UPnP in Python for the Maemo Platform

Dept. of CIS - Senior Design 2009-2010

Zachary Goldberg
zgs@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Jonathan M. Smith
jms@cis.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

ABSTRACT

The modern individual is faced with a complex problem of having devices in several locations that frequently are incapable of interoperation and hence a source of major frustration. Technologies such as UPnP (Universal Plug and Play) attempt to solve this problem by providing protocols that devices can support and hence provides them with nearly limitless interoperability. The problem remains, however, for new devices to support these protocols and for developers to be able to easily implement them in their applications.

1. BACKGROUND AND DEFINITIONS

The technologies herein discussed will together describe a solution to the problem of media interoperability. The technologies include UPnP, Maemo, C/Python bindings via GObject introspection and PyGi. Some definitions of these technologies follows.

UPnP, or Universal Plug and Play, is a set of protocols which define how a device can discover other devices on a network and access and control the content on those devices [16]. These standards are capable of delivering an impressive home media experience, allowing one to store content in several places and render it on a wide variety of other devices [14].

Maemo is a platform for mobile devices based around a Debian Linux distribution. It's core is open source under the GPL license and it builds around common Linux technologies [2]. Nearly all technology that is used in Maemo is also available under a normal Linux distribution and visa versa.

The classic technique for generating bindings for python from C is to generate them statically. When the python program loads it imports the static bindings as a single blob and can then make the foreign calls. This has several disadvantages, including the requirement that all of the symbols from the C library be loaded at start up time into memory.

A more modern technique, generally referred to as dynamic or run time bindings, has recently become popular. Instead of a large memory and time penalty at boot up dynamic bindings spread that penalty out during run time. This is accomplished by generating the necessary foreign function interface capabilities when the functions are called. This requires a "middle ware" library to receive the foreign function calls, generate the real remote call and perform it on behalf of the running application. It must also return the values and clean up memory appropriately.

2. INTRODUCTION

As consumers build up larger media collections and become accustomed to always having their media with them their expectations for the how and when they'll be able to consume that media are growing. Unfortunately often the device a user will use to store their media is not the same device that the user would prefer to consume the media on. Most media is stored now in portable devices such as advanced cell phones or media players and because of practical constraints these small devices are not high quality renderers. The speakers and displays are very small and low powered. To fully take advantage of the media and render it in the best way possible a larger and higher powered device is often necessary, such as a living room speaker set or large screen television. Thus a mechanism, such as UPnP, must be used in order to manage the transfer of the media for rendering on another device.

Figure 1: Device broadcasting media to home entertainment system

The goal is to help extend this rich cross-device experience to the Maemo platform.

Maemo comes with built in UPnP for support its core applications. Maemo is, however, designed to have much more functionality than what is simply provided by its core components. There is a rich community building third party applications for the platform. Many of these applications are written by individuals on their own time, and as such their time is limited. They therefore often wish to use a language

which allows for faster prototyping, and for Maemo the most frequently used rapid prototyping language is Python.

At present there is no way to use the native C-language UPnP libraries from within the python-maemo packages. Hence any applications on Maemo or Linux in python do not have access to this rich additional functionality. Thus the goal is to provide a set of python bindings for Maemo's UPnP libraries and then build out the python offerings which use UPnP.

3. RELATED WORK

There are a few open source projects related to UPnP support which have been investigated.

3.1 UPnP Protocol

The UPnP protocols have been around for several years and have been implemented in many devices and platforms by many different libraries. The UPnP specification [16] is extraordinarily detailed and takes advantage of numerous technologies such as HTTP, XML and SOAP. UPnP conceptually consists of three types of entities: devices, services and control points. A device is any physical device which implements the UPnP protocols, a service is something the device can do or perform for a user. A control point is a hub for coordinating devices and asking them to perform services.

