

Extending X3D for Distributed Multimedia Processing and Control

Michael Repplinger*
DFKI Saarbrücken

Alexander Löffler†
Saarland University

Benjamin Schug‡
Saarland University

Philipp Slusallek§
DFKI Saarbrücken
Saarland University



Figure 1: Specifying a distributed multimedia flow graph directly in X3D allows access to local and remote components for flexible media processing and usage of live streams instead of static files. The picture shows a screen shot of our multimedia-enhanced X3D render system. It shows a scene specified entirely in X3D, presenting the same live TV stream three times as a texture while applying different postprocessing steps after decoding the video data: From left to right, we see the original TV stream, the same stream with adapted brightness, and the same stream after a compositing with two static logos.

Abstract

Web-based applications of interactive 3D computer graphics are showing a tendency to get more interconnected and visually complex. Virtual communities like Second Life demand realism not only in terms of realistic rendering, but also in terms of integrated multimedia content. For these Web-based applications, X3D is the ISO-standard way to specify and manipulate scene descriptions. In terms of multimedia integration, however, X3D offers to specify content only in the form of URLs pointing to files. Modern middleware for distributed multimedia, on the other hand, allows applications to harness the full range of multimedia processing as well as transparent use and full control of both local and remote components. Integrating a full multimedia processing pipeline into X3D would enable Web authors to use, for example, streaming media, post-processing on media streams, or routing between scene elements (e.g., sensors) and elements of multimedia processing (e.g., TV cards). A full integration of multimedia in X3D is yet missing.

In this paper, we propose X3D extensions for a seamless mapping of a distributed multimedia flow graph onto an X3D scene graph, making all the features of a distributed multimedia middleware accessible and usable within an X3D scene. We present our proposed specification and implementation of multimedia nodes for X3D. Using examples and implemented X3D application scenarios, we show the simplicity and feasibility of our approach.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality; I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics; I.3.8 [Computer Graphics]: Applications—;

Keywords: X3D, distributed and parallel processing, distributed flow graph, distributed processing, multimedia middleware

*e-mail:michael.repplinger@dfki.de

†e-mail:loeffler@cs.uni-sb.de

‡e-mail:bschug@graphics.cs.uni-sb.de

§e-mail:philipp.slusallek@dfki.de

1 Introduction

In the field of web-based computer graphics, X3D [Brutzman and Daly 2007] is the established standard as file format and run-time architecture for the definition of interactive applications. X3D offers functionality for definition, viewing and navigation of 3D scenes, supports hierarchical geometry, animations, lighting, scripting, and much more. In terms of integration of *multimedia content* into a 3D scene, the X3D semantics support audio, which is specified by location and spatial distribution characteristics, and video, which is specified as a texture for existing scene geometry. The actual media container – typically a file – is specified in the form of a URL string.

Current and upcoming 3D applications for the Internet include *virtual worlds* like e.g. Second Life [Rymaszewski 2008], where participating applications are not separate entities anymore, but altogether form a larger collective. For the ever-increasing realism of those worlds, multimedia plays an important role to suit the needs of their users. In order to realize virtual world scenarios based on the X3D standard, its capabilities to define and manipulate multimedia processing are insufficient and need to be extended. We therefore define five requirements an integration of multimedia processing in X3D should fulfill:

(1) Explicit Specification of Media Processing From a high-level point of view, the drawbacks of current multimedia integration in X3D revolve around the inability to specify the actual media processing in more detail. Actions like post-processing of a video or audio stream, e.g. adjusting the brightness of a video, or routing the components of a multimedia stream to dedicated places

of output in the scene, are currently impossible to realize completely within an X3D application. Applications involving multiple live and synchronized audio and video streams in a 3D world, for instance a virtual conference room, should be possible to define inside a single X3D file.

(2) Streaming Media URL strings used to specify media items in X3D always describe *files*, even though the Internet is predestined for *streaming media*. The standard assumes that browsers completely download audio or video files specified by URLs before they start playing them. That is, they assume a *local* processing, even though URLs may very well specify a *remote* location. A scenario like watching a live real-world TV stream inside a virtual-world living room is thus rendered impossible, because the media download will not ever be able to finish. Moreover, in [Brutzman and Kolsch 2007], streaming support for media support is listed as an essential feature for future versions of the X3D standard.

(3) Control of Components Continuing the mentioned “virtual living room” scenario, the user of the virtual TV set wants to be able to switch channels as well. Back in the real world, this entails switching the channel on e.g. a TV card inside the displaying PC. X3D does not yet include a technique to route a click inside the virtual world to an outside event on an actual device. Even though X3D should indeed abstract actual hardware implementations to keep scenes portable, providing an interface to multimedia devices would be a useful extension. This would not only allow to switch channels on a TV card, but also, for instance, to control a video camera whose feed is again shown within the virtual world. Admittedly, some control events could in theory be encoded within a URL string, but this would not represent a natural way of specifying functionality and would quickly lead to entirely unreadable and unmaintainable URLs.

(4) Control of Media Transmission Multimedia devices should be controllable regardless of where their physical location within the network actually is. Such a feature also entails a control of the actual network connections inbetween several processing hosts to enable, for instance, streaming via transmission protocols that are more suitable for the transmission of live content. Having control of the network also enables to react to changes in connection quality and e.g. adapt the bitrate of a video encoding in case of losing too many packets during data transmission.

(5) Abstract Specification of Media Processing The last requirement is actually a result of the flexibility introduced by the former four. The theoretical ability to specify an entire multimedia processing pipeline within X3D should never entail that a user is *forced* to specify all the details every time she wants to include anything related to multimedia in her scene. Instead, we envision a principle of *scalable transparency* when specifying multimedia content: While it should be possible to dig deep into lower layers of processing, a basic multimedia usage should be intuitive and should involve only very few X3D nodes in the scene specification.

