# Middleware Support for Seamless Multimedia Home Entertainment for Mobile Users and Heterogeneous Environments

Marco Lohse and Philipp Slusallek
Computer Graphics Lab, Department of Computer Science
Saarland University, Saarbrücken, Germany
{mlohse, slusallek}@cs.uni-sb.de

**ABSTRACT**
The emergence of mobile devices in multimedia home entertainment demands new application scenarios like ubiquitous multimedia access. However today's home entertainment appliances are usually based on a closed hardware and software design and do not provide the extensibility and flexibility needed.

In this paper, we present a middleware that allows to control and connect distributed and mobile multimedia devices with different underlying technology. Based on this middleware we demonstrate an extensible application framework for a multimedia home entertainment center. This framework provides session hand off for seamless multimedia playback in heterogeneous environments where users with mobile devices can transparently exploit the capabilities of nearby stationary systems.

**KEY WORDS**
Distributed Multimedia Systems, Mobile Multimedia, Multimedia Home Entertainment Applications

## 1 Introduction

The market for home entertainment systems is currently dominated by inflexible, proprietary, and closed hardware and software systems. While traditional systems like radio, TV, and CD players are still widely used, there is a clear trend towards media convergence, where most of the home entertainment options will be based on digital media existing in various formats.

With the availability of cheap, small, quiet, and powerful multimedia PCs on the one side, and programmable mobile devices like PDAs on the other side, it is now possible to create an open and extensible platform for multimedia home entertainment. In such a scenario, the user can use a mobile device to access multimedia data and then transparently use the resources of stationary devices for media decoding and playback.

In this paper, we describe a flexible application framework supporting such scenarios. The framework is built on top of the Network-Integrated Multimedia Middleware (NMM) that allows to locate and control distributed multimedia devices and software components of different underlying technology. Furthermore, local and remote components can be integrated into a common flow graph of processing units.

The overall system architecture consists of stationary Linux PC-based systems that provide an integrated solution for playing CDs or DVDs, TV access with time-shifting, browsing and playback for various supported media formats, and features like transcoding to different media formats. The architecture supports the use of remote devices like special hardware (e.g. a board for receiving digital TV) and the distribution of time consuming tasks like video transcoding to remote systems using the facilities of the underlying middleware. Additionally, users can initiate playback on mobile devices, or transparently hand off playback sessions to stationary systems in order to access their computational resources and rich I/O capabilities (e.g. high-quality audio output or video rendering on large displays). Furthermore, the usage of the underlying middleware also allows to present the same user interface and interaction possibilities on stationary and mobile devices.

## 2 Related Work

Today's common multimedia middleware such as DirectShow from Microsoft [1], Apple's Quicktime [2], or the Java Media Framework from Sun [3] adopt a PC-centric approach that only supports access to directly connected devices. The network is mostly used as a data source.

Current distributed multimedia systems are mostly based on exiting middleware for distributed object computing (DOC) like OMG's CORBA [4]. These frameworks extend this middleware with several properties needed in a distributed multimedia environment. One example are the Multimedia System Services (MSS) [5] that are based on the ISO Presentation Environment for Multimedia Objects (PREMO) [6]. In contrast, the multimedia architecture described in this paper allows the transparent usage of different networking and middleware technologies (like CORBA) which is especially important in heterogeneous scenarios.

Standards for device discovery in a home networking environment like Sun's Jini [7] and HAVi [8] only partly solve the requirements mentioned above. Nevertheless, these technologies could be transparently integrated into our middleware using suitable proxies (see Section 3).

Some PC-based home entertainment centers have recently been introduced, but are still restricted to operate on a single host [9] or concentrate on the application of using remote and local storage together with Internet connectivity [10]. On the other side, the Multimedia Home Platform (MHP) [11] aims at enhancing the basic TV functionality with interactive content but still needs a middleware – like the one described in this paper – to operate on.

The idea of accessing multimedia data on nearby stationary systems is also presented in [12]. However, this approach is restricted to the IEEE 1394 networking technology. An OSGi compatible solution for location dependent playback of multimedia data is described in [13]. While OSGi provides a standard way to connect devices such as home appliances it does not provide facilities especially needed for handling multimedia, e.g. synchronization [14]. Therefore, the cited approach uses third-party components for streaming and playback of multimedia data and does not provide the extensibility provided by our multimedia middleware and application framework.