UPnP, generically, also establishes three roles. A single device can take on more than one role. The roles are: media renderer, media server and control point. A media render implies that the device can retrieve media and render it for a user. A server must be capable of storing media and delivering it to other devices. The control point is responsible for accessing media servers and allowing the user to chose where the media should be rendered, and initiate and control the rendering process. A media server is typically a personal computer or a server, a render is most often a TV or DVD player and a control point can take many forms: a mobile device, a computer, a portable remote control etc. Via GUPnP-Python we wish to be able to create applications which can be any or all all three types of roles.

3.2 BRisa

BRisa [3] [12] is a python UPnP framework. It is a pure-python implementation of the UPnP protocols and a media server, renderer and control point. BRisa is also targeted at the Maemo platform, however it is not an official framework for UPnP that ships with the Maemo platform.

BRisa walks the line between being a simple library and framework and a full blown application. BRisa is implemented in a multi-threaded and pluggable manner, allowing it to have some very interesting features such as Flikr and Shoutcast integration [3]. These capabilities have set a new benchmark in the field of what UPnP is capable of; *i.e.* that media servers do not simply have to deliver local content, they can integrate with modern web application sources. However, this advantage is at the same time a disadvantage. The BRisa api is not as simple and elegant as a simple UPnP library might be, and as such makes it harder to use as a platform for developing new applications. Additionally, not having a low-level language back-end limits the applications to which BRisa can be ported. E.g. a platform like Maemo whose core applications are in C would have difficulty using a python library to provide UPnP services. Being in python also has performance implications, however in this instance they are not as important as there are larger performance bottlenecks in the network for most UPnP functionality.

Finally, BRisa is not based around the GObject run time system. By not integrating with these Gnome libraries BRisa does a lot of unnecessary work and has extra code. BRisa cannot take advantage of many of GObject's benefits and in particular is harder to integrate into other GObject based applications.

3.3 Coherence

Coherence is very similar to BRisa. It is a pure-python implementation of the UPnP protocols [4]. Like BRisa coherence also supports 'modern' features like Flikr and Youtube integration. Coherence, unlike BRisa, is targeted at desktop applications. Coherence has an interesting feature that sets it apart from other UPnP back ends, Pont Mirabeau (translated: Mirabeau Bridge). Mirabeau is a media server, based on Coherence, that allows for two distant UPnP networks to connect via XMPP (the Jabber protocol [5]). This is one of the most advanced uses of UPnP. Coherence also integrates with a commonly used technology on Linux known as DBus. DBus allows for different applications on the same device to easily communicate and ask each other to do things.

Coherence has not yet been demonstrated to be practical on mobile devices and suffers from many of the same disadvantages of BRisa (lack of GObject integration in particular) and as such it was not chosen as the foundation for the work herein.

3.4 libupnp

libupnp is a UPnP library written in the low-level C language [6]. libupnp was one of the first UPnP frameworks for Linux and as such it gets credit for helping to pioneer a software field. libupnp also gets credit for supporting a wide variety of platforms: Linux, Free-BSD and Solaris. However, as many software vanguards tend to, it suffers from architectural deficits, having not had previous generations of software to act as a guide. As such libupnp suffers from some severe complexities. GUPnP author Zeeshan Ali writes of using libupnp: "[...] frustration with libupnp and its mess of threads". Libupnp is used by a variety of applications, however, recently more modern implementations that are easier to use have come into use and development of libupnp has slowed considerably.

3.5 GUPnP

GUPnP is a UPnP framework implementation written in C. GUPnP is very modern in the sense that it is not written from the bottom up, instead building on a host of technologies commonly available in the Gnome Linux environment including GObject and libsoup [15]. GUPnP is targeted at Maemo (Maemo supports GObject and libsoup) and is currently an integral part of the Maemo platform; the UPnP capabilities that are built into the device use GUPnP. As of now, however, GUPnP does not have any support for a high level scripting language like Python.