As a result of the five presented requirements, this paper will introduce extensions to the X3D specification, which enable on the one hand full definition and control of a distributed multimedia flow inside an X3D application, and on the other hand the possibility to specify only very few necessary components. This paper is structured as follows: First, Section 2 highlights related work in the field. Afterwards, Section 3 presents how state-of-the-art multimedia software solves the integration requirements internally, before Section 4 shows how we integrate those multimedia concepts in X3D. Section 5 will then show how to use those X3D extensions in

practice, before Section 6 will give details on how the multimedia integration is implemented in our reference system. Finally, Section 7 will conclude the paper and illuminate further work.

2 Related Work

In the context of web technologies the term *multimedia* refers to a huge variety of different technologies. One of the most widespread solutions for presenting web-based multimedia is Adobe’s Flash technology [Adobe 2009]. Flash enables developers to combine audio and video data together with 2D animation using both raster and vector graphics. In addition to this, Flash provides a built-in scripting language to realize complex interaction with elements of a Flash file. Proprietary 3D extensions for Flash do exist, which enable 3D graphics inside a Flash movie. However, Flash is not used in combination with X3D: both a direct integration of interactive Flash movies into X3D, or an integration of an X3D scene graph into Flash movies do not yet exist.

In [ISO/IEC 14496-11 2007], an extension to the MPEG-4 standard is defined, which includes a scene description format based on VRML to specify animation and vector graphics within an MPEG-4 file. In addition, this extension even allows streaming of the actual scene description. Similar to Flash, there exists no direct work to combine elements of the MPEG-4 scene description with elements of an X3D scene graph. Furthermore, to our best knowledge, there exists no implementation of the MPEG-4 scene description that covers all aspects of the defined standard. Neither Flash nor MPEG-4 consider control of hardware components – see requirements (2) and (3) – or the delivery of media data via other transport protocols than HTTP (see requirement (4)).

The DeckLight processing engine [Müller et al. 2006b] is a part of the Soundium multimedia authoring and presentation framework [Müller et al. 2006a] and provides a real-time multimedia processing engine including audio control, MIDI control and OpenGL rendering. The utilized high-level flow graph structure of DeckLight can contain 3D scenes as sub-graphs. However, DeckLight does not provide transparent access to remote resources and a fine-grained specification of multimedia processing.

The Avalon Mixed-Reality Framework [Behr and Dähne 2003] provides numerous extensions to the X3D standard with a special focus on virtual and augmented reality (VR/AR) applications. For example, it integrates control devices for VR environments, multi-touch user interfaces, or physics simulations. Within the project, Avalon uses the concept of *namespaces* to separate extensions from the actual scene definition. Node types thus can belong to separate “areas of responsibility”, while still achieving compatibility with the X3D standard. For our extensions, a separate namespace is however out of the question, because we need the hierarchical combination of both scene and multimedia. Avalon does not concern itself about advanced multimedia processing, which is why it also does not fulfill our explicit media specification requirements (1) and (2).

In [Seibert and Dähne 2006], a VRML scene graph with custom nodes is used to express the data flow and preprocessing of the different input and output devices used in mixed reality applications. This work solved problems which are quite similar to multimedia processing. In both cases, different devices have to be connected with filtering and conversion steps in between. The data flow between devices and further processing steps of the produced media data are expressed with routes between the nodes of a VRML scene. In contrast to our approach, the presented work does not concern itself with data transmission across the Internet and does not provide a scalable transparency to hide or explicitly specify details of processing and transmission according to the requirements of the scene graph developer (see requirements (4) and (5)).

3 Multimedia Processing

In contrast to previous work available multimedia frameworks are suitable to fulfill all five requirements stated in Section 4 and provide suitable concepts to the application and application developer for distributed media processing and control. In the rest of this Section we discuss these concepts and how they fulfill the stated requirements.

3.1 Explicit Specification of Media Processing and Streaming Media

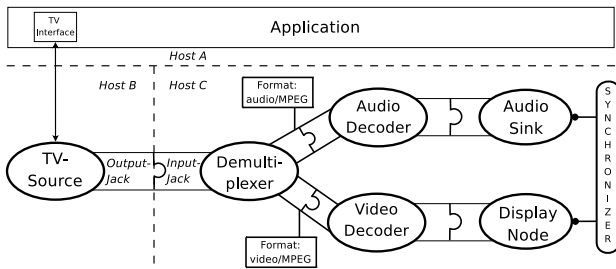


Figure 2: Distributed flow graph for watching live TV. Nodes of a flow graph represent processing units while edges represent connections between nodes. The synchronizer is responsible for synchronized media playback of audio and video streams.

Distributed multimedia middleware solutions like the Network-Integrated Multimedia Middleware (NMM) [Lohse et al. 2008] and NIST II [Fillinger et al. 2008] are especially designed to fulfill requirements (1) and (2) as stated in Section 1. A distributed multimedia middleware considers the network as an integral part of its architecture and uses a *distributed flow graph* to explicitly specify and distribute media processing.

As shown in Figure 2 the *nodes* of a flow graph represent the smallest kind of processing units, e.g., a TV tuner card, software decoder, or display, and can be explicitly specified by a unique name like `TVSourceNode`. In contrast to locally operating multimedia frameworks like Microsoft’s `DirectShow` [Microsoft 2009], or Apple’s `QuickTime` [Apple Inc. 2009], the nodes of a distributed flow graph can be distributed across the network while the application still has transparent access to all local and remote nodes. Each node can have an arbitrary number of *input jacks* and *output jacks* to receive or send processed media data, respectively. To distinguish between different jacks of a node, each jack is specified by a unique string called *jack tag*.

Nodes are connected by connecting an output jack with the input jack of the succeeding node and represent the *edges* of the distributed flow graph. Furthermore, edges hide all specific aspects of data transmission within a *transport strategy*. The strict separation of media processing and media transmission within a distributed flow graph enables the middleware to automatically configure a suitable transport strategy. When connecting nodes that run within the same address space, a distributed multimedia middleware selects a transport strategy that simply forwards pointers to processed media data between nodes. Only in case of connecting distributed nodes, the middleware establishes a network connection, e.g., using TCP, to transmit media data between the nodes.

