Introducing openBOXware for Android: The Convergence between Mobile Devices and Set-Top Boxes

Lorenz Klopfenstein[1,2] Saverio Delpriori[1,2] Gioele Luchetti[1,2]
Andrea Seraghiti[1] Emanuele Lattanzi[1,2] Alessandro Bogliolo[1,2]
[1]STI-DiSBeF - University of Urbino, Urbino, Italy 61029
[2]NeuNet Cultural Association, Urbino, Italy 61029
E-mail: lck@klopfenstein.net saveriodelpriori@gmail.com luchetti@sti.uniurb.it
andrea.seraghiti@uniurb.it emanuele.lattanzi@uniurb.it alessandro.bogliolo@uniurb.it

***Abstract*—Multimedia contents delivered over residential and mobile IP networks are among the main driving forces of the Internet. The pervasiveness of connected devices capable of receiving and decoding multimedia streams has induced 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, smartphones are the favorite 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 smartphones and tablet PCs, capable of handling HD multimedia streams while also retaining the distinguishing features of mobile devices, enables the convergence between personal communication devices and home entertainment appliances. This paper introduces openBOXware for Android, an application suite which makes it possible to use any Android device as a set-top box, allowing end-users to take advantage of the tailored run-time environment of their personal mobile devices while watching television in the comfort of their living rooms. OpenBOXware exploits technology convergence for usability, in the attempt of enhancing the accessibility of the Internet by providing a TV-like usage experience. The paper presents the key features of openBOXware, outlines the implementation on top of the Android application framework, and shows representative use cases.**

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

I. Introduction

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 smartphones 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 include mobility) [2], accelerating the convergence between mobile and residential broadband markets and creating the conditions for delivering mobile TV services [3]. IP traffic trends and forecasts [4], [5] indicate that multimedia contents delivered over residential and mobile IP networks are among the main driving forces of next generation networks.

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 smartphones (or similar hand-held 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 [6].

Exploiting add-ins and configuration options to create a perfectly tailored run time environment on a smart phone is an intriguing pastime that engages the vast majority of end-users. As a result, both the quality of experience offered by smartphones and the effort devoted to personalize them keep end-users from using (or at least from personalizing) 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 [7], [8], 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 very well suited for media consumption and is in contrast with the *lean-forward* attitude typical of smart phone users, which has sustained the market of mobile applications [9], [10].

On the other hand, personal handheld devices have never threatened the market of media centers and STBs because of their tight design constraints 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 has been closed, however, by the last generation of smartphones and tablet PCs, which support HD video streams and are equipped with HDMI interfaces, and by the advent of IP boxes providing the same application framework of the most popular mobile devices [11].

In a preliminary version of this paper [1], the authors investigated the possibility of making an Android tablet PC work as a STB in order to allow end-users to take advantage of their personal runtime environment in the comfort of their living room. This paper moves a step forward by outlining the key features and the implementation details of openBOXware (OBW) for Android, a modular application suite which makes it possible for Android devices, including smartphones, to switch from a *lean-forward* to a *lean-back* usage mode in order to provide a TV-like experience of Internet contents. The main purpose of openBOXware is to exploit this convergence, making both the advanced features of Android and the unlimited contents available on the Internet directly accessible to television viewers, presenting contents in a familiar way: as linear TV channels, possibly controlled with a simple remote control.

The rest of the paper is organized as follows: Section II presents the concept and the main features of openBOXware, Section III outlines the software architecture and the implementation details, Section IV shows representative use cases, and Section V draws conclusions.

II. OPENBOXWARE FEATURES

OpenBOXware is an open-source framework built on top of Android to provide a TV-like experience of multimedia contents taken from heterogeneous sources (in terms of format, protocol and access mode), while also allowing the end-user to enjoy all the applications installed on the underlying Android device. To this purpose openBOXware sports a custom user interface conceived to offer a lean-back usage experience by means of three home screens, granting access to: the *media library* (Figure 1), the list of *openBOXware applications* (Figure 2), and the list of all other *Android applications* installed on the device.

The media library is the default home screen, which allows the end-user to find media channels and to select the one to watch. Multimedia contents are made available by special add-ins, called *media sources*. A media source is a tree of nested multimedia nodes. Leaf nodes are *playable*, in that they can be forwarded to the media player for playback, while all other nodes are *explorable*, in that they allow the media library to navigate their content and display the list of children nodes. Examples of media sources include IPTV channels, Internet TV channels, UPnP/DLNA clients granting access to the multimedia contents made available by the UPNP/DLNA servers discovered in the LAN, collections of media elements stored in the local file system, and collections of online multimedia contents.

