in *Design, Specification and Verification of Interactive Systems'96, Proceedings of the Third International Eurographics Workshop, June 5-7, 1996, Namur, Belgium*, F. Bodart and J. Vanderdonckt, Eds. Springer, 1996, pp. 1–27.

[Taentzer00] G. Taentzer, "AGG: A Tool Environment for Algebraic Graph Transformation," in *AGTIVE '99: Proceedings of the International Workshop on Applications of Graph Transformations with Industrial Relevance*. London, UK: Springer-Verlag, 2000, pp. 481–488.

[TAM] "TAMODIA 2007 – 6th International workshop on TAsk MOdels and DIAgrams," [Website]. URL: http://liihs.irit.fr/tamodia2007/

[Tannenbaum98] R. S. Tannenbaum, *Theoretical Foundations of Multimedia*. New York: W. H. Freeman, 1998.

[Tap] "Taparo Games," january 23, 2008. URL: http://www.taparo.com/

[Tavares] A. Tavares, "An Interactive Home Media Center," [Website]. URL: http://www.adrianatavares.com/iyro/

[TERa] "TERESA Homepage," february 27, 2008. URL: http://giove.cnuce.cnr.it/teresa.html

[TERb] "XML languages of TERESA," juli 31, 2008. URL: http://giove.isti.cnr.it/teresa/teresa_xml.html

[Tidwell05] J. Tidwell, *Designing Interfaces: Patterns for Effective Interaction Design*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005.

[Tonella and Potrich04] P. Tonella and A. Potrich, *Reverse Engineering of Object Oriented Code*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2004.

[Too] "Sum Total Systems – Toolbook," [Website]. URL: SumTotalSystems

[Top] "Topcased," [Website]. URL: http://topcased.gforge.enseeiht.fr/

[Tra] "UsiXML – TransformiXMLWWW," [Website]. URL: http://www.usixml.org/index.php?mod=pages&id=34

[Trætteberg02] H. Trætteberg, "Model-based User Interface Design," Ph.D. dissertation, Norwegian University of Science and Technology, Oslo, 2002. URL: http://www.idi.ntnu.no/~hal/_media/research/thesis.pdf?id=research%3Athesis&cache=cache

[Tran-Thuong and Roisin03] T. Tran-Thuong and C. Roisin, "Multimedia modeling using MPEG-7 for authoring multimedia integration," in *MIR '03: Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*. New York, NY, USA: ACM, 2003, pp. 171–178.

[Troyer and Decruyenaere00] O. D. Troyer and T. Decruyenaere, "Conceptual modelling of web sites for end-users," *World Wide Web Journal*, vol. 3, no. 1, pp. 27–42, 2000.

[Tufte01] E. R. Tufte, *The Visual Display of Quantitative Information*, 2nd ed. Chesire, CT, USA: Graphics Press, 2001.

[UIM] "UIML.org," march 05, 2008. URL: http://UIML.org

[Unta] "Department für Pädagogik und Rehabilitation – Unterrichtsmitschau und didaktische Forschung," [Website]. URL: http://mitschau.edu.lmu.de/index.html

[Untb] "Department für Pädagogik und Rehabilitation – Unterrichtsmitschau und didaktische Forschung – Shop," [Website]. URL: http://mitschau.edu.lmu.de/av_medien/index.php

[Urbieta et al.07] M. Urbieta, M. Urbieta, G. Rossi, J. Ginzburg, and D. Schwabe, "Designing the Interface of Rich Internet Applications," in *Proc. Latin American Web Congress LA-WEB 2007*, G. Rossi, Ed., 2007, pp. 144–153.

[Usia] "UsiXML – Code Generators," [Website]. URL: http://www.usixml.org/index.php?mod=pages&id=21

[Usib] "UsiXML - Home of the USer Interface eXtensible Markup Language," february 27, 2008. URL: http://www.usixml.org/

[Usi06] "UsiXML v1.6.4 Metamodel as Rational Rose file," 2006. URL: http://www.usixml.org/index.php?mod=download&file=usixml-doc/UsiXML-v1.6.4.mdl

[Usi07] *UsiXML V1.8 Reference Manual*, February 2007. URL: http://www.usixml.org/index.php?mod=download&file=usixml-doc/UsiXML_v1.8.0-Documentation.pdf

[Van den Bergh and Coninx05] J. Van den Bergh and K. Coninx, "Towards modeling context-sensitive interactive applications: the context-sensitive user interface profile (CUP)," in *Proceedings of the ACM 2005 Symposium on Software Visualization, St. Louis, Missouri, USA, May 14-15, 2005*, T. L. Naps and W. D. Pauw, Eds. ACM, 2005, pp. 87–94.