3.6 GUPnP - Vala

GUPnP and libupnp are the only two technologies mentioned thus far which are written in C. Libupnp does not at present have bindings to any other languages. GUPnP, however, courtesy of it being built on technologies such as

GObject is capable of adding bindings to a number of different languages. Currently there exists bindings for only one additional language, Vala. (Vala is a language much like C# but modified to be more compatible with GObject type paradigms. Vala compiles into C which is then compiled by a normal compiler such as gcc [13]). The GUPnP-Vala bindings are thus far are only in use by one application, Rygel. Rygel is a UPnP media server and renderer based on GUPnP-Vala and is under active development [1].

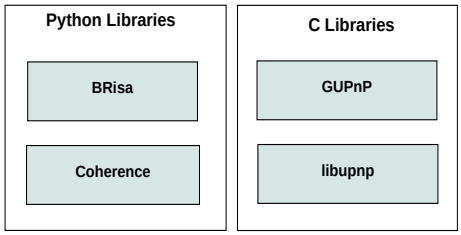

Figure 2: Overview of available UPnP libraries

4. SYSTEM MODEL

To achieve the desired goal of building a platform for easy development and prototyping of applications in Python using GUPnP it is necessary to do several things. The first is to integrate the needed technologies together to make GUPnP usable in Python (herein: Aeryn). One must then develop several applications which demonstrate the efficacy of Aeryn and serves as a motivator for further Aeryn development. Applications which act as all three UPnP device profiles (renderer, server and control point) as well as several "modern bridge" applications will be demonstrated, namely: Zhaan, a UPnP Control Point, MPD-UPnP a media server and renderer for the Music Playing Daemon, Pandora-UPnP, a media server based on Pandora Internet Radio content and Auto Remote, a location aware UPnP macro service.

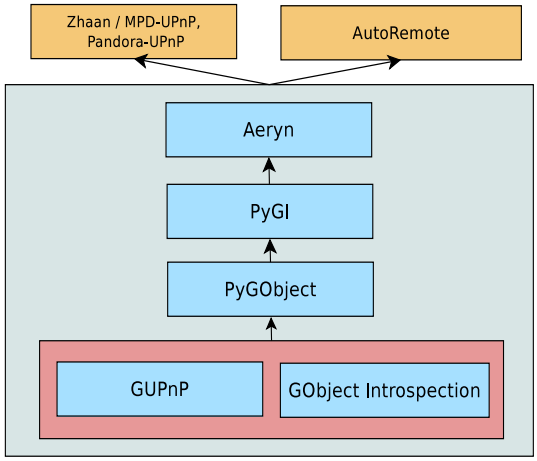

Figure 3: The software stack, including the building blocks for Aeryn

4.1 Aeryn

To achieve the goal of success in an embedded scenario the bindings must have a small memory footprint. Therefore a runtime binding technology, GObject introspection (herein: GI) is used. GI will enable the ability to rapidly generate runtime bindings using another technology known as PyGI. GI also has the enormous benefit of flexibility. Runtime bindings generated with GI can be used with any GI middleware (such as PyGI for python, or Seed/GJs for javascript) and hence with minimal effort the work on Aeryn could also be applied to javascript or other languages.

PyGI does not support a programming language technique called function callbacks. Callback support is essential to Aeryn. (*e.g.* One could pass a python function to the GUPnP library and have GUPnP call the python function when an action has been performed on the network without GUPnP knowing that the function was any different from a normal C function.) UPnP is heavily based on asynchronous calls and hence function callbacks. Therefore implementing callbacks falls into scope of this research.

4.2 Location Aware UPnP Macro Service: AutoRemote

AutoRemote depends on retrieving data from multiple location sources (both wifi and GPS) as well as the ability to access UPnP. Wifi capability and awareness can be determined using the free and open source module python-wifi. GPS data on Maemo can be provided from the python-location module, a set of static bindings to the Maemo Location APIs provided by Nokia.

