Connectivity

Contents

Network management is the task of managing the access to networks. In other words, deciding when and through which means to connect to the internet. In an IVI context this task is affected by several conflicting requirements. Connectivity may be spotty at times, with tunnels, high speed causing WiFi networks to come and go quickly, low cell phone signal strength, and so on. On the other hand, potentially good connectivity while parked, since the user might have high quality WiFi at the office and at home. Network Management will be discussed in Network management.

Online and cellular-based real-time communications, including chatting, voice calls, VoIP and video calls are covered in Real-time communications.

It is very common these days to have people carrying one or more smart devices with them. People want those smart devices to connect to their in-vehicle infotainment system for playing audio, importing contacts and also use or share Internet connections. This is discussed in Tethering from mobile devices.

The main medium used for inter-device communication, Bluetooth, and its var-

ious profiles are discussed in Bluetooth support. A brief discussion of using GPS to enhance network management and about the GeoClue framework are the subject of Global Positioning System (GPS).

Contacts management is covered by a separate document. Integration with other devices by means other than Bluetooth and USB mass-storage, such as reading songs off of an iPod is the topic discussed in Media downloading.

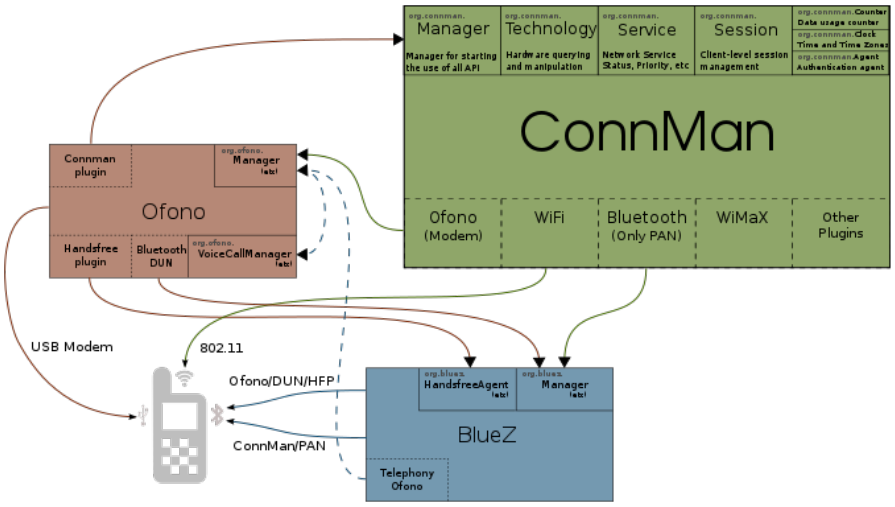
Network management

The main goals of network management in an IVI system are to make sure the best connection is being used at all times while providing enough information to applications so that they can apply reasonable policies. For example, the IVI system should be able to fall-back to a metered 3G connection when an active WiFi connection is lost (because, say, the user drives their car out of their garage). In addition, big downloads should be paused in such a case; these would only be resumed when on an unmetered connection, to avoid significant charges on the user's phone bill.

ConnMan[1] is the central piece of the network management system being considered for Apertis. It is focused on mobile use cases, provides good flexibility and features that allow implementation of the use cases mentioned above. oFono[2] is the de-facto standard when it comes to cellular connections and related features, and it is able to work in cooperation with ConnMan.

To complete the functionality expected from a modern network management framework, Bluetooth integration is also important. BlueZ[3] is used to provide that integration, allowing ConnMan to use Bluetooth devices to go online. Illustration has a schematic view of the interactions among these frameworks.

[1] https://git.kernel.org/pub/scm/network/connman/connman.git/
[2] http://oFono.org/
[3] http://www.bluez.org/



Switching to a different connection

There are very specific requirements about how and when the system should switch from an active Internet connection to a newly-available one. ConnMan developers are working in a infrastructure for policy plugins. Collabora believes this new infrastructure can be used to implement the policies required to satisfy the requirements.