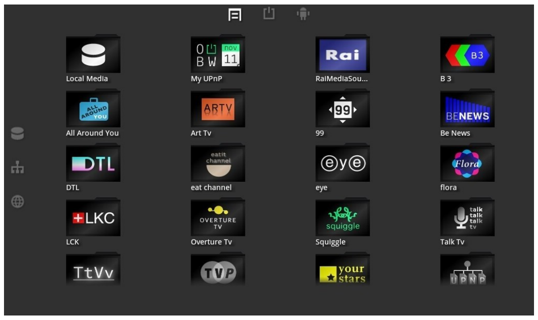

Figure 1. Media library home screen.

Media source nodes may contain metadata, including title, duration, and an icon, that can be displayed by the media library to provide a richer and more vivid browsing experience and to help the end-user to decide which content to pick.

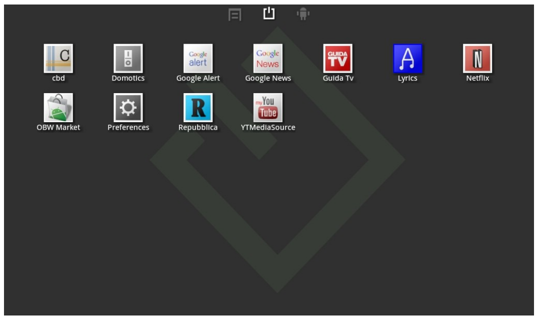

Figure 2. OpenBOXware applications home screen.

Figure 1 shows the media library home screen, with the icons of all the media sources installed in the device. When a media source is selected, the media library shows the icons of its children nodes. The three small icons on the top of the screen can be used to switch among the three homes, while the ones on the left represent filters that can be applied to the media sources based on the location of the contents they link to: local file system, LAN, Internet.

Playable media source nodes provide a TV-like watching experience by offering both linear channels or contents on demand. A content on demand is a media source node associated with a single resource which is played back whenever the media source node is selected by the user. The node's content do not change and depend exclusively on the user's choice. A linear channel, on the contrary, can be either a link to a continuous stream provided by a live streaming server, or a (possibly unlimited) list of disjoint multimedia

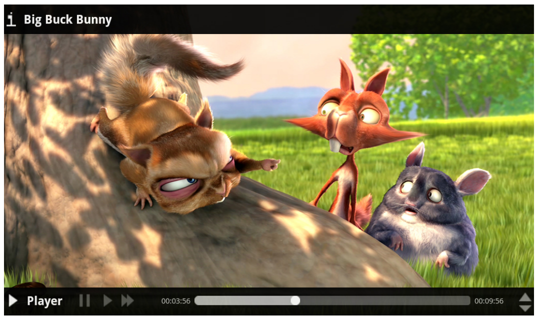

Figure 3. OpenBOXware media player with visible control bar.

elements which are glued together by the node and played back as a continuous stream. In this case, contents may depend on the moment in time the user decides to tune in.

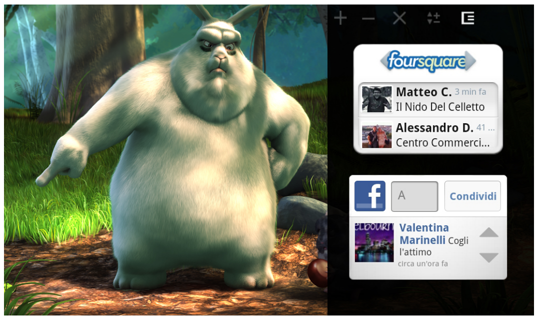

Figure 4. Sidebar displayed over the media player.

OpenBOXware applications are special Android applications which make use of additional features provided by the openBOXware library included in the Software Development Kit (SDK). Applications can be executed in *fullscreen* (if they take over the whole screen area, covering up other applications), in *background* (in case of services that do not require any graphic user interface), or in *sidebar* (the case of widgets that can be displayed on a small part of the screen letting the top-level fullscreen application shine through). An example of sidebar applications displayed on top of the media player is provided in Figure 4.

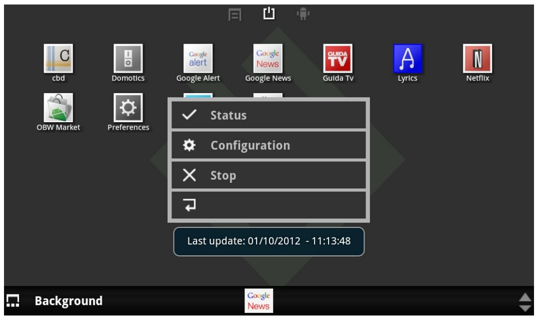

Figure 5. Background application management dialog box raised from the control bar.

To support a lean-back usage experience, openBOXware provides an overlay interface element, called *control bar*, that is displayed at the bottom of the screen with minimum interference with the foreground activity. User controls are organized in sections which differ in type and scope. Each section takes the entire area of the control bar, so that the control sections are displayed one at the time. This is due to two main reasons: first, to limit the area of the screen taken by the control bar, second, to make it easier for the end-user to issue commands which are limited in scope. In particular, both the switching among the sections and the controls contained in each one can be operated by means of the four arrows and the “OK” button available in any remote control. Control bar sections include: *playback* (Figure 3), containing a seek bar together with pause, skip, and stop buttons; *volume*, which controls the volume settings of the device; home, which provides short cuts to the three home screens; *sidebar*, which controls the configuration of the sidebar and the widgets displayed in it; and *background*, which provides a scrollable list of services running in background and allows the end-user to pick one in order to check its status, change its configuration, or stop it (Figure 5).