To ensure that only nodes are connected that can process the same kind of media data, each jack provides a specification of supported media data, called *media formats*. The most common approach to specify media formats is to describe data with *major type*, e.g., audio, or video, followed by a *minor type* like `mpeg`. Finally,

each format includes specific parameters, e.g., *resolution* or *bitrate* as key-value pairs. This flexible approach to describe a media format allows to handle the huge variety of available media formats. Two nodes can only be connected if the format of the input- and output-jack are identical. Formats are identical if and only if their major and minor-type are identical, the same keys occur in all formats, and the corresponding values are identical.

Finally, a *synchronizer* is responsible for synchronized media playback of different streams, e.g., for audio and video. As soon as multiple media streams are processed, it is essential for the applications to specify which streams have to be synchronized or not. Typical multimedia applications have to synchronize at least the sink nodes of a flow graph for synchronized audio and video playback. However, if multiple live sources are used, e.g., a camera and a microphone, even the sources of a flow graph have to be synchronized.

In contrast to using a URL like in X3D, a typical multimedia application specifies the type of multimedia processing by connecting nodes to a directed distributed flow graph. The structure of the distributed flow graph specifies the overall multimedia operation to be performed. The most important aspect of a multimedia application is that requirement (1) is fulfilled and media processing can be explicitly specified.

Using a distributed flow graph for integrating multimedia processing into X3D, automatically integrates streaming capabilities (requirement (2)), because media data from remote components can be used transparently while still being able to explicitly specify the media processing. Thus, a distributed multimedia middleware using the concept of a distributed flow graph is used as basis to extend X3D by multimedia functionality.

3.2 Control of Components

To enable transparent control of local and remote components like nodes, which is stated as requirement (3) in Section 1, a distributed multimedia middleware performs a strict separation of method invocation and method execution. *Interfaces* perform only method invocation and can be used within the application. The distributed multimedia middleware delegates a method invocation on a specific interface to the corresponding local or remote component for execution.

Furthermore, interfaces are used as a *functional description* of components, especially nodes. This description of nodes is especially important for media processing within a distributed environment where a node with a specific name cannot be assumed to be available on all systems or has different names on different platforms. Instead, interfaces of nodes can be used within a distributed flow graph to specify the required functionality of the node. In the context of our previous example of a flow graph for watching TV, an application can specify a node as source that supports the `TVInterface` instead of specifying a node with name `TVSourceNode`. Distributed multimedia middleware solutions like NMM automatically find nodes – e.g., a `TVSourceNode` – within the network, that support this interface [Lohse et al. 2002].

3.3 Control of Media Transmission

As already stated in requirement (4) of Section 1, a multimedia application requires full control on media transmission. This is required to cope with two aspects of distributed media transmission, especially within the Internet.

First of all, an application has to be able to specify the utilized network protocol for media transmission. This feature is required to

consider the timing requirements of media data. Reliable transport protocols like TCP, which is also used for HTTP streaming, are suitable to cover standard scenarios where a file is completely downloaded before being processed, which is the standard approach of media processing in X3D. Here, lost packets are retransmitted.

However, in case of a live stream, e.g., received from a TV source, media data have timing constraints that have to be considered to achieve continuous playback. In general, retransmission of lost packets disagrees with the timing constraints of live streams, where it is acceptable to sacrifice reliability in favor of meeting continuous playback. For this purpose, user-level protocols like the *realtime transport protocol* (RTP) [Schulzrinne et al. 1996] were proposed that meet special requirements of media transmission. Moreover, the RTP protocol provides additional transmission statistics, e.g., lost packets, and transmission delay, that can be used by an application to react to a bad network connection. Since the decision on the most suitable transport protocol as well as how to react to bad network connections strongly depends on the application scenario. The application itself needs the possibility to specify the used transport protocol as well as to access its underlying parameters and transmission statistics.

Therefore a distributed multimedia middleware also represents the edges of a distributed flow graph as first class object to the application. This allows an application to explicitly specify the used transport protocol and to access the parameters or transmission statistics of the underlying network protocol. However, advanced distributed multimedia solutions like NMM offer a flexible and extendable adaptation service to the application that can be used to automatically react on variations of data transmission [Replinger et al. 2009]. Since these aspects are essential for media transmission within the Internet, they should be considered when integrating multimedia processing into web standards like X3D.

3.4 Abstract Specification of Media Processing

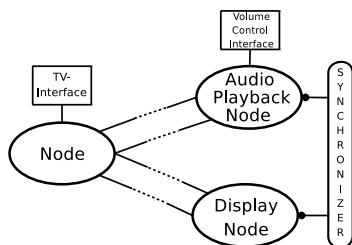


Figure 3: This Figure shows a user graph for watching TV. A user graph is a high-level specification of a flow graph that only includes the essential nodes and edges of a flow graph from the applications point of view. Moreover, interfaces like the *TVInterface* can be used to specify the abstract functionality of a node.

The last requirement (5) stated in Section 1 is also supported by distributed multimedia middleware solutions. Due to the complexity of distributed multimedia applications, a developer should be supported by offering an abstract specification of media processing. Typically, most multimedia applications do not need full access to all nodes or edges of the flow graph.

For this reason, most multimedia middleware solutions additionally offer the concept of a *user graph* [Lohse and Slusallek 2005]. A user graph is a high-level specification of a flow graph that only includes the essential nodes as can be seen in Figure 3. Here, only the key nodes, their connections, and additional constraints are specified. The most important impact for the application developer is that

a user graph can be independent of concrete nodes that are used. Instead, other aspects of nodes like interfaces are used to specify the required functionality of nodes.

In Figure 3, the abstract description of an multimedia application for watching TV can be seen. This user graph is then automatically mapped to the flow graph that can be seen in Figure 2 while the specified constraints, e.g., a specified format or transport protocol are considered. Another important aspect here is that a distributed multimedia middleware automatically searches the entire network or a set of specified locations to find nodes required to set-up a complete distributed flow graph.

