Towards Ubiquitous Distributed Multimedia: UPnP-Based Control and Scripting

Milan Z. Bjelica, Bojan Mrazovac, Nikola Teslić, Member, IEEE

Abstract—One of the most important enablers when assembling a modern ubiquitous computing system is distribution of multimedia content throughout a home and control of its playback. Additionally, ambient intelligence requirements include playback timeline programming, interaction with sensors and event-based playback triggering. Normally there is a central server dedicated to content storage and streaming to one or more spatially displaced viewing screens. In this paper we present a newly developed portable software library that provides mechanisms to control multimedia distribution and playback from one or more content servers, to multiple screens. The control is based on UPnP/DLNA protocol stack, therefore supporting integration of many off-the-shelf UPnP/DLNA compatible multimedia servers or playback devices. The primary goal is to interpret custom scripts written to achieve desired multimedia ambient. These scripts provide control over content distribution and playback based on time triggers and external, arbitrary events, such as sensor readings and user commands. The library provides faster response to a control command in comparison to traditional UPnP control points. Content providers and their physical addresses are hidden from users enforcing the concept of a media “cloud” to fetch the content from. Content can also be distributed to multiple targets at once, whereas it retains control synchronization.

Index Terms—ambient intelligence, control point, distributed multimedia, DLNA, UPnP

I. INTRODUCTION

UBIQUITOUS computing as a concept that has been rising in popularity since Weiser’s definition back in the 1990’s demands integration of different computer-based technologies into everyday human life. It is essential for this integration to maintain invisibility of technology to humans [1]. Multimedia distribution and playback are one of the essential building blocks of such systems. Key requirements are the following: (1) separation between content providers and content players; (2) synchronized content playback on multiple players; (3) centralized playback control; (4) support for integration of different widely available multimedia devices.

To achieve previous requirements UPnP / DLNA (Universal Plug and Play / Digital Network Alliance) protocol stack is most frequently used, together with the palette of compatible media devices [2][3]. This protocol defines three vital entities: media server, media player (renderer) and control point (that controls playback and content distribution). Nevertheless, most commercially available home media systems bundle playback and control functions together [4][5][6]. Users can list all the available media servers, traverse their content until the desired media item is found and then play it. Basic functions that are supported are mainly: play, pause, resume and stop. Also, current playback state and changes in playback state are broadcasted by media players as media events. The main drawback of such solutions is their unsuitability for use in ambient intelligence environments, primarily because they lack centralized media playback control as well as the ability to program the playback timeline.

Commercial solutions that comply with the previous requests are scarce. Currently the most popular implementation of a control point is Cidero [7]. This software application enables listing all servers and players that are attached to the local network. A graphical user interface is provided to browse content on servers, choose a specific multimedia file and start/stop its playback on an arbitrary player. However, it is difficult to integrate Cidero to an ambient intelligence system. This application provides neither synchronized playback control on multiple players, nor playback timeline programming. Response time to a control command (~2 seconds) is not viable for applications with harsh media synchronization requirements. Nevertheless, Cidero was useful to authors as a reference point in development and testing, given its usage simplicity.

Numerous authors have presented results of their researches regarding home multimedia, playback synchronization and content distribution. Initial predictions in this area as well as possible media synchronization issues were given by Ramanathan and Rangan in 1992. They introduced a concept of media server that can serve both content and playlists to players (mediaphones) [8]. Latest researches often sketch various middleware for distributed multimedia systems. Zhang et al. proposed a solution for synchronized multimedia presentation across heterogeneous applications, called NetMedia [9]. Sung et al. presented their framework for multimedia playback control based on UPnP [10]. Specific
architecture of a distributed multimedia system based on RTP (Real Time Transport Protocol) and dedicated proprietary players (digital audio/video station – DAVS) were presented by Young et al [11]. To the best of our knowledge, however, current research stream fails to regard multimedia distribution and playback timeline programming as a whole.

In the rest of this paper we shall present the basics of the developed multimedia playback control library in a distributed system based on UPnP/DLNA, to which we refer as AVSXLib. Using AVSXLib it is possible to control content distribution and playback from one or more media servers, to several players. Key differentiators of our solution compared to the others are: (1) faster response time to a control command (that is traditionally degraded because of e.g. video decoder initialization); (2) simplified content paths addressing (it is not necessary to provide complete URI address to the target content, but only its name – it will be fetched from the “cloud”); (3) simultaneous playback control across several players and (4) execution of XML-based scripts that define the behavior within the multimedia system (playback timeline programming, sequential or shuffled playback, playback driven by external events). Following we shall present concepts and some implementation details for AVSXLib library. Finally, we shall present a use case with a simple user application utilizing AVSXLib to control multimedia. We shall also give test results, response time comparison to Cidero and control synchronization precision assessment.

