Making an Android Tablet Work as a Set-Top Box

Lorenz Klopfenstein Saverio Delpriori Emanuele Lattanzi Alessandro Bogliolo
Information Science and Technology Division of DiSBeF
University of Urbino
Urbino, Italy 61029
alessandro.bogliolo@uniurb.it

Gioele Luchetti
NeuNet
www.neunet.it
Urbino, Italy 61029
gioele.luchetti@gmail.com

***Abstract*—The widespread diffusion of connected devices capable of receiving and decoding multimedia streams is inducing a change in the market of set-top boxes from dedicated proprietary appliances to software modules running on top of off-the-shelf devices. In spite of the large number of devices we use every day, smart phones are the favourite answer to our communication needs because of their availability, of their user friendliness, and of the great opportunities of personalization offered by user-generated mobile applications. The last generation of tablet PCs, capable of handling HD multimedia streams while also retaining the distinguishing features of mobile devices, enable the convergence between personal communication devices and home entertainment appliances. In this paper we discuss how to use an Android tablet PC as a set-top box, in order to allow end-users to take advantage of the tailored run-time environment of their personal mobile devices while whatching television in the comfort of their living rooms.**

***Keywords*-Set-top box; Tablet PC; openBOXware; Android; Streaming**

I. INTRODUCTION

IP traffic trends and forecasts [1], [2] indicate that multimedia contents delivered over residential and mobile IP networks are among the main driving forces of next generation networks.

The analog switch-off and the advent of *digital video broadcasting* (DVB) have enabled the technological convergence of client-side equipment required to take advantage of broadcast TV channels, IPTV services, and Internet multimedia streams. Nowadays, all new television sets come with embedded decoders, and most of them are Internet enabled. In this scenario, software components running on top of off-the-shelf connected devices are replacing proprietary *set-top boxes* (STBs), while traditional IPTV models are undergoing deep changes in order to face the pressure of *over-the-top* (OTT) multimedia contents streamed across global *content delivery networks* (CDNs).

At the same time, the widespread diffusion of smart phones and Internet enabled mobile devices, together with the growing coverage of broadband wireless networks, have induced operators to move from *triple-play* offers (i.e., Internet access, VoIP, and IPTV) to *quadruple-play* offers (which includes wireless connectivity) [3], accelerating the convergence between mobile and residential broadband markets and creating the conditions for delivering mobile TV services [4].

In spite of the wide diversity of connected devices which might work as multimedia boxes (including connected TV sets, media centers, DVB decoders, video game consoles, and personal computers), end-users spend most of their connected time using personal smart phones (or similar handheld devices) which have several competitive advantages: they are available everywhere and at any time, they offer intuitive user interfaces, they provide suitable answers to any communication need, and they provide unprecedented opportunities of personalization thanks to the thriving market of user-generated contents and applications [5].

If, on one hand, exploiting addins and configuration options to create a perfectly tailored run time environment on a smart phone is an intriguing pastime, both the actual quality of experience offered by the device and the effort devoted to personalize it keep end-users from using other devices.

Although a new generation of STBs has recently sprouted which allow end-users to create their own applications and to easily install third-party addins [6], they are far away from gaining the popularity of their mobile counterparts and the gap is hard to be closed in the near future. In fact, mobile devices are always at users' disposal and they will maintain their dominant role of personal communication equipment. Moreover, STBs are typically installed in a living room where they are mainly expected to provide a *lean-back* usage experience, which is in contrast with the *lean-forward* attitude typical of smart phone users, which has sustained the market of mobile applications [7], [8].

On the other hand, personal handheld devices have never threaten the market of media centers and STBs because of their tight design contraints, imposed by portability requirements, which made them unsuitable to sustain the workload of high definition multimedia streams. The gap between personal mobile devices and multimedia boxes is about to be closed, however, by the last generation of tablet PCs, which support HD video streams and are equipped with HDMI interfaces.