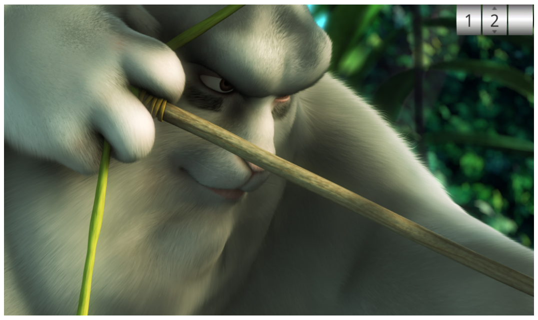

Figure 6. Overlay digits providing feedback of the zapping command issued while watching a movie.

To further improve usability, any playable media element can be associated with a unique 3-digit number to be used as a short cut to directly zap to that channel from the media player (or from any home screen) without going through the media library and browsing media sources. Figure 6 shows the three digits which appear in overlay whenever the end-user presses a numeric key in the remote control to exploit the zapping functionality.

OpenBOXware also supports the so called *configurable media sources*: essentially standard media sources bundled with companion openBOXware applications that can be used to configure them. In what ways and to what extent a media source can be customized depends on the structure of the multimedia provider for which the media source is developed, but the auxiliary application typically provides a user interface to change settings and preferences, to set search criteria, or to apply filters. Examples of configurable media sources include a UPnP client with an auxiliary application to be used to associate a channel to a specific directory or to a specific file made available by some UPnP server in the LAN, or a YouTube media source with an auxiliary application allowing the end-user to create a channel associated with a specific query and search options. In both cases, the channels created by the end-user through the companion applications appear as new children of the corresponding media sources and they can be used as any other playable node and possibly associated with numeric shortcuts for quick zapping.

This advanced feature grants to openBOXware the flexibility required to provide a TV-like experience of any online contents (even if they are not natively organized in linear channels) making them available to a broader audience. For instance, people not accustomed to browse the Internet can take advantage of thematic channels preinstalled or created by other expert users. In addition, parent can create channels of contents expressly selected and filtered for their children.

III. OpenBOXware implementation

OpenBOXware is built on top of Android which is, in its turn, built on top of Linux kernel. The Android architecture consists of three main layers: *i*) the *Android runtime*, based on the *Dalvik virtual machine* (VM) with additional support libraries, *ii*) the *application framework*, and *iii*) the *applications* which run on it [11]. For portability and compatibility reasons, openBOXware has been fully developed at the application level.

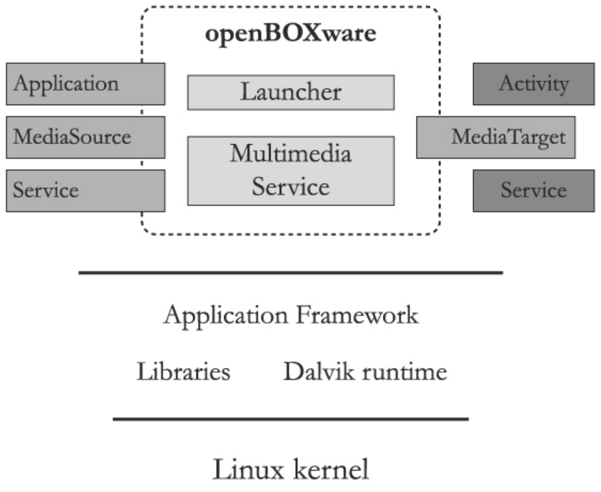

Figure 7. Software architecture of openBOXware.

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 application views.

The *openBOXware core* is implemented as a package containing a launcher and several additional components, including background services and other activities, which handle multimedia functionalities, notifications, and sidebar widgets. As mentioned in Section II, the user interface of openBOXware discriminates between normal Android applications and special openBOXware applications (identified by the intents they are registered to handle, as detailed below). An openBOXware application is nothing more than a standard Android application, mainly composed of activities and services, which additionally relies on the APIs exposed by the *openBOXware SDK* and links to the *openBOXware library*. Those APIs, written on top of the features provided by the *vanilla Android* platform, enable the application to be integrated inside the openBOXware environment and give access to advanced multimedia features through high level programming interfaces.

A schematic representation of the software architecture is provided in Figure 7, where the openBOXware platform rests on the Android application framework. Media sources and openBOXware applications are represented as boxes partially overlapping the openBOXware core to denote the fact that they make use of the extended features provided by the Software Development Kit, while all other Android applications (activities and services) do not.