Other middleware solutions study architectural support for context-aware applications in terms of sensing the environment [15]. We are not discussing these topics in the scope of this paper, but assume that the application has knowledge about its environment – either by means of user interaction or simple location sensing via infrared transmitters.

## 3 Underlying Middleware

The *Network-Integrated Multimedia Middleware* (NMM) [16] allows to control distributed multimedia devices and software components and to integrate local and remote components into a common flow graph. The architecture offers an open architecture that does not rely an a particular technology or middleware but allows for the flexible usage of different networking and middleware technologies. Together, this offers following advantages.

- Support for heterogeneous environments through the usage of mediating proxy objects and parameterizable communication strategies. This also allows to integrate emerging new technologies easily.
- Explicit binding allows to chose the communication strategy independently for the transmission of multimedia data and the controlling of components. Appropriate parameters can be set for every connection.
- Usage of optimized communication strategies. This removes the overhead of the middleware for locally running parts of the application. Also, platforms with restricted resources, like PDAs, can be integrated by using light-weight transport strategies.
- Reflection and event notification. All components can be queried for their supported functionality and the application can register to be notified when a certain event occurs.

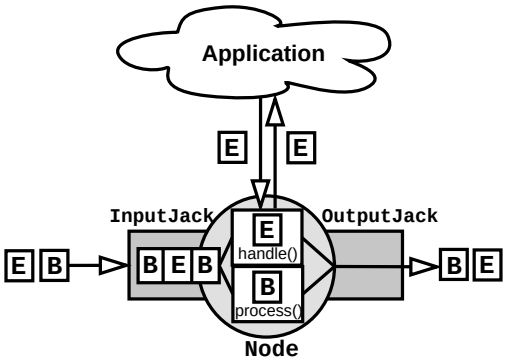


Figure 1. Node with input and output jack; messages ('E' event, 'B' buffer) are sent instream and out-of-band.

In the following we briefly describe the features of the NMM architecture that form the basis on which the multimedia home-entertainment center is created. NMM is implemented in C++ and runs under Linux. The NMM framework and applications are available as Open Source (see www.networkmultimedia.org).

### 3.1 Basic Concepts

The design approach of the local operating NMM architecture is similar to other architectures, e.g. the Java Media Framework, which makes it possible to integrate these technologies into our architecture as described below. Within NMM, all hardware devices (e.g. a DVD-ROM drive) and software components (e.g. decoders) are represented by so called *nodes* (see Figure 1). Nodes have properties that include its input and output ports, called *jacks*, together with their supported multimedia formats. The innermost loop of a node produces data, performs a certain operation on the data, or finally consumes data. These nodes can be connected to create a flow graph, where every two connected nodes support the same format. The structure of this graph then specifies the operation to be performed.

The NMM architecture uses a uniform message system for all communication. There are two types of messages. Multimedia data is placed into messages of type *buffer* that are streamed across connected jacks. In contrast messages of type *event* forward control information such as a change in speaker volume. Events are identified by a name and can include arbitrary typed parameters. Events can be generated within nodes and are then forwarded instream just like buffers, or they can be sent between the application and nodes (out-of-band). Instream messages are queued within the jacks of a node; out-of-band events are handled synchronous.

The unified message system allows for registering an application as *listener*: the application will then be notified when a certain event occurs at a node. Furthermore, the

message system provides reflection: nodes can be queried for all events they can handle together with their parameter types. Additionally, nodes can dynamically add and remove the ability to handle certain events.

We consider the reflection property and the possibility to register additional listeners to be essential parts of a multimedia middleware. This is especially important in distributed environments where new devices with unknown properties may become available at any time.

The framework offers facilities for efficient memory management, scheduling, and a sophisticated state machine for all processing nodes. Mechanisms for synchronizing different media streams with QoS control are also provided. Generic base classes exist that simplify the integration of new processing units and codecs by inheriting from a suitable class and only implementing the specific multimedia processing code.