This paper investigates the possibility of making an *An-*

 ISBN: 978-1-61208-148-9

droid tablet PC work as a STB, thus allowing end-users to take advantage of their personal runtime environment in the comfort of their living room, possibly switching from a lean-forward to a lean-back usage experience. The starting point is *openBOXware* (http://www.openboxware.net/trac/), an open-source framework for the development of bandwidth-aware multimedia applications originally implemented on top of *Mono*, using *GStreamer* for the multimedia subsystem and *Qt* for the graphic user interface. The concept, the main features, and the architecture of openBOXware are outlined in Section II, while Section III shows how to port the key features of openBOXware on top of Android, also discussing the switching between the lean-forward interface typical of an Android tablet PC and the lean-back interface of a STB.

II. OPENBOXWARE

OpenBOXware is on open-source framework which works as a general handler of multimedia flows streamed from heterogeneous sources to both local and remote targets. The framework automatically creates the streaming pipelines between media sources and media targets, possibly including the required transcoding stages. The capability of handling multiple simultaneous pipelines while streaming them to remote targets makes it suitable to be installed not only at the receiving end point, but also at the intermediate nodes of a content delivery network.

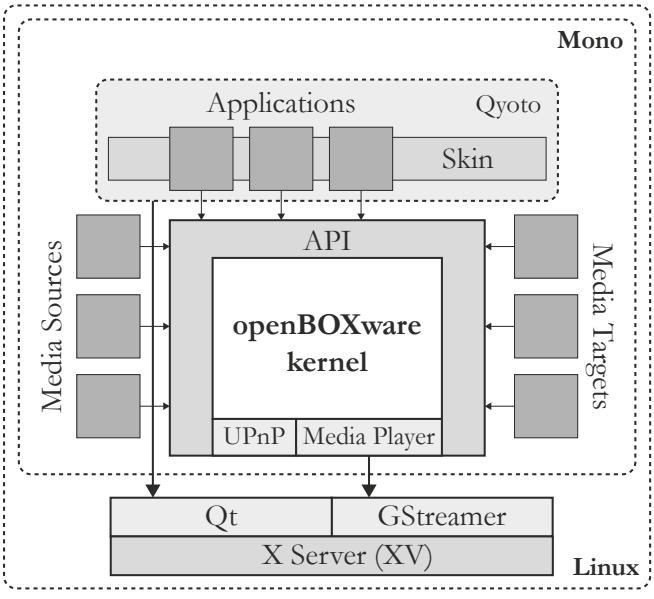

Figure 1. The software architecture of openBOXware.

In particular, openBOXware provides support for incoming and outgoing multicast streams, thus enabling the implementation of bandwidth-aware content distribution mechanisms within managed IP networks [9], [10].

OpenBOXware has a layered architecture, as shown in figure 1, which grants portability by abstracting the underlying HW platform and software components. The first level, which is the *kernel*, includes the implementation of all actual components and makes them interact with each other. It also handles bootstrapping and loading.

The second level, which is composed of the application programming interfaces (API), provides the abstraction to the functionalities exposed by the kernel. This interface level also provides means for external components, built by third-party developers and deployed as *add-ins*, to create custom applications that run on top of openBOXware and to describe media sources and media targets in an abstract fashion.

Those add-in components represent the highest architectural level, which includes all other software directly relying on the API. In particular, a special component, called *skin*, acts as application manager and determines the main user interface of the platform. All other applications, including *media sources* and *targets*, can expose new functionalities to the system. By changing the skin it is possible to change the usage experience while maintaining compatibility with the applications. Media sources and media targets, implemented as high-level add-ins, expose external resources to the system and, thus, to any other component willing to use them.

A media source represents an abstract browsable tree of media elements, each one described in such a way to enable a media target to open it and start playing it back. Given the abstract descriptions of the source media element and of the media target of choice, the framework handles multimedia loading, streaming, transcoding (if needed) and delivery to the target. For instance, a media source could describe an online video service, whose videos can be streamed to a specific media target (for instance, a remote TV set). This is done either by using a general application, called *media library*, which allows the end-user to browse media sources and easily bind them to a media target, or by using specific applications which create ad-hoc pipelines between predefined sources and targets, exhibiting a brand-specific user interface.