A. Home application

The core package, once installed onto an Android device, works as a launcher which provides the three home screens from which media sources can be explored and any other application is launched.

It is worth noticing that multiple launchers can be installed on the same device, but only one at the time can be set as default and run in foreground. Hence, the device must be setup to allow the user to select the launcher in order to switch from a *lean-forward* to a *lean-back* use of his/her own device. This can be achieved either by changing the default launcher, or by avoiding to specify a default one (in this case the choice is made every time the end-user taps the home button, through an Android dialog box – shown

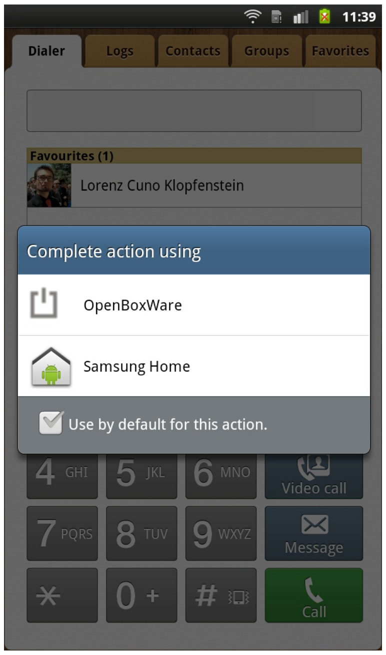

Figure 8. Launcher selection dialog box.

in Figure 8 for a Samsung smartphone – which allows the user to pick the desired launcher).

Once the *openBOXware launcher* is launched, it becomes the main graphical user interface of the system, acting as an application launcher and displaying additional messages through the system bar and a custom control bar. As mentioned in the previous section, the launcher provides three separate home screens: the media library, the openBOXware application list, and the Android application list.

B. Media source

A media source is implemented as an Android service, reacting to the `openboxware.media.source.ACCESS` intent. The media library (and in fact any other Android application) can search for media sources by querying for that specific intent and getting a list of the installed services. Media sources may use an additional category (namely `openboxware.media.source.LOCAL`, `LAN`, or `INTERNET`) to specify the location of the data the media source points to. These are known as content-type flags. The Android package manager can filter out specific kinds of media sources based on the location specified in the intent query. From a user-interface perspective, filters are activated by the three clicking on the three icons available on the left-hand side of the media library home screen (see Figure 1).

Media sources implement a set of remote procedure calls (RPCs) using the Android *AIDL* interface. The openBOXware library provided by the SDK includes a collection of interfaces and higher level classes (contained in the *Tahweed* APIs) that make implementing such a RPC interface much easier and more structured. The developer can simply implement a small set of classes and connect the exploration actions (i.e., their RPCs) to the corresponding back-end code needed to access the specific media content.

Each node of a media source is an instance of the `MediaSourceNode` class, which provides a set of abstract methods that need to be overridden in order to be used by the client to fetch children nodes and multimedia resources.

Playable media source nodes provide an enumerable sequence of content items, which will be played one after the other by the media player. Each item of this stream is an instance of the `MediaElement` class and can link to any remote or local multimedia resource (including images, audio files, and videos) through its URI.

Since every playable node includes a – possibly unlimited – stream of multimedia elements, as mentioned in Section II they can be used to create a linearized channel. An example thereof can be a virtual TV channel built from multiple separate videos, possibly encoded in different formats and taken from different sources. Media source nodes can also transform any feed of data into a linearized stream of media. For instance, any feed containing links to images (an RSS feed, possibly) can be represented as a stream of pictures, which is then played back by the media player as a slideshow.

C. Media library

The media library is the default home screen of the openBOXware launcher, which represents the main user interface to the platform's multimedia exploring and streaming capabilities: its primary function is to list all media sources installed and available to the user, allowing him/her to explore media sources' contents, discover media channels, and select the ones to watch.

Media sources are displayed in a grid in alphabetical order, as shown in Figure 1. When the user decides to select one particular media source, the openBOXware library opens an *AIDL* connection to the respective service and starts communicating with the service implementing the media source (through the RPC interface described before). Media source's nodes are explored hierarchically: the history of the exploration is kept in a stack and displayed on the left-hand side of the media library grid, enabling the user to easily understand his/her position in the media source's structure and to navigate back either by using the *back* button or by clicking on the stacked ancestral nodes.

When the user clicks on the icon of an explorable node the media library pushes it into the stack and displays its children nodes. When he/she clicks on a playable node it is passed on to the integrated media player, which attempts to play back the contents of the node starting from the first multimedia element.

Finally, the media library includes a link to the Android marketplace with a shortcut that quickly filters out installable media sources: the user can explore applications which are available to download, install them, and get back to the media library to make use of the new media sources and of the multimedia resources they provide access to.

D. Media player

The media player (Figure 3) is the openBOXware component that handles all multimedia playback requests by the media library (and by other openBOXware applications): it is implemented as a single activity sporting a simple user interface and capable of playing back a variety of linearized contents as described by media source nodes (instances of `MediaSourceNode`) exposed by the media sources installed on the device.