## 3.2 Distributed Proxy Architecture

While the approach described above is similar to other multimedia middleware that operates on a single host only, our architecture allows to transparently use distributed nodes with different underlying device and networking technology. To achieve this, objects like the NMM nodes and jacks are controlled via proxy objects. Proxy objects allow for redirecting events to and from remote objects. Furthermore, proxies can act as translators between different technologies and allow to integrate middleware like the Java Media Framework (JMF) into NMM [16] by means of mediating proxies that translate between the different middleware APIs and protocols.

Finally, proxies are used to provide object oriented interfaces that allow to control objects by simply invoking methods on such interfaces, which is more type-safe then sending events. These interfaces are described in an interface definition language (IDL) that is similar to CORBA IDL. For each interface defined in the IDL, an interface class and an implementation class is generated by an IDL compiler. Internally, these classes use events, and therefore also provide the possibility to notify any listener. During runtime, any supported interfaces can be queried by the application. Interfaces are used within applications as _var types that provide memory management comparable to CORBA [4].

The bidirectional communication between proxy and its referring object is performed with so called *communication channels*. Messages sent across a communication channel are automatically serialized, transmitted, and then deserialized. For this, a communication channel internally uses a transport strategy that employs one (or more) serialization strategies. Communication channels are modeled as *first class data type* meaning that they can be used and manipulated like any other object. Therefore, communication channels provide an explicit binding as opposed to the implicit binding mechanisms that can be found in traditional middleware.

```
// connect output jack to input jack
// returns communication channel (stored in _var type)
CommunicationChannel_var cc(out_jack->connectTo(in_jack));
// use the first (best) transport strategy
cc->setTransportStrategy(cc->getStrategyList().front());

// try to get tcp control interface (stored in _var type)
ITCPControl_var itcp(cc->getInterface<ITCPControl>());
if(itcp.get()) {
  // set some parameter
  itcp->setSendBufferSize(2048);
}

// set XML value stream
cc->setValueStream(XMLIValueStream(), XMLOValueStream());
```

Figure 2. Explicit binding: setting and configuring the transport strategy and the serialization strategy.

Explicit binding allows for selecting the way in which the serialization and deserialization is performed. Also, the networking protocol or technology to be used for data transport can be chosen and configured (see Figure 2). This feature is especially important for controlling components in heterogeneous environments and for streaming multimedia data between these components. Here, specialized and optimized serialization strategies and transport strategies can be used to control devices like PDAs or set top boxes, which might only offer limited resource or proprietary networking technologies. We currently provide an XML serialization strategy and a more efficient strategy using "magic numbers" where type information is mapped to predefined numbers during serialization. Both representations can be directly transmitted over sockets using protocols like TCP or UDP. Another option is the combination of a serialization and transport strategy that uses the CORBA any-types for transmission. We currently use The Ace Orb (TAO) [17] which also provides real-time extensions.

The communication channels are also used to establish the multimedia data connection between different nodes in a flow graph. Here, special networking protocols suitable for multimedia data transport can be used, e.g. RTP [18].

Our architecture also allows to use a stack of serialization (and corresponding deserialization) strategies: in addition to an XML serialization strategy, a strategy for compressing the XML representation can be appended prior to transmission.

While establishing a communication channel, an application can also use a negotiation mechanism that automatically selects strategies for serialization and transport. If for example a communication channel to a locally operating node is established, the pointer forwarding strategy would be chosen, because it is the most efficient one. This strategy simply forwards the pointer to a message between two objects.

Although CORBA and other distributed object environments (DOE) also employ the proxy pattern, our framework only uses these middleware facilities as one possible solution among others for realizing access to remote objects. Our approach provides an open architecture that offers the transparent and flexible usage of different net-

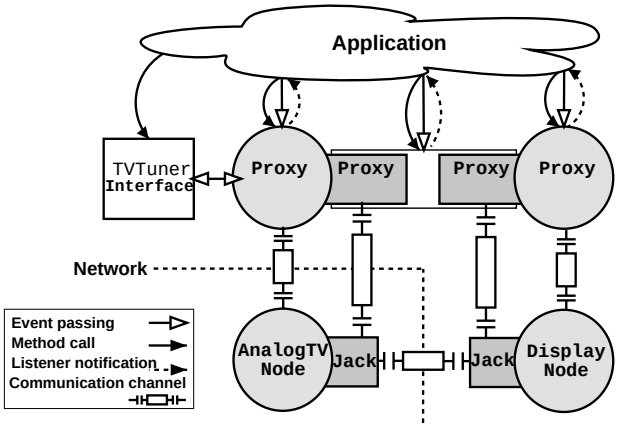


Figure 3. A remote node is controlled via an interface and connected to the local display.

working and middleware technologies [16]. This allows us to use different specialized solutions with low resource requirements, which is especially important in heterogeneous environments with mobile devices.