[Van den Bergh and Coninx06] J. Van den Bergh and K. Coninx, "CUP 2.0: High-Level Modeling of Context-Sensitive Interactive Applications," in *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006, Proceedings*, ser. Lecture Notes in Computer Science, O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, Eds., vol. 4199. Springer, 2006, pp. 140–154.

[Van den Bergh06] J. Van den Bergh, "High-Level User Interface Models for Model-Driven Design of Context-Sensitive User Interfaces," PhD, Hasselt University, Diepenbeek, Belgium, 2006. URL: http://research.edm.uhasselt.be/~jvandenbergh/phd/phd-JanVandenBergh.pdf

[van Welie] M. van Welie, "A Pattern Library for Interaction Design," [Website]. URL: http://www.welie.com/

[Vanderdonckt and Bodart93] J. M. Vanderdonckt and F. Bodart, "Encapsulating knowledge for intelligent automatic interaction objects selection," in *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*. New York, NY, USA: ACM, 1993, pp. 424–429.

[Vanderdonckt et al.04] J. Vanderdonckt, N. J. Nunes, and C. Rich, Eds., *Proceedings of the 2004 International Conference on Intelligent User Interfaces, January 13-16, 2004, Funchal, Madeira, Portugal*. ACM, 2004.

[Vanderdonckt96] J. Vanderdonckt, Ed., *Computer-Aided Design of User Interfaces I, Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces, June 5-7, 1996, Namur, Belgium*. Presses Universitaires de Namur, 1996.

[Vanderdonckt05] J. Vanderdonckt, “A MDA-Compliant Environment for Developing User Interfaces of Information Systems,” in *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings*, ser. Lecture Notes in Computer Science, O. Pastor and J. Falcão e Cunha, Eds., vol. 3520. Springer, 2005, pp. 16–31.

[Vangheluwe et al.03] H. Vangheluwe, H. Vangheluwe, and J. de Lara, “Computer automated multi-paradigm modelling: meta-modelling and graph transformation,” in *Proc. Winter Simulation Conference*, J. de Lara, Ed., vol. 1, 2003, pp. 595–603 Vol.1.

[Vazirgiannis and Boll97] M. Vazirgiannis and S. Boll, “Events in interactive multimedia applications: modeling and implementation design,” in *Proc. IEEE International Conference on Multimedia Computing and Systems ’97*. IEEE Computer Society, 1997, pp. 244–251.

[VIA] “VIATRA2 Component – Eclipse Generative Modeling Technologies (GMT) Project,” [Website]. URL: http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/VIATRA2/index.html

[Villard et al.00] L. Villard, C. Roisin, and N. Layaïda, “An XML-Based Multimedia Document Processing Model for Content Adaptation,” in *Digital Documents: Systems and Principles, 8th International Conference on Digital Documents and Electronic Publishing, DDEP 2000, 5th International Workshop on the Principles of Digital Document Processing, PODDP 2000, Munich, Germany, September 13-15, 2000, Revised Papers*, ser. Lecture Notes in Computer Science, P. R. King and E. V. Munson, Eds., vol. 2023. Springer, 2000, pp. 104–119.

[Vis] “Microsoft MSDN – Visual Studio.” URL: http://msdn.microsoft.com/en-us/vstudio/default.aspx

[Vitzthum and Hussmann06] A. Vitzthum and H. Hussmann, “Modeling Augmented Reality User Interfaces with SSIML/AR,” *Journal of Multimedia (JMM)*, vol. 1, no. 3, pp. 13–22, June 2006.

[Vitzthum and Pleuß05] A. Vitzthum and A. Pleuß, “SSIML: designing structure and application integration of 3D scenes,” in *Proceeding of the Tenth International Conference on 3D Web Technology, Web3D 2005, Bangor, UK, March 29 - April 1, 2005*, N. W. John, S. Ressler, L. Chittaro, and D. A. Duce, Eds. ACM, 2005, pp. 9–17.