The application itself is a glorified UPnP control point. The only difference is that the control point executes actions when certain conditions are met, as opposed to in response to direct user input.

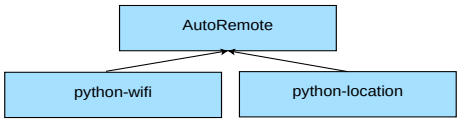

Figure 4: AutoRemote Dependencies

4.3 Zhaan

Maemo at present does not have a viable UPnP Control Point application. Therefore it would be very useful and a great demonstration to the developer community if such an application where created. Zhaan is capable of controlling the entire process of consuming media over UPnP, from selecting the source and renderer to renderer playback control and status display to the user.

4.4 MPD-UPnP

MPD, the Music Playing Daemon, is a media serving daemon for Linux computers. Effectively it provides an API for applications to control its output of music. MPD-UPnP will implement the UPnP API using the MPD API; effectively exposing MPD as a fully capable media server and renderer to any other UPnP devices on the network.

This has the benefit of allowing a whole new class of applications to control MPD. In addition, classically an application would need to know the details of how to communicate with MPD in order to use the API (the hostname, port,

etc.). UPnP provides a discovery protocol enabling devices to more easily connect with MPD than by standard means.

4.5 Pandora-UPnP

Pandora is an internet radio service that is controlled via Flash technology in the web browser. Pandora provides users with custom media selections based on the user's inputted preferences. Controlling Pandora is somewhat onderous as it requires the web browser, Flash technology and the use of a mouse.

There exists a daemon application which can be controlled via Linux FIFO objects. Similar to MPD-UPnP one can implement the UPnP API ontop of this FIFO API, exposing Pandora control over UPnP for applictions like Zhaan to consume.

5. SYSTEM IMPLEMENTATION

5.1 Aeryn

The GUPnP source code will be annotated with markings indicating certain features of the code that GI could not statically detect. With the annotations an intermediate "gir" (GI repository) file is generated which contains metadata for all of the exported symbols of the library. This data is used by the run time bindings generator (PyGI) to properly generate the necessary library calls.

Implementing callbacks in PyGI is a substantial problem. The major concerns are that one cannot simply invoke a python function pointer using normal C calling conventions. Instead one has to call a CPython API call such as *PyObject_CallFunction*. To handle this difficulty the library called libffi is used to generate a new method which is used in place of the real python function. That method takes the arguments that the C library calls and in turn passes them, as well as some additional data, to a PyGI function which can properly make the python call. PyGI must then properly convert arguments and return values and of course call the callback.

5.2 AutoRemote

A user interface will be implemented which allows the user to define actions in a precise matter. This is possible as there are several discreet things one might wish to do with UPnP such as starting and stopping media playback. There will not be a large amount of data so no formal database is required to store the users macro settings; instead a simple json (javascript object notation, similar to XML) encoded data file can be used for convenience.

The service will provide a long running daemon which reads in the json data file and listens to python-wifi and python-location for location changes. It will be responsible for executing macros as appropriate based on these location changes and the json data.

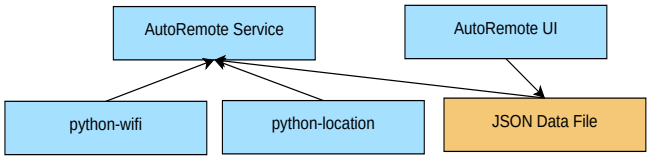

Figure 5: AutoRemote System Architecture

The long running daemon inside AutoRemote is programmed using a model of the world, called a WorldData to effectively determine when macros should be executed. A WorldData is a list of historical WorldStates. A WorldState represents a given instant in time. (WorldStates are captured every 5 seconds within AutoRemote.) At a capture point a set of physics rules (*e.g.* a clock for time, location information from wifi) define the new WorldState.