Figure 3 shows the overall structure of a distributed flow graph. Here, a remote node representing a PCI board for receiving digitized video from analog TV (AnalogTVNode) is connected to a local display (DisplayNode). These two nodes are controlled by the application via their proxy objects. Similarly, the jacks of the nodes are controlled through corresponding proxy jacks. The application can control the specific functionality of the source node with the corresponding interface (TVTunerInterface).

### 3.3 Distributed Registry and Migration

The registry service provided by NMM allows discovery, reservation, and instantiation of nodes available on local and remote hosts. In this step, the application sends a request to a local or remote registry. This request can include the specific type of a node (e.g. "AnalogTVNode"), the provided interfaces (e.g. "TVTuner"), or the supported formats (e.g. "video/raw"). Again, proxies and communication channels are used to contact the registry service. Nodes are then created by a factory either on the local or remote host. The registry also administrates resources like network bandwidth and computing power.

The NMM architecture also allows nodes connected in a flow graph to migrate to a different host during runtime (*session hand off*). In this step, the registry service of the new host is queried for the wanted node. Then upon instantiating the node, only the communication channels for control and data transmission have to be updated. The application still uses the same proxy object for controlling the object.

```
<configuration>
  <menu id="MainMenu" background="main.png"
        needvideo="no" needaudio="no">
    <entry index = "1"
      on = "dvd.on.png" off = "dvd.off.png"
      x  = "60" y = "200">DVDPlayer</entry>
    <entry index = "2"
      on = "mp3.on.png" off = "mp3.off.png"
      x  = "330" y = "200">MP3Player</entry>
  </menu>

  <mp3play id="MP3Player" background="mp3.png"
           needvideo="no" needaudio="yes">
  </mp3play>

  <dvdplay id="DVDPlayer" background="dvd.png"
           needvideo="yes" needaudio="yes">
  </dvdplay>
</configuration>
```

Figure 4. XML configuration with main menu including DVD and MP3 player.

## 4 Application Framework

Our overall system architecture consists of stationary and mobile systems. The stationary systems are called *Multimedia-Box*, and provide an integrated and extensible solution for a multimedia home entertainment center. Several of these systems at different location (e.g. different rooms) are connected over a network. The user can access multimedia data at the stationary systems or at a mobile device, e.g. a PDA, connected via WLAN. The user can move around with a mobile device, play back multimedia on the mobile device, or perform a session hand off to use the capabilities of nearby stationary devices for rich multimedia playback.

### 4.1 Multimedia-Box

The design goal for the Multimedia-Box was to create an open and extensible home entertainment platform on top of NMM that runs on a commodity Linux-PC with multimedia extension boards. This PC is controlled by a remote control and connected to a TV display and loudspeakers as in most living room scenarios and then acts as a replacement for traditional devices like VCRs, CD and DVD-players[1].

An important aspect in the design of the Multimedia-Box was easy configuration and extension of the application. In addition to easily changing the look of the application with so called *skins*, this requires the ability to adapt and extend the structure of the application itself. Therefore the whole application is built as a plug-in architecture that is assembled using an XML-based configuration language that describes the hierarchical menu structure the user can navigate in (see Figure 4). The leaves in this hierarchy are then the "actions" like DVD playback.