Other possible add-ins include client-side web applications, ranging from social network clients and feed readers to fully fledged web browsers, and server-side applications, such as *UPnP* services for home entertainment/automation or any kind of web services. Moreover, any application can expose a remote interface, possibly built by taking advantage of the embedded web server. Each application can run in one or more *execution modes*, including *fullscreen* (which takes over the whole screen area, covering up other applications), *sidebar* (which share the foreground with the top-level fullscreen application), and *background* (for services that do not need any graphic user interaction). The execution modes implemented by a given application are declared in its *manifest*, which is an XML file providing to the framework the information required to present it to the end-user and to load its components when required.

 ISBN: 978-1-61208-148-9

OpenBOXware is currently implemented on top of *Mono*, an open source implementation of the Microsoft .NET framework, providing the required abstraction from the underlying hardware and software components. The multimedia subsystem relies on *GStreamer*, while the graphic user interface is based on *Qyoto*, a binding library of the *Qt* framework for .NET. Video output is enabled through the XV overlay mechanism of *X server*.

The current 1.2 software release is freely available from the official website of the openBOXware project, including its source code (http://www.openboxware.net/trac/). This first release has been developed and tested on x86 PCs running the GNU/Linux operating system, while work on a port to an ARM based embedded platform (namely the IGEPv2 board equipped with a TI OMAP processor) is currently under way, specifically targeting the MeeGo operating system [11], whose core distribution includes all required components and exists in both x86 and ARM flavors.

III. PORTING OPENBOXWARE ON ANDROID

The Android architecture is built on top of Linux kernel and consists of three main layers: the *Android runtime*, based on the *Dalvik* virtual machine (VM) with additional *support libraries*, the *application framework*, and the *applications* which run on it [12]. An Android application can be made of several components. For our purposes, the most important types of components are *activities*, which represent screens with specific user interfaces, and *services*, which run in background. Each application runs in a separate VM instance for security and protection. Communication among applications is guaranteed by an asynchronous message passing mechanism which allows a component to issue an *intent* message which is handled by another component possibly belonging to a different application. Each intent contains action and data specifications which are used by the application framework to dispatch the intent and trigger one of the components registered for performing the requested action on the specific type of data. The main graphic user interface is provided by a *launcher*, which is a special activity registered to react to a particular intent issued by the operating system at start up. The launcher allows the end-user to browse and launch activities which publish the `MAIN` intent filter. In addition, the launcher can also act as a *widget host* to allow end-users to customize the main page by embedding their preferred miniature applications.

The porting of openBOXware on top of Android, schematically represented in Figure 2, can take advantage of the features of the Android application framework in order to avoid re-implementing the bootstrapping and component handling functionalities of the kernel. The task of handling, installing, and loading component packages can be easily left to the default package manager provided by Android, ensuring that all required openBOXware components are correctly mapped to standard Android activities and services.

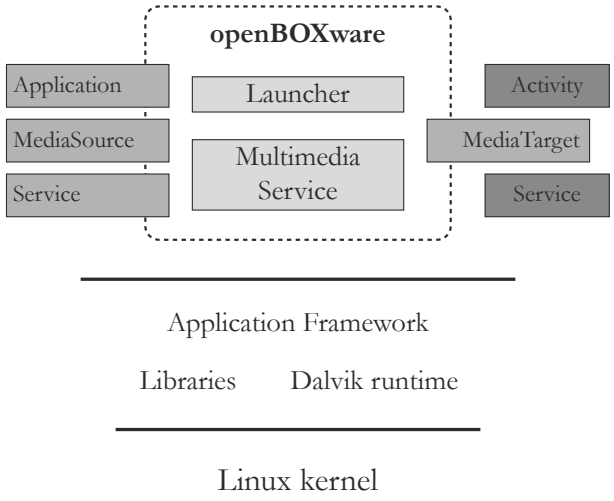

Figure 2. The openBOXware ecosystem running on top of Android.