The daemon also defines the notion of a trigger. A trigger is a rule based on the available laws of physics (time, location, *etc.*) which, when it passes, executes a UPnP action. The triggers are all managed by a TriggerMaster which is responsible for keeping track of triggers and feeding them the current WorldData so each can determine whether it should fire.

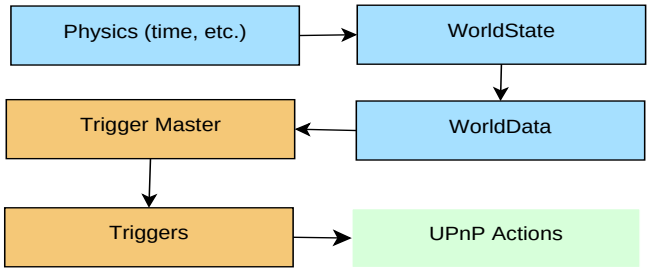

Figure 6: AutoRemote Daemon Architecture

5.3 Zhaan

Zhaan's implementation is divided into two components. There is a strictly UPnP based component which is responsible for communicating with the other devices on the network and the user interface component. The UI is further subdivided into two more components, the GTK UI which will be used to control Zhaan from a Linux desktop computer, and the Hildin UI which will be shown to users of Zhaan on Maemo devices.

5.4 MPD-UPnP

There is pre-existing work on controlling MPD from python using MPD's native HTTP based API in a module called Python-MPD. MPD-UPnP's job is to expose the standard UPnP MediaServer and MediaRenderer class functions and in turn call the appropriate MPD api calls.

To manage this MPD-UPnP needs to storage a cache of a significant amount of MPD's meta-data in order to properly expose the data in the UPnP hierarchy. This cache is periodically updated in a thread so as to not interrupt regular network operations.

The traditional MPD usage paradigm is such that there is one database of media and MPD controls playback of that database on one local machine. UPnP, however, allows for playback in many places using media from also many sources. Therefore it is possible to extend MPD's basic capabilities and allow it to perform four key functions, only two of which are possible using Non-UPnP based MPD clients. All four functionalities are implemented.

- (Standard Feature) MPD as a Media Renderer of content that MPD already knows about

- (Standard Feature) MPD as a Media Server of content MPD already knows about that only MPD can consume
- (New functionality) MPD as a Media Renderer of arbitrary UPnP-exposed audio content
- (New functionality) MPD as a Media Server of its own data to other peers.

5.5 Pandora-UPnP

Pianobar is a pre-existing application which can play Pandora internet radio. Pianobar runs only at the command line or as a background process and exposes an interface via a Linux FIFO file descriptor. Pandora-UPnP will use the pyPandora pianobar bindings (also written as part of this work) to interface with pianobar. Similar to MPD there are four possible modes of operation one could concieve of for Pandora-UPnP. Due to limitations of Pianobar, however, only one is implemented.

- (Standard Feature) Pandora as a Media Renderer of content that Pandora already knows about. (This includes controlling the current radio station, as well as pause/play/stop/next functionality.)

6. SYSTEM PERFORMANCE AND RESULTS

6.1 Aeryn

Quantitative benchmark performance is not a concern with the bindings for GUPnP-Python. The primary driver of performance is the network over which GUPnP operates. Any additional computational overhead from the bindings, unless extraordinary, are swamped by network latency. Performance instead can be better benchmarked by how complete the bindings are and how well they work. To this end a unit test suite has been created which tests the Aeryn API. The test suite passes and has approximately 85% coverage of the API. (*Note:* Memory performance is a major benchmark for PyGi itself, but is not effected by Aeryn work.)

6.2 Location Based UPnP Macro Service (AutoRemote)

Autoremote, similar to Aeryn, does not have valid quntitative performance metrics. Qualitative metrics, such as usability and functionality, provide a much more accurate measure of AutoRemote's success. AutoRemote, both the user interface and the long running service/daemon are functional.