The two main concerns are that the system should not switch to a WiFi network that just became available, since that may just be an open network in a café the car is passing by, but that it should also take advantage of good known connections when they are available.

ConnMan provides several facilities to gather information useful for such policy decisions. For instance, a network that has been manually selected by the user will have the Favorite property set to true.

That can be used to implement a policy of never automatically migrating to open WiFi networks that are detected, unless it has been successfully used before. This would guarantee that the system is able to switch automatically to relevant networks without running into the problem of trying to associate with the every open network it passes by.

Because connections may be lost or replaced by a better connection by ConnMan, applications need to be aware that their session may go away at any time, and be able to recover from that. When a connection change happens, ConnMan will emit a D-Bus signal, and applications may need to drop connections they have started and restart whatever they were doing.

A concrete example would be the email application that is connected to an IMAPx server; when a connection change happens, the application gets notified

the connection it was using has gone away, so it drops all connections it had with the IMAPx server. If the new connection satisfies the requirements specified by the email client on its ConnMan session, it gets a "now online"notification and reconnects to the server.

Application requirements and expressing preferences

One very important characteristic of Apertis is that its Internet connectivity will vary a lot –going from a high speed WiFi network to a slow, unreliable, metered GSM connection and back is a common scenario, as discussed in the previous section. It may also be required that a particular type of connection be established to accommodate the needs of some applications.

ConnMan has a feature called Sessions. What that feature provides is a way for applications to tell ConnMan what they expect from an Internet connection, and get from ConnMan a network connection status that is relative to those requirements.

For instance, an application that downloads podcasts may have a policy that it would only perform the downloads when on WiFi. This application would create a session with ConnMan, and specify settings that ConnMan uses to decide whether that session is to be considered online or not.

The main settings are the **AllowedBearers** and **ConnectionType**. The first of these specifies which kinds of connections are allowed for the type of traffic this application intends to do. It is a simple list that specifies the preferred connection methods, such as, [**cellular, wifi, bluetooth**], which would specify a preference of cellular connection over both WiFi and Bluetooth .

A special "*****" bearer can be used to specify all bearers, which makes it easy to specify preference for one over all others, which will be treated as equivalent. When one of the connections allowed for an application comes online, the sessions is declared to be online. When a change happens on a Session setting ConnMan updates the application with the new values for the changed settings.

The second, **ConnectionType**, is used by the application to tell ConnMan if a connection wants to be online or if local connectivity is enough. Local connectivity means only connectivity in the internal network is needed, for example, an application may want to exchange data with other devices inside the same network. There is not much use for this setting in Apertis.

An application can have more than one ConnMan session at the same time, allowing applications to specify multiple policies and preferences, and perform work according to what is actually available. In addition to these three settings discussed here, ConnMan also provides several settings that can be used to customize how both sessions and the system deal with networking[4].

[4] https://git.kernel.org/pub/scm/network/connman/connman.git/tree/doc/session-api.txt?id=HEAD

Note that the Session API is still in a experimental state and it implementation and API are changing rapidly. This means both that it cannot be considered a stable part of the API supported by Apertis and that very few existing applications use it's current form. This should not be a problem for Apertis since applications are not intended to use ConnMan directly, so a wrapper API can be specified for the SDK.

Binding to the appropriate network interface

ConnMan allows multiple connections to exist at the same time. This might be useful for various reasons but it also brings some complications with it. First of all, if an application wants to use a specific connection it needs to explicitly bind its network usage to the desired network interface.

However, binding to a specific interface requires the NET_RAW capability[5] that is not something that regular applications should be allowed to have. A possible solution would be to also delegate this binding to special application that has the privileges to do such binding. The viability of such a solution needs to be properly investigated during the initial development of the feature.

Also, keeping in mind the desire to take complexity and control away from applications it seems desirable to abstract this complexity away to the SDK. The SDK can provide APIs that wrap ConnMan functionality and handle binding for the application. This means more Apertis-specific code, however, meaning less code reuse for existing applications.