In order to provide the usage experience typical of a STB system, openBOXware on Android sports a custom launcher conceived to offer an easy to use lean-back experience and to discriminate between normal Android applications and special openBOXware applications (identified by the intents they are registered to handle, as detailed below). It is worth noticing that multiple launchers can be installed on the same device, but only one at the time can be running. Hence, a home switching mechanism is required to allow the user to switch from a lean-forward to a lean-back use of his/her own Android device. There are three main ways to achieve this switching functionality: by changing the default launcher in the Android settings, by avoiding to specify a default launcher (in this case the choice is made by the end-user on a dialog box which appears whenever he/she presses the home button on the device), or by means of a specific *home-switching* application. Once the openBOXware launcher is selected, it becomes the main graphical user interface of the system, which shows a status bar, allows the user to launch applications, and displays the installed media sources and targets. In practice, the launcher essentially takes over the role of the skin in openBOXware, providing three different home screens: *i)* the media library, *ii)* the openBOXware application grid, and *iii)* the Android application grid. By default, the launcher presents the media library home screen with the previews of the available media sources that can be easily selected and played back on the default media target (which is the built in media player) as conventional TV channels. Advanced functionalities provided by add-ins can always be accessed from the other home screens.

Applications which provide a graphical interface are to be implemented as activities. By default, activities that desire to be listed and started by the launcher must handle the `android.intent.action.MAIN` intent, which tells them to start up and present their user interface. This mech-

 ISBN: 978-1-61208-148-9

anism is extended in openBOXware by adding one custom intent, `openboxware.app.FULLSCREEN`, which allows the launcher to recognize openBOXware applications, providing a dedicated STB-like viewing experience, and to list them into the specific home screen.

On openBOXware, applications could be run both in fullscreen and sidebar mode. On the Android port however, only the fullscreen mode has been retained. The decision to drop the sidebar mode for applications is due to some limitations of the Android window manager, which does not correctly support interactions with windows other than the one in foreground. Displaying a sidebar application is technically feasible, but it could provide an inconsistent interaction with the background fullscreen application, if any. Thus, instead of allowing applications to run in sidebar, a separated system sidebar has been implemented, which is displayed in overlay and acts as a widget host, containing any number of widget applications according to users' choices. This solution allows openBOXware to take advantage of the existing widgets in a manner which fits into the Android architecture.

Component mapping

The main components of the openBOXware API are mapped on Android as follows.

Notifications: openBOXware offers a simple API to enqueue text notifications and display them on the system status bar, allowing the user to react to the corresponding events. This API may be used by applications running in background to ask for user interaction. The notification hub is directly implemented on top of the Android notification manager and provides an additional user interface which makes it compatible with the STB-like experience.

Persistence: The original framework includes a simple associative key/value map that is persisted to disk, in isolated storage for each installed application. On Android, applications can make use of the system *SQLite* database storage to this purpose.

Networking: Web server functionalities and UPnP browsing and service consumption rely on external libraries which are readily available for Java on the Android operating system.

Multimedia playback: OpenBOXware 1.0 exposes an API that wraps the GStreamer multimedia framework. Similarly, a thin wrapper around the built-in Android media player can be provided to developers as a public API. However, playback capabilities of the system multimedia backend are heavily constrained due to the nature of the platform. Hence, many functionalities which are easy to provide using GStreamer must be emulated, reimplemented, or dropped altogether (e.g. UDP/RTP output streaming) on top of Android.

Media sources: Add-ins that export a media source to the system are implemented as Android services, which react to the `openboxware.mediasource.ACCESS_MEDIA` intents. This intent allows other applications to bind to the service and start a bidirectional RPC communication session, which allows the application to browse the media tree and the service to return structured data bundles representing media elements. These data bundles encapsulate all data required to play back the media resource and they can be consumed directly by the media player API.