6.3 Zhaan

Zhaan performance can be measured by functionality. Zhaan itself should be able to identify any UPnP media renderer or server on the network. From there one should be able to access all of the functionality Zhaan is built for that these devices support. (For example, Zhaan supports volume control and hence controlling volume should work on all devices on the network which support volume controls).

As can be seen in figures 8, 9, 10 and 11 Zhaan is working and fully function both on the linux desktop using GTK to draw windows and on the Maemo desktop (on the Nokia N900) using the Hildon library for screen drawing. Zhaan has been publically released and seen a fair amount of interest in the general open source and Maemo communities.

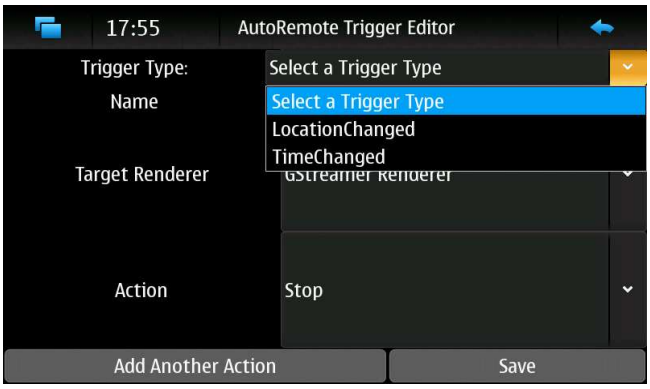

Figure 7: AutoRemote Macro Definition Page

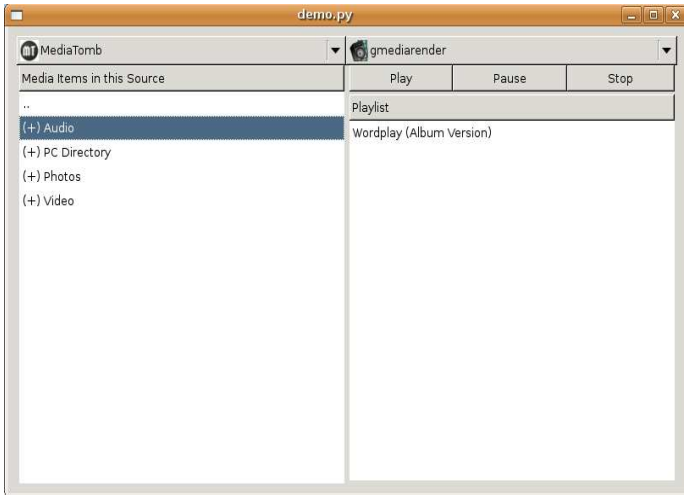

Figure 8: Zhaan GTK Demo

6.4 MPD-UPnP

MPD-UPnP was initially implemented in about 50 lines of Python code. This is a strong testament to the functionality and developer friendlyness of the Aeyrn platform. Since then the three other major pieces of functionality were added to MPD-UPnP and it has grown significantly from the original 50 lines. MPD-UPnP is fully controllable via Zhaan and all four critical components of functionality work as expected.

6.5 Pandora-UPnP

Pandora-UPnP, similar to MPD-UPnP was implemented in a remarkably small library. The completed version of Pandora-UPnP with all the features required to entirely satisfy its main goal is 190 lines of Python code.

7. FUTURE WORK

This work has begun to attract interest in the general Maemo and GI communities. Anderson Lizardo, a Nokia employee, has approached the project and offered his assistance in porting PyGI and my work on GUPnP to the Maemo platform (a not so technically interesting and tedious task involving mostly packaging logistics). Anderson