The overall system architecture of the Multimedia-Box application consists of several interacting parts (see Figure 5). From the XML-description the application is

[1] For videos and screen shots, see www.networkmultimedia.org/Status/MMBox/

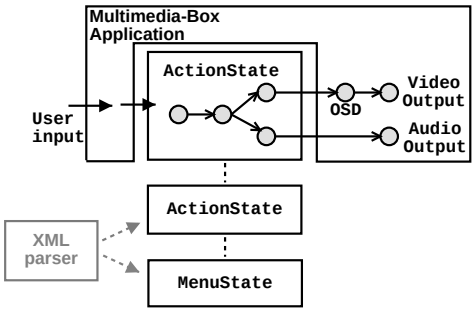


Figure 5. Architecture of the Multimedia-Box.

automatically set up: for each menu or action, a *MenuState* or an *ActionState* object is created, respectively. Events are generated according to user input with a remote control. These events are forwarded to the currently active state object. The state objects implement the *State* design pattern [19]. Depending on the received events the active state objects performs an internal state transitions (e.g. from "play" to "pause") or an external state transition by activating another state object. The sink nodes for audio and video output can be used by all states, but only the currently activated state is connected to them. Additionally a node for blending on-screen elements onto the current background (OSD) is inserted before the video sink node. Figure 5 shows the DVD state being activated and its simplified flow graph for DVD playback being connected to the common sink nodes. Notice, that the figure shows only the connected nodes; the proxies being controlled by the application are not shown for simplicity. Internally all states are realized as flow graphs of NMM nodes like the one in Figure 3.

Our current implementation provides ActionState objects for audio CD playback, MP3 encoding of an audio CD, MP3 playback including navigation in the directory structure, DVD playback, transcoding of multimedia data like DVDs to other formats, TV access with time-shifting, programming of video recordings, and a browser and media player for all supported media types, including MPEG4, Ogg/Vorbis, and many other. For receiving TV programs two options are supported. A DVB-board directly receives MPEG-2 encoded digital TV, while for analog TV we use a combination of a TV-tuner board and a cheap KFIR MPEG-encoder board.

With the application framework, integrating a new action, for instance a video conferencing application, is straightforward. All that needs to be done is implementing a new ActionState object identified by a unique string and adding a new menu entry in the XML configuration. Also, special versions of the Multimedia-Box that are not connected to a TV but only provide audio output can be configured easily by modifying the menu structure in the XML configuration file.

In addition to being able to navigate the hierarchy of states, our application framework also supports multitasking. The user can put a running state to background or reactivate a state running in background. States running in the background get a lower priority assigned. In addition, if the currently running state is to be replaced with a new state, the application framework disconnects the running state from the audio and video output only if the new state is configured to need one or both of these outputs. E.g. the DVD state will require both audio and video while the MP3 state only needs the audio output and a menu needs neither audio nor video. This behavior is also to be set in the XML configuration (see *needaudio* and *needvideo* in Figure 4).

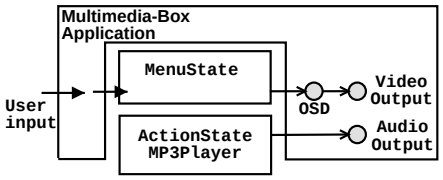


Figure 6. The MP3 player is put to background by a menu state but keeps on playing music.

## 4.2 Networked Home Entertainment

As all ActionStates are realized as flow graphs of NMM nodes, the distribution of workload to remote hosts can also be performed very easy. This is especially useful for resource consuming tasks like transcoding of audio and video. The nodes performing these tasks are requested from a remote registry while the proxies to control these nodes still reside within the local Multimedia-Box application.

Furthermore, the Multimedia-Box can transparently use distributed devices: if the local host does not provide the possibility to receive TV, the application searches for the required NMM node (a node that provides the "TV-Tuner" interface) on remote hosts and then integrates them into a flow graph as described in Section 3.

## 4.3 Multimedia Access for Mobile Users

Our middleware can also be used on mobile devices like the iPAQ PDA running under Linux and using WLAN. Also, the application framework running the PDA is similar to the one running on the Multimedia-Box. The user interface offers the same appearance and behavior as on the stationary system. We currently provide a state for playback of audio files (e.g. MP3) on the mobile device.

The user can either play back audio on the mobile device or perform a session hand off to a remote host in the proximity of the user to enjoy better audio output than provided by the mobile device. If the audio is played at the mobile device all nodes are instantiated locally.

Figure 7 shows the setup when audio is decoded on a remote stationary host running the Multimedia-Box application. Here, the application running on the mobile host

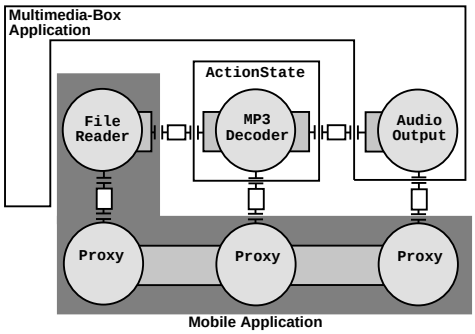


Figure 7. Cooperation of a mobile and a stationary system.