4 Integrating Multimedia into X3D

In the following, we will use the requirements defined in Section 1 as a guideline to illustrate our integration of multimedia processing into the X3D language specification. Since our X3D extensions have to fulfill the requirements stated in Section 1, we integrate all of the concepts of multimedia processing presented in Section 3.

4.1 Explicit Specification of Media Processing

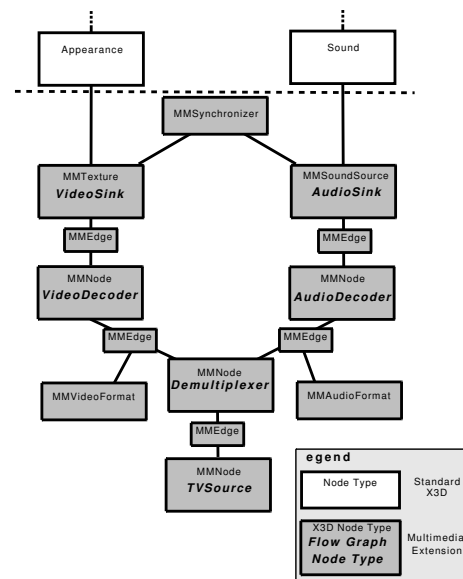


Figure 4: The general idea of integrating a multimedia flow graph into an X3D scene graph is to specify the flow graph including all components as a sub-tree within the X3D scene. Since the sink nodes of a flow graph represent the connection between produced media data and their representation within X3D, they are connected as children of standard X3D nodes. Thus, the source nodes of a flow graph represent leaf nodes of the full X3D scene graph.

To fulfill requirement (1), we first extend the X3D language specification by the concept of a multimedia flow graph. This approach provides the maximum flexibility to specify media processing as discussed in Section 3.1. The general idea of this multimedia sub-graph in an X3D scene is depicted in Figure 4.

To integrate a flow graph into an X3D scene graph we specify the entire flow graph including all components as sub-tree inside the X3D scene. The sink nodes of a flow graph represent the connection between produced media data and their representation within X3D. Therefore, the sink nodes of a flow graph are connected as child nodes of classic X3D nodes responsible for media specification.

The source nodes of a flow graph represent leaf nodes within an X3D scene graph.

To clearly define multimedia processing within an X3D scene graph, we introduce the following new X3D nodes:

- **MMNode:** A *multimedia node* represents a single node of a multimedia flow graph within the X3D scene graph. For explicit specification of media processing it only includes a `nodeName` field to specify the name of the node within a flow graph. In a first step, a multimedia node is specified as follows:

```
MMNode {
  SFString [in,out] nodeName ""
  MFNode [in,out] inEdges [] [MMEdge]
  SFNode [in] addEdge [MMEdge]
  SFNode [in] removeEdge [MMEdge]
}
```

- **MMEdge:** A *multimedia edge* represents an edge of the flow graph, connecting two multimedia nodes. Within the specification, a multimedia edge is always a child of an `MMNode`. Its `target` field always contains this parent node to enforce that the edge always connects to exactly two `MMNode` instances. It includes all the required information to describe a flow connection between two multimedia nodes. We specify a multimedia edge as follows:

```
MMEdge {
  SFNode [in,out] source NULL [MMNode]
  SFNode [in,out] target NULL [MMNode]
  SFString [in,out] inJackTag "default"
  SFString [in,out] outJackTag "default"
  SFNode [in,out] format NULL [MMFormat]
}
```

- **MMFormat:** A *multimedia format* describes the media format, which should be used on the connection between two multimedia nodes. Since there is a vast number of multimedia formats with very different properties that should be exposed to the scene, we defined an abstract node type `MMFormat` from which concrete format types inherit. To allow specifying only the major type of a format, we additionally introduce the `MMVideoFormat` and `MMAudioFormat` node types.
- **MMTexture:** A *multimedia texture* represents the video output of a flow graph and includes the video stream as a texture into the X3D scene graph. Therefore, this X3D node acts as a link between the flow graph and the X3D scene graph. Since it represents a sink in the context of a flow graph it inherits from `MMNode`. To enable its usage as a texture within the scene, it furthermore inherits from `X3DTexture2DNode`, so it can be used in the texture field of an `X3D Appearance` node. Together, the specification of this new X3D node is as follows:

```
MMTexture: X3DTexture2DNode, MMNode {
  SFNode [in,out] inEdge NULL [MMEdge]
  [+inherited fields from X3DTexture2DNode]
}
```

- **MMSoundSource:** A *multimedia sound source* represents the link between audio output of media processing within the flow graph and a source of audio data within the X3D scene graph. Therefore, it inherits from `MMNode` as well as from the standard X3D type `X3DSoundSource`. It can thus be seamlessly used in the `source` field of an `X3D Sound` node. Its specification is as follows:

```
MMSoundSource : X3DSoundSourceNode {
  SFNode [in,out] inEdge NULL [MMEdge]
```

```
[+inherited fields from X3DSoundSourceNode]
}
```

- **MMSynchronizer:** The *multimedia synchronizer* represents a synchronizer of the flow graph and has an arbitrary number of multimedia nodes as children. For specification, this node is required to define which multimedia nodes – or the corresponding nodes of the underlying flow graph, respectively – have to be synchronized by the utilized multimedia middleware. Since all valid children are of type `MMNode`, the specification of this new X3D node is as follows:

```
MMSynchronizer {
  MFNode [in,out] children NULL [MMNode]
  SFNode [in] addChild [MMNode]
  SFNode [in] removeChild [MMNode]
}
```

4.2 Streaming Media

To master requirement (2) as stated in Section 1, we extend our previously defined `MMNode` to be able to specify *distributed* multimedia nodes. Using a distributed multimedia middleware, the specification of remote components is limited to specify either a single location, or a set of locations from where to request a remote component.