Media targets: Since additional media targets cannot rely on GStreamer, their whole streaming backend must be implemented from the ground up. Thus, media targets are implemented as full-fledged media player instances that can be used instead of the default one provided by the system.

Dæmons: Applications that do not provide a graphical interface are naturally implemented as Android services.

IV. Conclusions

In summary, openBOXware can be ported on Android as an ecosystem which encloses a multimedia subsystem, a custom launcher, a set of specific intents and a development framework for applications that want to be recognized as openBOXware add-ins.

As a last remark, the default multitouch input device of Android has to be complemented by a remote control in order to provide full support to a lean-back living room experience. The bluetooth interface makes it possible to use off-the-shelf input devices to this purpose. More advanced control functionalities can be achieved by using an Android smart phone as remote.

The proposed architecture is currently under development on Android 2.2 running on two different devices: a *Samsung Galaxy Tab* with dock station, representative of state-of-the-art tablet devices, and an *IGEPv2 board*, representative of open-source embedded hardware platform. In both cases, a *Logitech diNovo Mini* is used as bluetooth remote and keyboard.

Acknowledgment

The research leading to these results has received funding from the EU IST Seventh Framework Programme ([FP7/2007-2013]) under grant agreement n 25741, project ULOOP (User-centric Wireless Local Loop), and from the Italian ICT4University Programme, project U4U (University for University).

References

[1] Akamai, "Q3 2010 - The State of the Internet," *Akamai report*, 2011.

[2] Cisco, "Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2010-2015," *Cisco White Paper*, 2011.

[3] K. Mikkonen, "Exploring the creation of systemic value for the customer in advanced multi-play," *Telecommunications Policy*, vol. 35, no. 2, pp. 185 – 201, 2011.

 ISBN: 978-1-61208-148-9

[4] L. Zhou, A. V. Vasilakos, L. T. Yang, and N. Xiong, "Multimedia Communications over Next Generation Wireless Networks," *EURASIP Journal on Wireless Communications and Networking*, 2010.

[5] A. Holzer and J. Ondrus, "Mobile Application Market: A Mobile Network Operators' Perspective," in *Exploring the Grand Challenges for Next Generation E-Business*, ser. Lecture Notes in Business Information Processing, W. Aalst *et al.*, Eds. Springer Berlin Heidelberg, 2011, vol. 52, pp. 186–191.

[6] C. Maturana, A. Fernndez-Garca, and L. Iribarne, "An implementation of a trading service for building open and interoperable dt component applications," in *Trends in Practical Applications of Agents and Multiagent Systems*, ser. Advances in Intelligent and Soft Computing, J. Corchado *et al.*, Eds., 2011, vol. 90, pp. 127–135.

[7] D. Gavalas and D. Economou, "Development platforms for mobile applications: Status and trends," *IEEE Software*, vol. 28, no. 1, pp. 77–86, 2011.

[8] E. Tsekleves, R. Whitham, K. Kondo, and A. Hill, "Investigating media use and the television user experience in the home," *Entertainment Computing*, 2011.

[9] L. Klopfenstein, A. Seraghiti, S. Bonino, A. Tarasconi, and A. Bogliolo, "Multicast TV Channels over Wireless Neutral Access Networks," in *Proceedings of Int. Conf. on Evolving Internet*, 2010, pp. 153–158.

[10] L. Klopfenstein, S. Delpriori, A. Seraghiti, and A. Bogliolo, "Protected Delivery of Multimedia Contents over Multicast IP Networks: an Open-Source Approach," in *Proceedings of the International Symposium on a World of Wireless, Mobile and Multimedia Networks*, ser. WoWMoM-2011. IEEE, 2011.

[11] A. Schroeder, "Introduction to MeeGo," *IEEE Pervasive Computing*, vol. 9, no. 4, pp. 4–7, 2011.

[12] M. Butler, "Android: Changing the Mobile Landscape," *IEEE Pervasive Computing*, vol. 10, no. 1, pp. 4–7, 2011.

 ISBN: 978-1-61208-148-9