(shaded dark) uses a locally running node for reading a file from the internal PDA memory. This node is controlled with the corresponding proxy. To perform the decoding and playback of the audio on the remote system, the application has requested an appropriate audio decoder node from the remote registry. Notice, that in this step the application running on the mobile host uses all facilities of the underlying NMM middleware, e.g. the dynamic migration during runtime as described in section 3.

A special state within the Multimedia-Box application then gets connected like any other state, the only difference is that its instantiated nodes are controlled from the mobile device. Additionally, the application on the mobile device can request a proxy for the remote audio output, e.g. for controlling the volume. To this end the mobile device can be used as a remote control for stationary Multimedia-Boxes. All nodes are then connected and appropriate communication channels are established, e.g. using WLAN for connecting from the mobile device. With our current implementation, the complete setup and hand off requires about 2 seconds.

## 5 Conclusions

We have described an home entertainment platform based on a multimedia middleware that provides transparent control and cooperation of distributed components with different underlying technologies. The application framework consists of stationary and mobile systems. The stationary hosts act as an integrated and extensible solution for various kinds of multimedia entertainment. In addition users can access multimedia data on mobile devices or transparently use remote resources like computing power and I/O capabilities. Future work will concentrate on the provision of session hand off for distributed rendering of audio and video on mobile and stationary systems.

## References

[1] Microsoft. DirectShow. http://msdn.microsoft.com/.

[2] Apple. Quicktime. http://www.apple.com/quicktime/.

[3] Sun. Java Media Framework. http://java.sun.com/products/java-media/jmf/.

[4] Object Management Group. http://www.omg.org.

[5] Hewlett-Packard Company and IBM Corporation and SunSoft Inc. Multimedia System Services.

[6] David Duke and IIvan Herman. A Standard for Multimedia Middleware. In *ACM International Conference on Multimedia*, 1998.

[7] Ken Arnold and Bryan O'Sullivan and Robert W. Scheifler and Jim Waldo and Ann Wollrath. *The Jini Specification*. Addison-Wesley, 1999.

[8] The HAVi Specification - Specification of the Home Audio/Video Interoperability Architecture. http://www.havi.org.

[9] Klaus Ilgner and John Cosmas. System Concept for Interactive Broadcasting Consumer Terminals. In *International Broadcasting Convention*, 2001.

[10] Bostjan Marusic and Marijan Leban. The myTV System - A Digital Interactive Television Platform Implementation. In *IEEE International Conference on Multimedia and Expo (ICME)*, 2002.

[11] Multimedia Home Platform (MHP). http://www.mhp.org.

[12] Heribert Baldus, Markus Baumeister, Huib Eggenhuissen, Andras Montvay, and Wim Stut. WWICE - An Architecture for In-Home Digital Networks. In *Multimedia Computing and Networking (MMCN)*, 2000.

[13] Heinz-Josef Eikerling and Frank Berger. Design of OSGi Compatible Middleware Components for Mobile Multimedia Applications. In *Protocols and Systems for Interactive Distributed Multimedia Systems (IDMS/PROMS)*, 2002.

[14] Open Services Gateway Initiative (OSGi). http://www.osgi.org.

[15] George Coulouris, Hani Naguib, and Scott Mitchell. Middleware Support for Context-Aware Multimedia Applications. In *Conference on Distributed Applications and Interoperable Systems (DAIS)*, 2001.

[16] Marco Lohse, Michael Repplinger, and Philipp Slusallek. An Open Middleware Architecture for Network-Integrated Multimedia. In *Protocols and Systems for Interactive Distributed Multimedia Systems (IDMS/PROMS)*, 2002.

[17] Douglas C. Schmidt et al. The Ace Orb(TAO).

[18] RTP. http://www.cs.columbia.edu/∼hgs/rtp/.

[19] Erich Gamma, Richard Helm, and Ralph Johnson. *Design Patterns*. Addison-Wesley, 1995.