The benefits of our approach, using a distributed multimedia middleware are shown here: We integrate streaming functionality just by extending the specification of the `MMNode` type by a `location` field:

```
MMNode {
  SFString [] nodeName ""
  MFNode [in,out] inEdges [] [MMEdge]
  SFNode [in] addEdge [MMEdge]
  SFNode [in] removeEdge [MMEdge]
  MFString [in,out] location [] [URI]
  SFString [in] addLocation [URI]
  SFString [in] removeLocation [URI]
}
```

The newly introduced field `location` stores an arbitrary number of locations which are used to request the corresponding flow graph node. If the field is not set, the underlying multimedia middleware searches all reachable systems to request the node. A single specific location can be set by adding exactly one entry to this field.

4.3 Control of Components

To meet requirement (3) as stated in Section 1, nodes of a flow graph should be controllable from within an X3D scene graph. As mentioned in Section 3.2, multimedia components are controlled by *interfaces* used from within a controlling application. In the case of X3D being the language of application definition, we need to introduce a concept of interfaces inside of X3D: We represent an interface by an X3D node derived from the abstract type `MMInterface`. A concrete implementation thereof then defines the fields necessary to work with the interface in an X3D context. For example, a derived `TVInterface` would at least define one `channel` field, which is readable and writable and thus can be used for routing within the X3D scene. Internally, the underlying multimedia technology has to provide the mapping to actual method calls to manipulate the switching of the channel on the actual hardware. How this is accomplished in our implementation will be elaborated in Section 6.

To assign the functionality of a multimedia interface to a multimedia node in X3D, we add the respective interface node to the `interfaces` field of a multimedia node, which may contain an

arbitrary number of interfaces. Here, each interface represents a self-contained group of functionality exposed by its parent multimedia node. This enables multimedia nodes to be defined completely abstract, just as a collection of interfaces it supports. The resolution towards a concrete implementation is done by the multimedia system underneath.

Summing up, the complete specification of a multimedia node within X3D is now defined as follows:

```
MMNode {
  SFString  []      nodeName      ""
  MFNode    [in,out] inEdges      [] [MMEdge]
  SFNode    [in]    addEdge        [MMEdge]
  SFNode    [in]    removeEdge     [MMEdge]
  MFString  [in,out] location      [] [URI]
  SFString  [in]    addLocation    [URI]
  SFString  [in]    removeLocation [URI]
  MFNode    [in,out] interfaces    [] [MMInterface]
  SFNode    [in]    addInterface   [MMInterface]
  SFNode    [in]    removeInterface [MMInterface]
}
```

4.4 Control of Media Transmission

To master requirement (4) stated in Section 1, we need to be able to specify the utilized networking protocols for media transmission inside the X3D scene graph. As mentioned in Section 3.3, *transport strategies* can be specified on the edge representing the connection between two nodes. To maintain type safety within the X3D specification, we introduce a new abstract node `MMTransportStrategy`.

A concrete implementation of a transport strategy node then defines a number of fields that allow the X3D scene to interact with it. For example, a derived `RTPInterface` would at least include the information provided by transmission statistics (e.g., `packetLoss`) as out-values for being able to route them within the X3D scene.

To assign the functionality of a transport strategy interface to a multimedia edge in X3D, we extend the edge by a field named `transportStrategy` and add the respective interface node to this field, which may contain an arbitrary number of multimedia transport strategy interfaces. The specification of a multimedia edge is thus extended as follows:

```
MMEdge {
  SFNode    [in,out] source        NULL    [MMNode]
  SFNode    [in,out] target        NULL    [MMNode]
  SFString  [in,out] inJackTag     "default"
  SFString  [in,out] outJackTag    "default"
  SFNode    [in,out] format        NULL    [MMFormat]
  MFNode    [in,out] transportStrategy [] [MMTransportStrategy]
}
```

Notable in this context is the simplicity achieved to react on network variations provided to the X3D scene graph developer, because information about connection quality can be directly routed to other X3D components and handled appropriately. In Section 5.2 we explain a scene graph that automatically adjusts the bitrate of transmitted media streams with regard to lost packets.

4.5 Abstract Specification of Media Processing

To master the last requirement (5) as stated in Section 1, we integrate the concept of a user graph as described in Section 3.4 into X3D. A media processing subgraph in X3D is always mapped onto the user graph of the distributed multimedia middleware because it

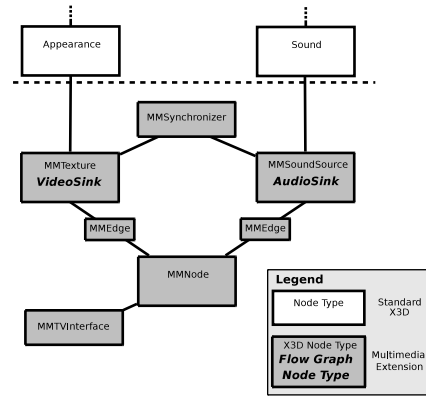


Figure 5: The abstract specification within an X3D scene enables an easy integration of distributed media processing and control. In this example for watching TV, only those aspects relevant for the scene graph developer are explicitly specified, the rest is configured automatically by the multimedia middleware.

allows to explicitly specify either a full flow graph or only components that are important for the application.

Thus, this integration of media processing and control into X3D provides a scalable transparency for the scene graph developer when specifying multimedia content. The most basic description of a user graph within X3D consists of multimedia nodes that represent sources and sinks of the flow graph. Since the sink nodes of the flow graph are either represented by X3D nodes of type `MMTexture` or `MMSoundSource`, a scene graph developer first adds edges as children to these multimedia nodes and then the sources of a flow graph as leaf nodes. The sources of a flow graph have to be specified either by setting the field `nodeName` or by adding a descriptive interface. Finally, the developer has to add a multimedia synchronizer as parent of those X3D nodes representing the multimedia sinks (`MMTexture` or `MMSoundSource`), if their media playback should be synchronized.

An example of such an abstract description can be seen in Figure 5 which shows the abstract specification for watching TV. Here, the multimedia node representing a TV source is specified by using a multimedia interface of type `TVInterface`. Even though the specification of this scenario is quite simple, it highlights the benefits provided by our integration of media processing into X3D. Firstly, live streams can be used within X3D; secondly, remote resources are found automatically if no local TV tuner card is available; and thirdly, controlling the TV card is seamlessly integrated by changing the `channel` field of the `MMTVInterface`.