II. AVSXLIB LIBRARY BASICS

AVSXLib library is based on UPnP/DLNA A/V architecture. This architecture enables sharing multimedia content between media servers and media players (A/V devices), regardless their hardware, multimedia content formats or data streaming protocols used. UPnP defines communication primitives (command and event formats) between A/V control points and A/V devices. According to the DLNA architecture, media server provides both content acquisition and media library traversing. The server is also in charge of content cataloguing, communication with control point and A/V devices, and final data transport (AVTransport). A/V device (renderer) is a network device that can decode acquired content and present it to a local screen. Once transport protocol and media formats are agreed with the server, renderer receives the content using a dedicated, out-of-band FTP/HTTP protocol. UPnP/DLNA architecture is presented in Figure 1.

To send UPnP commands and exchange information on devices and services SOAP (Simple Object Access Protocol) [12] is used. All exchanged data is presented as XML [13]. UPnP includes a mechanism to discover compatible devices in the network based on SSDP (Simple Service Discovery Protocol) [14], by having all devices broadcast their name and their address (advertisement). Renderers also broadcast changes in current playback state (playing, stopped, paused, resumed, transitioning), for what they use GENA (General Event Notification Architecture) [15] primitives.

AVSXLib uses libupnp library [16] for UPnP protocol stack implementation. Libupnp is written in C/C++ and portable to different operating systems, including Windows and Linux.

III. AVSXLIB IMPLEMENTATION DETAILS

AVSXLib library consists of the following basic software blocks: (1) UPnP library (libupnp), (2) DLNA support modules, (3) control middleware, (4) XML script interpreter and (5) library API (application programming interface). Support for both Windows and Linux is achieved by using POSIX (Portable Operating System Interface) primitives [17] and pthread library [18], which minimized the need for double coding. AVSXLib software architecture is given in Figure 2.

Several basic UPnP library (libupnp) primitives were used in implementation. After initialization (UpnpInit) a client is registered (UpnpRegisterClient) with a callback function to be called each time a new event is received from the network (advertisement, playing, stopped etc). Using the same callback libupnp interfaces upper software layers (DLNA). For instance, once libupnp is instructed to discover all compatible devices in the network (UpnpSearchAsync), the callback is called each time new device is recognized. Now, DLNA module can link newly discovered device to servers or renderers list, together with the acquired details in XML format (UpnpDownloadXMLDoc). Further, actual control is performed by sending a UPnP command (action) to target device address (UpnpAddToAction, UpnpSendAction).

DLNA modules keep track of all existing servers and renderers in the network, as well as of their current states. Server catalogue traversing mechanism is implemented, as well as mapping media file names to media URLs, which are required for issuing a UPnP playback command. This
facilitates search for the desired media file and its playback on a target device only via name. DLNA modules also include functions for parsing tags in XML device descriptions.

Control middleware provides all necessary distributed multimedia system functions: (1) start control block (DmediaInit); (2) stop control block (DmediaExit); (3) search for servers (DmediaServerDiscovery); (4) search for renderers (DmediaRendererDiscovery); (5) choose a media server catalogue (DmediaSetCurrentDir); (6) start/resume media playback on target device (DmediaPlay); (7) pause media playback on target device (DmediaPause) and (8) stop media playback on target device (DmediaStop). Two additional functions are provided: (1) start synchronized playback (DmediaPlaySync) and (2) load server media “cloud” configuration (DmediaLoadServerCloud).

To achieve playback control synchronization (best effort principle), DmediaPlaySync has the following parameters: (1) time when the playback should start measured from the moment the function was called, (2) list of all devices that should start the playback, and (3) name of the media file to be played. After the call library calculates target time moment: 
\[ t_{\text{target}} = t_e + \Delta t_e - \Delta t_p, \]
where \( t_e \) being the time when the function is called, \( \Delta t_e \) the time given as a parameter and \( \Delta t_p \) lead time needed to start playback on a target device. \( \Delta t_p \) is initially set to \( \Delta t_p=10\text{sec} \) and is reevaluated for every target device after each call to DmediaPlay. The library measures elapsed time from the moment when UPnP command is issued, to the reception of playing event, and updates \( \Delta t_p \) to maximum time measured. Now a background thread waits for absolute time \( t_{\text{target}} \), when DmediaPlay is called followed by an immediate DmediaPause. When the moment \( t_{\text{target}} + \Delta t_p \) is reached, DmediaPlay is called one more time. This technique minimizes initial delay by using resume primitive, which continues an ongoing playback without waiting for stream reception, decoder initialization etc.

The library provides mechanism to define media “cloud” by defining all paths to server catalogues in a configuration file loaded by DmediaLoadServerCloud call. Once the cloud has been set up, all media file names referenced in control commands are translated to their respective URLs, without the need to previously set catalogue using DmediaSetCurrentDir.