[Vitzthum05] A. Vitzthum, “SSIML/Behaviour: Designing Behaviour and Animation of Graphical Objects in Virtual Reality and Multimedia Applications,” in *ISM ’05: Proceedings of the Seventh IEEE International Symposium on Multimedia*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 159–167.

[Vitzthum06] A. Vitzthum, “SSIML/Components: a visual language for the abstract specification of 3D components,” in *Web3D ’06: Proceedings of the eleventh international conference on 3D web technology*. New York, NY, USA: ACM, 2006, pp. 143–151.

[Vitzthum08] A. Vitzthum, “Entwicklungsunterstützung für interaktive 3D-Anwendungen,” Ph.D. dissertation, University of Munich, Munic, Germany, 2008. URL: http://edoc.ub.uni-muenchen.de/9459/

[Völter and Kolb05] M. Völter and B. Kolb, "openArchitectureWare und Eclipse," *Eclipse Magazin*, vol. 3, Mai 2005.

[Voss et al.99] J. Voss, P. Pauen, H.-W. Six, M. Nagl, A. Behle, B. Westfechtel, H. Balzert, C. Weidauer, W. Schäfer, J. Wadsack, and U. Kelter, Eds., *Studie über Softwaretechnische Anforderungen an multimediale Lehr- und Lernsysteme*. Forschergruppe SofTec NRW, September 1999.

[Vredenburg et al.02] K. Vredenburg, J.-Y. Mao, P. W. Smith, and T. Carey, "A survey of user-centered design practice," in *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2002, pp. 471–478.

[W3C03] *Scalable Vector Graphics (SVG) 1.1 Specification*, W3C, January 2003. URL: http://www.w3.org/TR/SVG11/

[Wahl and Rothermel94] T. Wahl and K. Rothermel, "Representing time in multimedia systems," in *Proc. International Conference on Multimedia Computing and Systems*, 1994, pp. 538–543.

[Walker et al.03] R. J. Walker, L. C. Briand, D. Notkin, C. B. Seaman, and W. F. Tichy, "Panel: empirical validation: what, why, when, and how," in *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 721–722.

[Ware04] C. Ware, *Information Visualization*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, 2004.

[Warmer and Kleppe03] J. Warmer and A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA*, 2nd ed. Addison-Wesley Professional, 9 2003. URL: http://amazon.com/o/ASIN/0321179366/

[Wasserman85] A. I. Wasserman, "Extending State Transition Diagrams for the Specification of Human-Computer Interaction," *IEEE Trans. Softw. Eng.*, vol. 11, no. 8, pp. 699–713, 1985.

[Weba] "The Web Engineering Community Site," [Website]. URL: http://webengineering.org

[Webb] "WebRatio," [Website]. URL: http://www.webratio.com/Home.do?link=oln489d.redirect

[Weiser99] M. Weiser, "The computer for the 21st century," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, 1999.

[Wenz et al.07] C. Wenz, T. Hauser, and A. Kappler, *ActionScript 3 – Das Praxisbuch*, 1st ed. Bonn, Germany: GALILEO PRESS, 12 2007. URL: http://amazon.de/o/ASIN/3836210525/

[Wernesgrüner] Wernesgrüner, "Go For Green – Gewinnspiel," [Website]. URL: http://www.wernesgruener.de/www/wg/bereich/gewinnspiel/

[Wie] "München TV – Wiesn Minigolf," [Website]. URL: http://www.muenchen-tv.de/archiv/Wiesn_Minigolf-1031.html

[Williamson et al.07] C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, Eds., *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*. ACM, 2007.

[Winckler et al.07] M. Winckler, H. Johnson, and P. A. Palanque, Eds., *Task Models and Diagrams for User Interface Design, 6th International Workshop, TAMODIA 2007, Toulouse, France, November 7-9, 2007, Proceedings*, ser. Lecture Notes in Computer Science, vol. 4849. Springer, 2007.

[Wirsing90] M. Wirsing, "Algebraic Specification," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier and MIT Press, 1990, vol. B: Formal Models and Sematics, pp. 675–788.