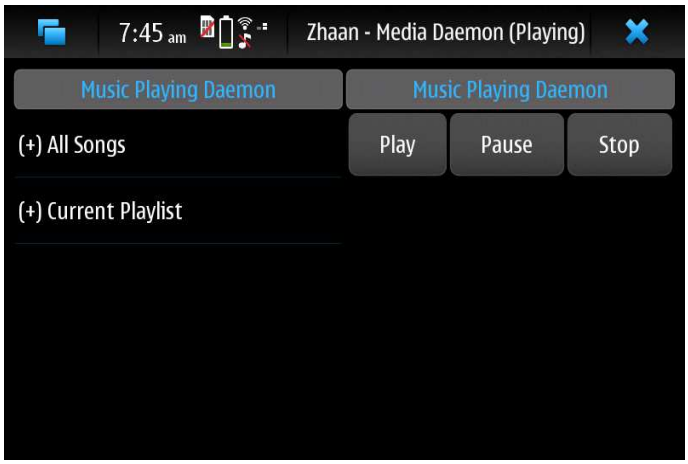

Figure 9: Zhaan (Hildon) Browsing A Media Source (top level)

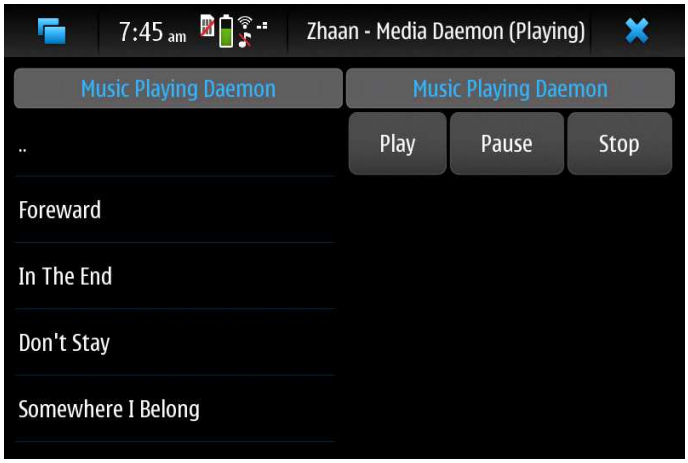

Figure 10: Zhaan (Hildon) Browsing a Media Source

Lizardo and the PyMaemo team have successfully completed this task and the Aeryn platform now works well on the Nokia N900 and other Maemo devices.

Aeryn itself can be considered a completed project. GUPnP now works in Python on the Maemo platform. There is more work however which can be done on the PyGi bindings, as well as further applications developed and matured ontop of the platform.

8. CONCLUSION

The project blog[7] has seen significant attention as a result of the Aeryn platform. Posts on the blog have been syndicated and references by many other members of the community and interest has begun to brew over the ability to use Aeryn in mobile settings.

The project has been a fantastic success in porting new applications and functionality to the Maemo platform and efforts are in motion to continue this trend and improve the home media experience even further.

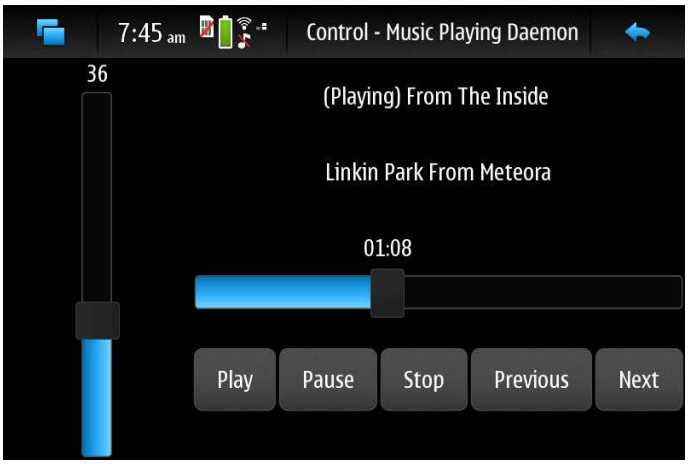

Figure 11: Zhaan (Hildon) Control View

9. REFERENCES

[1] Zeeshan Ali, Jens George, Thijs Vermeir, and James Henstridge. Rygel. `http://live.gnome.org/Rygel`.