The player activity reacts to the `openboxware.mediaplayer.PLAY` intent, which signals the request for playback issued by another application. The intent structure contains all the data required by the media player to initiate the playback. In case of media source nodes containing a single media element, the intent contains directly the `MediaElement` instance to be played back. In case of playable media source nodes containing more elements, the intent contains a so-called *media source node identifier*, which will be used by the media player to open a connection to the original media source, make a request for the node to playback via RPC, and then start enumerating the multimedia resources provided by the node. Each single media element returned by the node is reproduced as part of a continuous linear stream.

The media player determines the type of each media element to play back. Videos and audio files are played back using the default Android MediaPlayer component, while pictures are displayed by a custom slideshow component. Media identification relies on the metadata provided by the developer of the media source, which include a MIME type specification (i.e., a string formatted according to the *Multipurpose Internet Mail Extensions* standard, which gives a textual description of the content type of a file, such as “text/html” or “video/x-h264”). If absent, the media player will attempt to guess the nature of the media element (e.g., by checking the file extension in the element’s URI or by other heuristics). When an unsupported (or unknown) media element is passed to the media player, playback fails and the media player attempts to skip it and go to the next element of the media source node.

The media player activity does not display any additional user interface elements, nor any buttons that enable the user to control media playback. Instead, the player emits a sequence of intents to the system notifying its current status, time of playback, and other additional data (like the name of the current media element which is being played back). These data intents can be intercepted by any other component of the system to keep track of the player’s actions. Most notably these intents are used by the control bar which, when shown, displays playback progress, common playback controls, and other data directly to the user (as shown in Figure 3).

On the other hand, the media player can also be controlled by generating special intents which represent commands to pause, resume, skip, or stop playback. By default, these intents are sent by the control bar when the user clicks on the corresponding control buttons, but they can also be used by other applications and services that might need to interact with the media player.

E. Sidebar

The sidebar (shown in Figure 4) is implemented as a single instance activity which is displayed as a transparent dialog, covering only a small side of the screen. Because of limitations of the Android window manager, dialog activities which do not cover the whole screen take the entire input focus of the user. Thus, interaction with the underlying fullscreen activity is either impossible or unreliable because of its dependence on device-specific implementation choices. This problem is partially mitigated by the fact that the most common fullscreen activity, i.e., the media player, does not provide any touch interface and all interactions usually pass through either the remote control (via intents) or the control bar (which is never displayed together with the sidebar).

The sidebar activity acts as a so-called *widget host*, which can accommodate any number of external widgets implemented by other applications installed on the system. Widgets commonly used on Android devices include simple front-ends to communication applications (e-mail readers, contacts lists, incoming text message viewer, feed readers, ...) and social networking clients (Twitter, Facebook, LinkedIn, ...).

Widgets added to the sidebar are identified by a unique *appWidgetId* that is used to grant persistence to the list of hosted widgets, to store their display order as decided by the end-user, and to handle removal. Widgets are automatically updated by the Android system.

The sidebar provides simple control buttons that can be used to add, move, or remove widgets either through the touch screen or from a remote control.

F. OpenBOXware applications

Different classes of applications can be developed to target the openBOXware environment, in order to make use of the features of the framework or to integrate with the media library. In particular, openBOXware supports four ways of integrating third party add-ins.

Interactive applications (namely, *fullscreen* openBOXWare applications) can be implemented as activities that will run taking the whole device screen over and delivering an immersive usage experience to the user. As mentioned in Section II, these applications run one at a time, demand exclusive focus from the user and can also rely on the media capabilities of openBOXware to explore media resources or demand media playback. Applications of this class will be listed separately in the openBOXware applications list by the launcher (they will be hidden from the standard Android applications list of the openBOXware launcher and of other Android launchers, by default).

Background applications, which usually play the ancillary role of helper services to a fullscreen activity in order either to periodically fetch updated data or to poll for some resource, can be implemented as Android services. These applications will also appear in the openBOXware applications list and can be explicitly launched in background by the user. The control bar will also provide a graphical interface that allows the user to interact with such background services, updating their configuration and eventually terminating them. A background application can interact with the openBOXware framework, explore media resources, and invoke other fullscreen activities when needed.

Other applications that do not request the full focus of the user and allow to glance at information without interrupting the main usage experience can be implemented as standard Android widgets. Widgets have limitations on how often their code runs and how their interface is displayed, but they can be included in so-called host applications. Creating a widget allows the application to be hosted by the openBOXware *sidebar*, enabling the user to display the application while interacting with another fullscreen application (for instance, the media player). Every widget can be hosted by any widget host, for instance other Android launchers that provide this functionality.