Essential module and the greatest contribution of AVSXLib library is its script interpreter. Based on our previous works [19], the goal of the interpreter is to enable playback timeline programming through XML-like script language. This language is similar to high-level programming languages (supports declaration and usage of variables, loops, if-then-else constructs, delays with sleep instruction). Additionally, it is extended with commands to control playback, which are mapped to the API of the control middleware. Script execution can also depend on environment variables, that can be set from outside script by using a specific API function. This way it is possible to integrate AVSXLib to an ambient intelligence system. For example, environment variables can be set by sensor devices (such as cameras, e.g. motion detection) and to trigger playback when some desired value is recognized. Continued we give an example of simple script that controls playback on three different screens in a room, based on current value of environment variable @profile.

```xml
<use entity="server_roots"/>
<use entity="reload_on_event" param1="@profile"/>
</if>
</case>
<case name="@profile" val="relax" rel="=">
  <if>
    <case name="@profile" val="study" rel="=">
      <default><stop renderer="screen_A"/></default>
      <default><stop renderer="screen_B"/></default>
      <default><stop renderer="screen_C"/></default>
    </case>
  </if>
</case>
<case name="@profile" val="study" rel="=">
  <for><play media="party.avi" renderer="screen_C"/></for>
  <for><play media="sun.jpg" renderer="screen_A"/></for>
  <for><play media="beer.jpg" renderer="screen_B"/></for>
  <sleep time="3000"/>
  <play media="beach.jpg" renderer="screen_A"/>
  <play media="sea.jpg" renderer="screen_B"/>
</case>
<default><stop renderer="screen_A"/>
  <stop renderer="screen_B"/>
  <stop renderer="screen_C"/></default>
</if>
```

Toplevel module contains AVSXLib API, comprising of all previously enumerated functions of the control middleware, together with the following functions: (1) start AVSXLib library (AVSXLibInit); (2) stop AVSXLib library (AVSXLibExit); (3) start script playback (AVSXLibStartScript); (4) stop script playback (AVSXLibStopScript); (5) set environment variable (AVSXLibEnvVarUpdate). User application links AVSXLib library, accessing its functions by including avsx.h file.

IV. CASE STUDY

To demonstrate usage of AVSXLib library, we developed a simple console user application for Windows using Visual Studio 2005. Upon startup, user enters the name of XML script file and starts its playback. Subsequently user can alter environment variable values by pressing keys on keyboard. Pressing q at any time stops script execution and allows starting new scripts.

Described application was run on a PC connected to Ethernet network with 3 additional PCs where each PC was running UPnP/DLNA compatible software. SimpleCenter [20] used as media server was installed on a single PC, whereas Twonky Media Renderer [6] was installed on all PCs and used to show media on PC screens. The setup is shown in Figure 3.
For demonstration purposes we wrote a script that synchronously issues play command to all three renderers, playing the sequence of five video files fetched from the media server. Based on @profile environment variable, sequences are swapped and another set of five video files is played. Profile is switched manually by pressing keys A or B on the keyboard. User testing proved correct operation for the given script.

Finally, a comparison of response times to play command was conducted between AVSXLib and Cidero. Measuring started from the moment of issuing play command, to the moment playback appeared on renderer screen. Video camera was used to capture both user and renderer screens. The recording was then analysed in VirtualDub [21], frame by frame, to get two key timestamps: the moment when play command was printed to console screen, and the moment when playback started to be visible on a renderer screen. The time was measured between the moment playback appears on any of the renderer screens, to the moment playback appears on all screens. Results for 20 consecutive play commands during script execution are noted and presented in Figures 4 and 5.

Obtained results confirm decreased response time and better control synchronization performance when AVSXLib library is used. All measurements are performed under the assumption that play command is issued with an intentional 20sec delay.

V. CONCLUSION

In this paper we presented a UPnP/DLNA-based control unit for distributed multimedia systems, in the form of a novel AVSXLib library. Some of the most exclusive perks with AVSXLib are the existence of scripting driven by external events as well as support for off-the-shelf media devices. This facilitates library usage in ambient intelligence systems. Authors give an extended analysis of AVSXLib application in ambient intelligence in [22]. In this work, we used AVSXLib library inside an intelligent pinboard system at the university. Based on people presence detection in a certain area provided by a single 3D camera sensor, appropriate media presentation playback was triggered.

Synchronization and response time problems are tackled in this paper. However, solutions can be further improved. It was also noted that many off-the-shelf UPnP/DLNA renderers are not fully compatible in terms of external controllability. One of the important tasks for future work is to test the library with as many different UPnP/DLNA servers/renders as possible.

REFERENCES

[16] Portable SDK for UPnP devices (libupnp 1.6.6), http://upnp.sourceforge.net/
[22] B. Mrazovac, M. Z. Bjelica, I. Papp, V. Kovacevic, „A system for distributed video playback based on 3D localization of the user“, in proceedings of conference ETRAN, 2010 (to be published)