[2] Maemo Community and Nokia. Maemo.org licenses. `http://maemo.org/legal/terms_of_use/licenses/`.

[3] Open Source Community. Brisa project. `http://BRisa.garage.maemo.org/`.

[4] Open Source Community. Coherence - a dlna/upnp framework for the digital living. `http://coherence.beebits.net/`.

[5] Open Source Community. Pont mirabeau - an upnp bridge. `http://netzflocken.de/2009/6/30/pont-mirabeau-an-upnp-bridge`.

[6] Intel Corp and Open Source Communitie. Portable sdk for upnp devices. `http://pupnp.sourceforge.net/`.

[7] Zachary Goldberg. Bluesata - zachary goldberg's blog. `http://www.zachgoldberg.com`.

[8] Zachary Goldberg. Github - autoremote. `http://www.github.com/ZachGoldberg/AutoRemote`.

[9] Zachary Goldberg. Github - mpd-upnp. `http://www.github.com/ZachGoldberg/MPD-UPnP`.

[10] Zachary Goldberg. Github - pandora-upnp. `http://www.github.com/ZachGoldberg/Pandora-UPnP`.

[11] Zachary Goldberg. Github - zhaan. `http://www.github.com/ZachGoldberg/Zhaan`.

[12] A.L.V. Guedes, D.F.S. Santos, J.L. Nascimento, L.M. Sales, A. Perkusich, and H.O. Almeida. Brisa upnp a/v framework. *Consumer Electronics*, ICCE 2008:1–2, 2008.

[13] Raffaele Sandrini Jürg Billeter. Vala - compiler for the gobject type system. `http://live.gnome.org/Vala`.

[14] Wei-Shun Liao, Yen-Ju Huang, and Chih-Lin Hu. Mobile media content sharing in upnp-based home network environment. *SAINT*, 77:52–52, 2007.

[15] OpenHanded and Zeeshan Ali. Gupnp framework. `http://www.GUPnP.org/`.

[16] Universal Plug and Play Forum. Upnp specification. `http://www.UPnP.org/resources/specifications.asp`.

[17] PyGi. Gnome/python 2010 hackfest. `http://live.gnome.org/Hackfests/Python2010`.

10. APPENDIX

10.1 Source Code Repositories

All code produced under this project (several thousand lines of Python and tens of patches to C code projects GUPnP and PyGI) is publically available online and under the GNU LGPLv3 License. Applications Zhaan, MPD-UPnP, AutoRemote and Pandora-UPnP can be found on online hosting site GitHub [11], [9], [8], [10].

10.2 Author's Note on Open Development

From the beggining this project was part of an open source and free software ecosystem. All of the code was developed collaboratively with other people volunteering their time to the same projects from all around the world. Words cannot express what a good experience it is to have this sense of community with people you've never met in countries you've never been to whose national language you can't understand a word of. The people I had a chance to work with were relentlessly supportive and inspiring to me. I owe a deep sense of gratitude to the following people, and many others: Zeeshan Ali, Simon van der Linden, Tomeu Vizoso, Anderson Lizardo, John Palmieri, Colin Walters, David Malcom, John Ehresman.

As part of this project I had the opportunity to travel to Amsterdam for the Maemo Summit, where I got to experience firsthand the Maemo community and all of the wonderful people and projects it has to offer. I was also able to participate in the Gnome/Python Hackfest 2010 [17] in Boston and meet many of the PyGi developers. As a result of that project I am now an official maintainer of the PyGi project and will continue to contribute long after this project is complete.

Maintaining a project blog has also been invaluable for me. I have, as a direct result of my work on Zhaan et. al, and its publicity on my blog, been contacted by two different companies including Sonos Inc. and Amazon Inc. and have been asked if I were interested in a job interview. I highly encourage future senior design projects to maintain an open nature and an open blog about their work.