Finally, it is worth mentioning that *media sources* are application as well. In particular, they are implemented like standard Android services, but they implement a specific interface that enables the media library (and in fact any other application written using the openBOXware SDK) to access the service and fetch informations about the available media resources from the media source's hierarchical resource tree.

Being implemented as standard Android activities and services, openBOXware applications are able to react to common Android intents. It is up to the developer to include the default Android intents in their applications to enable them to be launched through any Android launcher instead of relying only on the openBOXware front-end. It is worth noticing, however, that in this case the openBOXware services might not be available.

In order to be listed and launched as openBOXware applications, activities must handle the `openboxware.gui.FULLSCREEN` intent and should extend the `FullscreenActivity` class included in the SDK. This allows the application to easily access all openBOXware features (e.g., raising the control bar, displaying the sidebar, issuing commands to the media player, or raising notifications, as described in Section III-H3). Fullscreen activities also automatically handle their own theme when launched, in order to match the look of the platform and the size of the screen without any additional effort for the developer.

Similarly, background services must handle the `openboxware.gui.BACKGROUND` intent in order to be listed among the openBOXware applications, and they have to extend the `BackgroundActivity` class to gain access to openBOXware features. In particular, this allows them to be launched, monitored, configured, and stopped by the openBOXware control bar.

As already mentioned, media sources handle the `openboxware.media.source.ACCESS` intent and extend the `MediaSource` class, which implements the basic AIDL which allows media source clients (like the media library) to discover multimedia content through RPC calls to the service. Media sources can also rely on the high-level *Tahweed* multimedia API, which provides a structured object-oriented wrapper around the imperative RPC constructs on which the Android inter-process communication system is based.

While all Android applications have access to platform's *context* in order to query base services and managers, openBOXware applications can also access their `OBWApplicationContext`, which enables them to integrate with the features of the platform and provides a mean to easily share status and information between different activities/services which are part of the same application. In particular, this allows the developer to implement an openBOXware application that supports all execution modes (i.e., fullscreen, background, sidebar) and switches among them without losing data and settings.

G. Control event injection

The control bar is implemented as a standard Activity and communicates with the openBOXware back-end through intents (e.g., as already mentioned, the media player signals its state continuously using intents and can be controlled by broadcasting other specific intents representing user's commands). The current implementation of the control bar allows the user to interact with the openBOXware environment both in a *lean-forward* stance, by using the touch interface of the device, and in a *lean-back* stance, by using a remote control to issue commands. Any Bluetooth remote that is able to connect to the Android device can be used to this purpose.

In order to maintain full compatibility with existing Android devices, the decision was made not to implement a custom Android build with system-wide changes. This hinders the capability of openBOXware to provide deep integration with a remote control promoted to a full-fledged interaction device capable of injecting interface events to Android applications other than the openBOXware launcher. On the other hand, this kind of customization could be provided in device-specific openBOXware distributions targeting Android IP boxes.

H. Advanced features

1) Configurable media sources: Configurable media sources are standard media sources that include a companion openBOXware application. The media source and the configuration application share the same Android application package and thus also share a common space of isolated storage on the device. This storage area is used to store structured information (usually a SQLite database, a shared preference structure, or simple files in any parsable format) that can be altered by the application and read by the media source in order to display custom contents based on queries and preferences specified by the end-user.

A representative example of a configurable media source is shown in Section IV.

2) Zapping: Any playable media source node can be marked as *pinnable* in order to enable the zapping functionality. In practice, this means that the media library is allowed to store a reference to that particular node, i.e., to *pin* it and associate it to a number. OpenBOXware lets the end-user create a numbered set of preferred channels, which he/she can play back quickly and easily by *zapping* to the corresponding numbers. On the other hand, a pinnable node requires the ancestral media source to be able to regenerate that node and its complete state (which could require a complex interaction with the back-end content provider). This is done by generating an identifying code for the pinned node (i.e., an arbitrarily complex string constructed by the media source and containing all relevant information) that can be parsed at any time (even on a different instance of the same media source or after a full device reboot) in order to regenerate the original media source node.

In practice, the launcher makes use of the zapping function by allowing the user to pin nodes when they are declared as pinnable by the media source developer. When the user executes a long-press on a pinnable node displayed in the media library (holding the node for a couple of seconds), the library displays an overlay that enables the user to select a channel number for that particular node. Once the user confirms the input, the node is pinned and associated with that number.

Subsequently, whenever the end-user picks the channel number using either the device or the numeric keys of the remote control, the media library attempts to retrieve the corresponding pinned node and use its identifying code to restore the desired `MediaSourceNode` instance. The node is then forwarded to the media player and directly played back.

3) Notifications: Standard Android notifications can be sent to the Android notification bar by any application by instantiating a new notification and sending it to the *NotificationManager*, which is part of the standard Android programming interface. The notification is then displayed in the notification bar until it is dismissed by the user.