5 Application Scenarios

In this Section, we will show you how to specify two exemplary applications with the X3D extensions presented before, and how even complex scenarios can be realized through an intuitive and simple specification.

5.1 Virtual TV

To include a live television feed in a 3D scene, we start at the locations in the X3D graph where the actual media should be mapped to: an `Appearance` node below the TV screen geometry to assign the video texture to, and one `Sound` node for each audio channel to be placed in the scene. For this example, we use the capabilities of X3D and assign two mono sound sources to speaker ge-

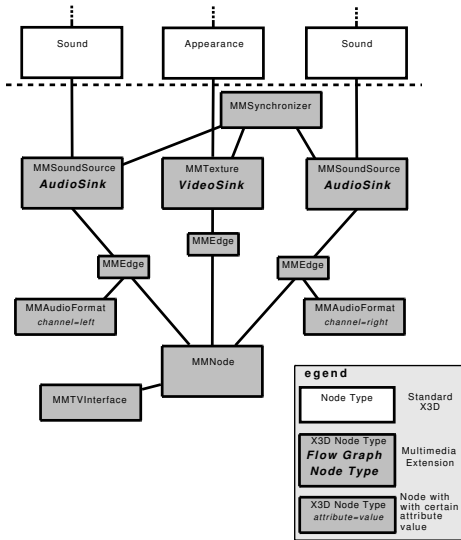


Figure 6: Full control over the multimedia flow graph allows scene authors to define different locations for the separate channels of stereo sound.

ometry in the scene, being able to define the directional characteristics of each source. Below those native X3D texture and sound nodes in the graph, we then attach an `MMTexture` node and two `MMSoundSource` nodes. They represent the sinks of the underlying multimedia flow graph and need to be explicitly specified as targets inside the scene. As the three nodes are supposed to be synchronized, i.e., both audio channels should temporally match the video, we attach them all to a common `MMSynchronizer`, indicating their connection to the multimedia subsystem.

At the opposite end of the multimedia flow graph to be specified, we need one X3D node that represents the TV card hardware to be used as source for the live television feed. For this purpose, we place a single `MMNode` at the bottom of the graph. If we wanted to explicitly use a dedicated implementation inside the X3D node, we could specify a concrete node of the multimedia flow graph in `nodeName` field of the `MMNode`. But we would like to keep the implementation generic and executable on any TV card that is available. Therefore, we only attach a control interface of type `TVInterface` (derived from the abstract `MMInterface`) to our source node, and let the multimedia system decide for a concrete implementation supporting the given interface.

Now, the only thing missing is the connection inbetween the TV card and the sinks, all the above defined in the scene by the respective X3D nodes. We therefore connect the multimedia texture and sound nodes via an instance of the `MMEdge` node to the `MMNode` that abstracts the TV card. The final adjustment to make the scenario entirely unambiguous is attaching a node of type `MMFormat` to each audio edge, inside of which we specify the type of sub-stream that is supposed to flow through its parent. In our scenario, the multiplexed TV stream is separated into a video part, an audio part for the left stereo channel, and an audio part for the right stereo channel. The final setup of X3D nodes used to realize this scenario is again shown in Figure 6.

5.2 Dynamic Bitrate Adaptation

This scenario is supposed to highlight our concept of scalable transparency by realizing a complex adaptation scenario – as described for instance in [Replinger et al. 2009] – entirely within an X3D

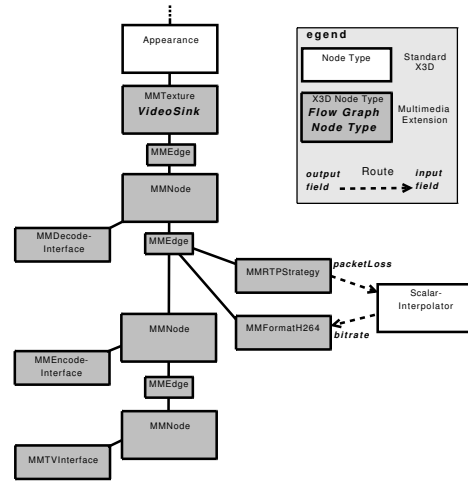


Figure 7: Dynamically adapting the bitrate to the connection quality can be expressed directly in the scene using only two routes and an X3D `ScalarInterpolator` node.

scene. The basic idea of the adaptation to be implemented is monitoring the number of lost packets on an RTP network connection between an encoder and a decoder node, and deciding on adjusting the encoding bitrate if that number gets too high.

We start with the video branch of the TV graph described in Section 5.1, i.e., an `MMTexture` node connected to an `MMNode`, which abstractly defines a TV card though the respective child interface. We extend this graph by specifying two additional nodes inbetween, which are to encode the video stream, and decode it again. Again we specify the used nodes abstractly by attaching the respective `EncoderInterface` and `DecoderInterface` nodes (which are both derived from `MMInterface`) to an otherwise unspecified instance of `MMNode` each. In order to make this setup unambiguous in the range of available encodings, the format on the connecting edge must be explicitly specified in the form of an `MMFormat` node attached to the connecting edge. In this case, a `MMVideoFormatH264` node is used to specify the H.264 video format. Also, in a real-world setting, the encoder and decoder components are usually located on different machines to legitimate the encoding effort.

We assume the encoder and decoder – as well as the TV card and X3D browser, respectively – to be located on separate hosts connected through the Internet, using the Real-Time Transport Protocol (RTP) on the connection inbetween. We express this fact through our X3D extensions by attaching a node of type `RTPStrategy` (derived from `MMTransportStrategy`) to the connecting edge node between the encoder and decoder nodes. This already concludes our basic flow setup, which enables an encoded video transmission with an explicitly specified encoding format, and an explicit transport protocol to use.