[Wisneski et al.98] C. Wisneski, H. Ishii, A. Dahley, M. G. Gorbet, S. Brave, B. Ullmer, and P. Yarin, "Ambient Displays: Turning Architectural Space into an Interface between People and Digital Information," in *CoBuild '98: Proceedings of the First International Workshop on Cooperative Buildings, Integrating Information, Organization, and Architecture*. London, UK: Springer-Verlag, 1998, pp. 22–32.

[Wolff et al.05] A. Wolff, P. Forbrig, A. Dittmar, and D. Reichart, "Linking GUI elements to tasks: supporting an evolutionary design process," in *Task Models and Diagrams for User Interface Design: Proceedings of the Forth International Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2005, Gdansk, Poland, September 26-27, 2005*, M. Sikorski, Ed. ACM, 2005, pp. 27–34.

[Wolff05] C. Wolff, "Media Design Patterns," in *Designing Information Systems*, ser. Schriften zur Informationswissenschaft, M. Eibl, C. Wolff, and C. Womser-Hacker, Eds. Konstanz: UVK, 2005, vol. 43, pp. 209–217.

[Wu06a] W.-W. Wu, "Abbildung von MML-Modellen in Code für das Java-Framework Piccolo," Project Thesis, University of Munich, Munich, February 2006.

[Wu06b] W.-W. Wu, "Analyse und Vergleich ausgewählter Ansätze zur modellgetriebenen Entwicklung von Benutzerschnittstellen," Diploma Thesis, University of Munich, Munich, November 2006.

[WWWa] "World Wide Web Consortium," [Website]. URL: http://www.w3.org/

[WWWb] "WWW2007 – 16th International World Wide Web Conference," [Website]. URL: http://www2007.org/

[XAM] "Microsoft XAML," [Website]. URL: http://msdn.microsoft.com/en-us/library/ms752059.aspx

[Xiong et al.07] Y. Xiong, D. Liu, Z. Hu, H. Zhao, M. Takeichi, and H. Mei, "Towards automatic model synchronization from model transformations," in *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. New York, NY, USA: ACM, 2007, pp. 164–173.

[XUL] "XULPlanet," [Website]. URL: http://www.xulplanet.com/

[Yard and Peters04] T. Yard and K. Peters, *Extending Macromedia Flash MX 2004: Complete Guide and Reference to JavaScript Flash*, 1st ed. Berkeley, CA, USA: friends of ED, 1 2004. URL: http://amazon.com/o/ASIN/1590593049/

[Zendler98] A. Zendler, *Multimedia Development Systems (with methods for modeling multimedia applications)*, R. Haggenmüller and H. Schwärtzel, Eds. Marburg, Germany: Tectum Verlag, 1998. URL: http://amazon.de/o/ASIN/3896089285/

[Zhao and Zou07] X. Zhao and Y. Zou, "A Framework for Incorporating Usability into Model Transformations," in *MDDAUI'07*, ser. CEUR Workshop Proceedings, A. Pleuß, J. V. den Bergh, H. Hußmann, S. Sauer, and D. Görlich, Eds., vol. 297. CEUR-WS.org, 2007.

[Ziegler08] S. Ziegler, "Steigerung der Effizienz bei der UI-Gestaltung im Kontext der interdisziplinären Zusammenarbeit unter Einsatz von WPF," Diploma Thesis, University of Munich, Munich, January 2008.

[Zoller01] P. Zoller, "HMT: Modeling Interactive and Adaptive Hypermedia Applications," in *Information Modeling in the New Millennium*, M. Rossi and K. Siau, Eds. Hershey, PA, USA: IGI Publishing, 2001, pp. 383–405.

[Zschaler07] S. Zschaler, "A Semantic Framework for Non-functional Specifications of Component-Based Systems," Dissertation, Technische Universität Dresden, Dresden, Germany, Apr. 2007.

Acknowledgements

First of all, I would like to thank Prof. Hußmann for his continuous support, his good advice, and the constructive and positive working atmosphere he created as head the research group.

I would like to thank Prof. Forbrig and Prof. Vanderdonckt for their willingness to be reviewers for this thesis and for their contributions and remarks.

I would like to thank all my colleagues for the fruitful and pleasant collaboration in our research group. In particular, I would like to thank my former colleague Arnd Vitzthum for the fruitful discussions on modeling and multimedia.

Finally, I would like to thank my family for their sincere support throughout all times and, in particular, Astrid Weiland whose support is the most important to me.