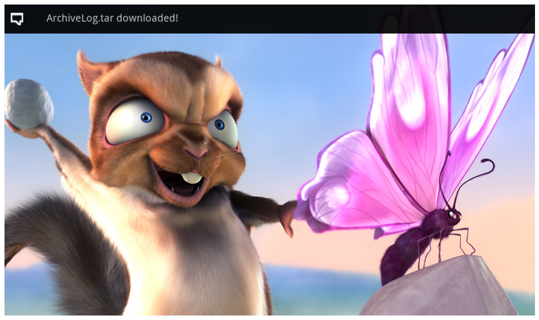

Figure 9. Example of an openBOXware notification.

Since the Android notification bar is covered by the openBOXware launcher by default, these notifications are hidden to the user and cannot be easily accessed. While any application can still make use of the default notifications system, raised notifications will only be displayed once the openBOXware launcher is terminated. If, on one hand, this behavior is intentional in that it avoids notifications, which usually interrupt the normal usage experience by prompting for user's attention, to detract from the *lean-back* usage experience typical of openBOXware, on the other hand, a less-obtrusive form of notifications are needed, to be possibly consumed while watching multimedia contents.

OpenBOXware notifications can be displayed by applications using the openBOXware APIs included both in the `FullscreenActivity` and in the `BackgroundActivity` base classes, instead of relying on the default Android notification manager.

The notification request is taken over by the openBOXware environment, which will display it by using a simple notification overlay on one edge of the screen (see Figure 9), while also forwarding the same notification to the underlying default NotificationManager to ensure that the notification can be read and acknowledged later, even if ignored while on screen.

When the user reacts to an openBOXware notification (by clicking or touching the overlay icon while displayed), the system hides both the overlay notification and the corresponding entry in the standard Android notification bar. At the same time the notification raises a custom intent that can be handled by the application.

IV. Release

A. Public demonstration

A pre-alpha version of the openBOXware framework was tested and publicly demonstrated on November 11, 2011 in Urbino (Italy), using a living room setting installed in a conference hall at the University of Urbino.

A HD TV set was connected to a Samsung Galaxy SII smartphone running openBOXware, providing a replica of

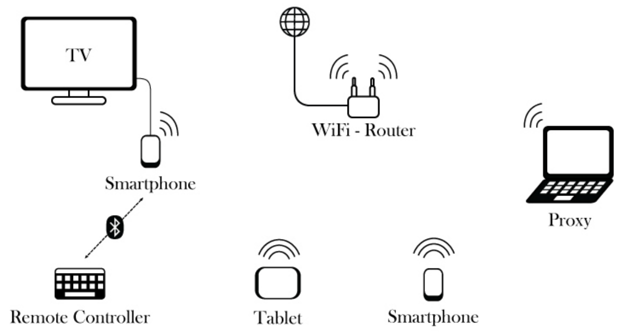

Figure 10. The openBOXware test bed, including a TV set driven by an Android smartphone controlled by a Bluetooth remote and several other Android devices connected to the same WLAN.

the screen user interface (in its original display resolution) through the HDMI port. The smartphone was placed close to the TV and far away from the sofa in order to force the end-user to rely on a remote control to issue commands (namely, a Bluetooth *Logitech diNovo Mini wireless keyboard* that can work both as a remote and as a wireless keyboard/mouse input device). The smartphone was connected to the Internet through a Wi-Fi router. Other Android devices (smartphones, tablet PCs, and IP boxes) running openBOXware were connected to the same WLAN for testing purposes. One of the smartphones was also running an UPnP server in the background. All devices were able to access Internet resources, files shared by the UPnP server, and resources stored on a HTTP server installed on a local computer playing the role of a proxy of video contents from RAI Radiotelevisione Italiana.

A schematic representation of the setting is provided in Figure 10. The setup was used to demonstrate the usability of openBOXware in a pure *lean-back* setting, to test the multimedia playback capabilities of the devices involved, and to show the key features of the platform by means of representative use cases. A video log of a demo is available at: http://youtube.com/watch?v=7RzdIiz1EQs.

Demonstrated features include: home screen navigation (see video log at 2:49s), media source exploration of representative media sources such as a Youtube channel, a UPnP client, and a HTTP proxy of RAI streaming contents (at 4:18s), playback of a full HD video, control bar (at 5:28s, with event injection from both the touch screen and the remote control), sidebar (5:57s), channel shortcut assignment and zapping (6:55s), multiple launch modes (8:10s) shown on a simple Google News client supporting both full screen and background execution modes, and configurable media source samples, such as UPnP and Youtube clients (10:20s).

The experiments conducted on configurable media sources are worth to be described in detail. Both the YouTube and the UPnP media sources developed for testing purposes had their own companion applications to be used to customize them outside the media library (which in fact provides only an interface for content consumption). The UPnP configuration application (called "My UPnP") showed the list of all available UPnP servers on the current network and allowed the user to pick a device (or a specific shared folder on that device) to be made available within the UPnP media source. Upon configuration, a node was added to the media source tree which directly connected to that folder allowing its contents to be played back by the media player.