However, we can push the scenario one step further by attaching both the read-only `packetLoss` field of the RTP strategy node and the exposed `bitrate` field of the connection format node to a `ScalarInterpolator`. The interpolator uses the lost packets as input value and calculates a new bitrate for the encoding as output value. Even though the calculation is very simple, it shows the power of the scalable transparency approach allowing to specify only those interfaces and fields that are actually needed for disambiguation or runtime manipulation. The interpolator in this example could also be replaced by a `Script` node with arbitrary inputs and outputs, allowing to specify the most complex control circuits di-

rectly inside X3D. Figure 7 shows the node setup for the described adaptation scenario again graphically.

6 Implementation

We implemented our proposed extension of X3D for distributed multimedia processing and control using the Network-Integrated Multimedia Middleware (NMM) and the RTSG scene graph library [Georgiev et al. 2008]. RTSG already supports VRML and X3D and is easily extendable by additional rendering components.

An important aspect of RTSG is its strict separation of the scene graph and the rendering components. In contrast to the usual definition, rendering in RTSG refers not only to graphical output but instead refers to any kind of presenting the scene, also for example in the form of sound output. This allows RTSG to combine multiple specialized rendering components, each of which supporting only a subset of all X3D nodes. This feature enables us to easily integrate the new multimedia extensions by adding a new rendering component.

6.1 Implementation of Multimedia Renderer

First of all, our current implementation on top of RTSG includes all X3D nodes we described in Section 4. Furthermore, we realized one multimedia interface of type `TVInterface` as an X3D node that enables channel switching on the used TV board and one transport strategy interface of type `RTPStrategy` to be informed about lost packets. The limited number of realized interfaces is motivated by the fact that NMM describes all its interfaces in an interface definition language (IDL). A corresponding IDL compiler generates source code that can be used by all NMM applications. We plan to add a different implementation of the IDL compiler that automatically maps interfaces specified within an IDL to X3D interface nodes. Even though the complete implementation of the new IDL compiler is not yet finished, we could evaluate our approach with the realized interface.

Furthermore, our implementation extends RTSG by a new *multimedia renderer* that is able to handle the newly added X3D nodes. The multimedia renderer traverses the X3D scene graph once when it is loaded and searches for defined multimedia nodes. Then, it maps the abstract multimedia processing defined in these nodes to the corresponding component of an NMM user graph.

We do not yet support modification, addition or removal of multimedia nodes or edges at runtime. We do, however, support the input and output fields of the `MMInterface`, `MMFormat` and `MMTransportStrategy` nodes, which allow the scene to modify the multimedia flow graph during runtime, or be informed about changes within the flow graph. To accomplish that, the multimedia renderer registers itself as listener for all input values. As soon as one input value of the newly added X3D nodes is changed, the multimedia renderer is informed, maps this value to a method invocation and invokes the method together with the changed value on the corresponding component of the user graph.

On the other hand, to inform the X3D scene about changed out-values of supported X3D nodes, the multimedia renderer registers itself as listener to the corresponding NMM component. As soon as a specific value within this component of the flow graph of NMM is changed – e.g., packet loss occurs during network transmission –, the renderer will therefore be informed. It will then change the value of the corresponding field of the X3D node and inform RTSG to route this information to all connected X3D node fields.

6.2 Implementation of NMM Nodes

Since all specified multimedia nodes of an X3D scene are directly mapped to nodes of a flow graph, we needed to implement new NMM nodes for the special X3D nodes `MMTexture` and `MMSoundSource`, which realize the connection between X3D and multimedia processing.

An X3D node of type `MMSoundSource` is mapped to the NMM node `AL3DPlaybackNode`. This NMM node enables NMM to play 3D sound features by using the OpenAL library [Creative Labs 2007]. Using OpenAL enables advanced features like Doppler effect or delayed playback based on the distance to the sound source in a 3D environment. Moreover, OpenAL allows to playback 3D audio through multi-speaker environments like 7.1 speaker setups. The multimedia renderer sets the current viewer position and the position of the sound sources in the `AL3DPlaybackNode`, including all transformations. For this, it keeps an internal representation of the relevant parts of the transformation hierarchy of the scene graph and uses listeners to be notified about any changes in the X3D scene. These changes are propagated to the `AL3DPlaybackNode` which in turns forward this information to OpenAL.

An X3D node of type `MMTexture` is mapped to the NMM node `TextureSinkNode`. This node uses a texture buffer of the rendering engine that is currently responsible for rendering textures. The `TextureSinkNode` requests at least two memory blocks for a specific texture to enable double buffering. As soon as a new video frame arrives at the `TextureSinkNode`, it copies the video frame to a free memory block for the corresponding texture and informs the texture renderer to use the texture buffer including the latest video frame. This approach automatically prevents any synchronization issues with the texture renderer because the renderer simply uses the latest texture buffer available. If the texture renderer achieves a higher frame rate as the assigned instance of the NMM `TextureSinkNode` delivers new frames, the same video frame is presented multiple times. If the texture renderer renders textures with a lower frame rate, some video frames can not be rendered, but the texture renderer always renders the latest video frame as texture.

We used our entire implementation of X3D extensions for distributed multimedia processing and control to realize the two application scenarios described in Section 5. Moreover, we realized an X3D scene that presents the the image of a TV source as can be seen in Figure 1. Here, different postprocessing steps are added after decoding the video data: adaptation of brightness, and pixel-based compositing for mapping logo graphics on top of the video stream. For postprocessing we used available NMM nodes that internally use the ImageMagick library [Still 2005]. In all application scenarios, there was no noticable performance hit by integrating the multimedia flowgraph into the X3D scene, neither on the rendering, nor on the multimedia processing part.

7 Conclusion and Future Work

This paper dedicated itself to a full inclusion of multimedia processing components inside an X3D scene specification. We identified five requirements of multimedia processing an X3D application definition should be able to include: explicit specification of a multimedia flow graph inside X3D, the ability to handle streaming media, the possibility to control single processing elements via dedicated interfaces, the ability to fine-tune network connections and specify transport protocols, and the possibility to reduce the specification to only those parts that are absolutely necessary for disambiguation or runtime reconfiguration of the multimedia flow.