Similarly, the Youtube configuration application (called "My YouTube") allowed the user to generate a set of custom channels, each based on a custom search query. A "snowboarding" channel (with contents sorted by publication date) was created and used during the demo.

Since both the configurable media sources used for testing purposes supported pinnable media source nodes, numeric shortcuts were added to the custom nodes in order to make them look as linear channels.

As a final remark, both UPnP and YouTube channels were dynamic in nature, in that the contents they granted access to changed over time. In particular the UPnP channel, when selected from the media library, connected to the UPnP server to obtain the list of resources available at that particular time in the device/folder specified by the configuration application. Similarly, the search criteria associated with the custom YouTube channel were applied whenever the channel was invoked. In both cases, the multimedia contents were organized at runtime in a list of media elements and passed to the media player for continuous playback.

This inherent update capability of the custom channels was demonstrated during the demo by showing on the TV screen the most recent snowboarding video on YouTube, and the slideshow of the pictures taken during the presentation with a smart phone running a UPnP server service.

B. Beta release

The openBOXware framework has been released on March 1st, 2012 and can be installed on any Android device: https://play.google.com/store/apps/details?id=it.uniurb.openboxware.launcher. The Google Play marketplace has been populated with the core environment and some default add-ins (sample media sources and applications) that can be installed and integrated with the launcher. Stubs of representative use cases will be also published to provide a base for the development of other resources.

The current release is compatible with Android version 2.2.1 and superior (compatibility with Android ICS 4.0 has not been assessed yet) and it has been tested on a variety of devices, including: *Motorola Atrix* smartphone, based on nVidia Tegra 2 SoC (Dual ARM Cortex-A9, at 1 GHz), with 1 GB RAM, 4.0' screen (540×960), running Android 2.2; *Asus Transformer TF101* tablet/notepad, based on nVidia Tegra 2 SoC (Dual ARM Cortex-A9 at 1 GHz), with 1 GB RAM, 10.1' screen (1280×800), running Android

3.2; *Samsung Galaxy SII* smartphone, based on Exynos SoC (Dual ARM Cortex-A9 at 1.228 GHz), with 1 GB RAM, 4.3' screen (480×800), running Android 2.3; and an Android IP-box (http://www.artwaytech.com/goodpro.php?id=266) based on Samsung PV210 SoC (Cortex A8 CPU at 1 GHz), with 512 MB RAM, 2GB Flash memory, HDMI video & Audio output, running Android 2.2.

V. Conclusions

The advent of digital broadcast television, the diffusion of mobile broadband networks, the computational power of smartphones, and the success of open application frameworks have enabled the de facto convergence between mobile devices and set-top boxes and between broadcast television and online multimedia contents. Such a convergence has been exploited so far to create a thriving market of connected devices (including the so-called IP-boxes and smart-TVs) and to enhance the usage experience of television viewers (by making available additional IPTV and Internet TV channels, and by granting Internet browsing capabilities to any television set). Taking a different perspective, however, the same enabling conditions could be exploited to enhance usability and to reduce device diversity.

This is the starting point of this paper, further supported by two observations: first, watching television is a much more inclusive experience than browsing the Internet; second, in spite of the proliferation of any sort of connected devices, smartphones are the preferred ones with clear competitive advantages which prevent them to be outstripped in the near future.

This paper has introduced openBOXware for Android, an application suite that can be easily installed in any Android device (including a smartphone) to make it work as a set-top box while also maintaining compatibility with all the applications installed in it. The openBOXware core encompasses a launcher, designed to provide a lean-back usage experience in order to take advantage of the personalized runtime environment of the smartphone while watching television in the comfort of a living room, and a SDK that can be exploited to develop media sources and Android applications compatible with a TV-like usage mode.

The paper has outlined the key features of openBOXware, discussed the implementation choices, and presented representative use cases. In particular, it has been shown that openBOXware provides the opportunity of creating custom TV channels made of linearized contents possibly taken from heterogeneous sources (including local file systems, UPnP servers, streaming servers, and HTTP servers). Custom channels can be associated with numeric short cuts in order to make it possible to directly zap into them using a standard remote control. This makes it possible for elders and kids who are not used to browse the Internet to gain access to online contents organized in personalized linear TV channels by their family members.

OpenBOXware will be available on the Android market since March 1, 2012.

Acknowledgments

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). The authors would like to thank RAI Radiotelevisione Italiana, BAX srl, and IMAB Group SpA for taking part in the setup of the demo, as well as Beatrice Bucciarelli and Lorenzo Bravi for their contribution to the implementation of openBOXware.

References

[1] L. Klopfenstein, S. Delpriori, G. Luchetti, E. Lattanzi, and A. Bogliolo, "Making an Android Tablet Work as a Set-Top Box," in *Proceedings of the International Conference on Andvances in Future Internet*, ser. AFIN-2011. IARIA, 2011, pp. 64–68.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] 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.

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

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

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

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