We showed how the concept of a distributed multimedia flow graph is applied in modern multimedia software to realize all of the defined requirements, and introduced a minimal set of X3D nodes to map a generic multimedia flow graph to an X3D scene graph. We introduced further X3D extensions for defining control interfaces, formats and strategies for network transport inside the scene. We incorporated a policy of scalable transparency in our extensions, such that scene authors only have to specify a minimal multimedia flow to make it unambiguous, and include only those detailed fields of an X3D node they need to realize a desired functionality. In two application examples we showed how to build up a multimedia flow from within X3D, and how to realize even seemingly complex adaptation functionality by employing simple X3D routing. We then detailed on our implementation of the presented concepts using the Real-Time Scene Graph (RTSG) and the Network-Integrated Multimedia Middleware (NMM). In summary, we showed that our concept and implementation fulfills the five requirements of a full multimedia integration into X3D.

At the moment, the entire X3D description including the multimedia flow is purely textual and not integrated into a higher-level editor. For the future, we envision a combined graphical editor for both X3D scene geometry and logic as well as the additional multimedia flow, allowing for instance to visually attach video and audio sinks to scene surfaces, or visually specify the directional characteristics of a sound source more intuitively by drawing the respective ellipsoids directly into the 3D scene.

Last, there is more future work necessary in terms of standardization of interfaces: Currently, we use our own definition of what functionality, for example, a TV interface should offer. By generating the respective X3D interfaces directly out of the IDL descriptions of NMM, we do offer the full functionality of NMM inside X3D, but might miss important other views onto the same problem. A final goal should be to define a standard set of multimedia nodes, which covers the full functionality needed for multimedia integration, but stays extendable for special purpose hardware or tasks. A first step into that direction would be having a different multimedia back-end provide its implementation to our X3D node concept, and evaluate the suitability for the entire range of applications again.

Acknowledgements

We would like to thank Wolfgang Burgard for his valuable work on sound rendering for RTSG, which constituted the first step towards a full integration of NMM-based multimedia processing into our X3D implementation.

References

ADOBE, 2009. SWF File Format Specification. Version 10. http://www.adobe.com/devnet/swf/pdf/swf_file_format_spec_v10.pdf.

APPLE INC., 2009. QuickTime Technologies. <http://www.apple.com/quicktime/technologies>.

BEHR, J., AND DÄHNE, P. 2003. AVALON: Ein komponentenorientiertes Rahmensystem für dynamische Mixed-Reality Anwendungen. In *Thema Forschung*, 66–73.

BRUTZMAN, D., AND DALY, L. 2007. *X3D: Extensible 3D Graphics for Web Authors*. Morgan Kaufmann.

BRUTZMAN, D., AND KOLSCH, M. 2007. Video Requirements for Web-based Virtual Environments using Extensible 3D (X3D) Graphics. <http://www.w3.org/2007/08/video/positions/Web3D.pdf>.

CREATIVE LABS, 2007. OpenAL. <http://openal.org>.

FILLINGER, A., DIDUCH, L., HAMCHI, I., HOARAU, M., DEGR, S., AND STANFORD, V. 2008. The NIST Data Flow System II: A Standardized Interface for Distributed Multimedia Applications. In *IEEE International Symposium on a World of Wireless; Mobile and MultiMedia Networks (WoWMoM)*, IEEE.

GEORGIEV, I., RUBINSTEIN, D., HOFFMANN, H., AND SLUSALLEK, P. 2008. Real Time Ray Tracing on Many-Core-Hardware. In *Proceedings of the 5th INTUITION Conference on Virtual Reality*.

ISO/IEC 14496-11, 2007. Scene description and Application engine(BIFS).

LOHSE, M., AND SLUSALLEK, P. 2005. Towards Automatic Setup of Distributed Multimedia Applications. In *Proceedings of The 9th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA)*, ACTA Press, 359–364.

LOHSE, M., REPPLINGER, M., AND SLUSALLEK, P. 2002. An Open Middleware Architecture for Network-Integrated Multimedia. In *Protocols and Systems for Interactive Distributed Multimedia Systems, Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems, IDMS/PROMS 2002, Proceedings*, Springer, vol. 2515, 327–338.

LOHSE, M., WINTER, F., REPPLINGER, M., AND SLUSALLEK, P. 2008. Network-Integrated Multimedia Middleware (NMM). In *MM '08: Proceedings of the 16th ACM international conference on Multimedia*, 1081–1084.

MICROSOFT, 2009. DirectShow Architecture. <http://msdn.microsoft.com/>.

MÜLLER, P., SCHUBIGER-BANZ, S., MÜLLER ARISONA, S., AND SPECHT, M. 2006. Interactive media and design editing for live visuals applications. In *International Conference on Computer Graphics Theory and Applications (GRAPP)*, 232–241.

MÜLLER, S., SCHUBIGER-BANZ, S., AND SPECHT, M. 2006. A real-time multimedia composition layer. In *AMCMM '06: Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, ACM, New York, NY, USA, 97–106.

REPPLINGER, M., LÖFFLER, A., THIELEN, M., AND SLUSALLEK, P. 2009. A Flexible Adaptation Service for Distributed Rendering. In *Proceedings of Eurographics Parallel Graphics and Visualization Symposium (EGPGV 2009) (to appear)*.

RYMASZEWSKI, M. 2008. *Second Life: The Official Guide*. Sybex.

SCHULZRINNE, H., CASNER, S., FREDERICK, R., AND JACOBSON, V., 1996. RFC 1889: RTP: A Transport Protocol for Real-Time Applications. <http://www.ietf.org/rfc/rfc1889.txt>.

SEIBERT, H., AND DÄHNE, P. 2006. System Architecture of a Mixed Reality Framework. *Journal of Virtual Reality and Broadcasting* 3, 7 (May). urn:nbn:de:0009-6-7774, ISSN 1860-2037.

STILL, M. 2005. *The Definitive Guide to ImageMagick*. Apress.