Connections policies and store applications manifests

As discussed above, some control can be exerted on how ConnMan ranks and chooses connections by having applications (or a system service on their behalf) provide ConnMan with a list of their requirements using the Session APIs.

It has been made clear that applications from the store should be specifying their needs as much as possible through the manifest file that will be distributed along with applications on the app store. For network management this means specifying the allowed bearers, mainly.

The policy plugin mentioned above could use information provided by the application manager and application manifest files to decide on what the best policy to implement is. This would not require changing applications, but limits the flexibility the developer has to work with.

Network-related events and how applications need to behave

For applications that are written to work with ConnMan, two signals are essential: connection is up, connection is down. When a connection comes up

[5] http://git.kernel.org/?p=linux/kernel/git/next/linux-next.git;a=blob;f=net/core/sock.c;h=b374899aecb6ea3a8590ae9ccdbb3e60225561d4;hb=HEAD#l470

the application takes the appropriate steps to start whatever its functionality is. An IMAP mail client would at this point connect to the IMAP server, and look for new messages, a podcast downloader would look for new podcasts to start downloading or resume any downloads that had previously been started, and so on.

When the connection goes down –even if it's just being switched from one connection method to another –any existing IP connections would not work any more, since the IP address will have changed. The application needs to close any connections. This means an IMAP mail client would close the sockets it had open with the IMAP server, a podcast downloader will close the HTTP or FTP connections, and so on. The connections can be re-established/resumed in case a new notification comes in that the system is online once more.

The following is a potential list of applications and events they will be interested in handling. As will be seen the events an application needs to handle are essentially limited to having a connection and not having a connection any more.

Email client

- Connected event
  - Connect to IMAP server and check for new mail; note that IMAP connections are usually kept alive to receive notifications of new email from the server
  - Connect to the SMTP server to send any emails stored in the outgoing mail box
- Disconnected event
  - Drop IMAP connections
  - Cancel ongoing loading of email messages
  - Cancel ongoing sending of email messages, making sure they stay in the outgoing mail box

Media player

- Connected event
  - In case multiple connections are supported, when a faster connection appears switch to it of needed.
  - If media is being played and previously disconnected, resume buffering
- Disconnected event
  - Drop connections

Feed reader

- Connected event
  - Begin download of the latest entries
  - If it's a fast connection, begin pre-caching of images and other big feed attachments
- Disconnected event
  - Drop connections

Continuing downloads The HTTP protocol provides clients and servers with the ability of picking up a transfer from a given point, so that partially downloaded content does not need to be re-downloaded in full when a connection is dropped and reconnected. Details about how the protocol supports partial downloads can be found in RFC2616[6].

In summary, when picking up a download the client should send a *Range* header specifying the bytes it wants to download. If the server supports continuing downloads and the range is acceptable a ***206 Partial Content*** response will be sent instead of the usual ***200 OK*** one. The client can then append the data to the partially downloaded file. If the server does not reply with a ***206*** response, then the file needs to be truncated, since the download will be starting from scratch.

Connectivity policies on Android

Connectivity policies on Android are a way more simpler. The Android system does not implement any per application configuration on how application should access the internet (wifi, 3g, etc.).

Also Android does not have any mechanism to notify the applications that the system is online, the applications just get a notification about a Network State Change and then they have to figure out by themselves if the system is online by requesting a "route to host".

One of the few network configurations android has is to enable/disable Wi-Fi, mobile data and roaming allowance globally. Apart from that the user can also restrict background data usage for each applications in the Global Settings.

Real-time communications

The Apertis needs to be well-connected with Internet communication services such as Google Talk and Skype. Telepathy[7] provides a framework for enabling messaging, video and audio calling through many of the existing services, and more can be supported by developing custom connection managers[8].

[6] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.35
[7] http://telepathy.freedesktop.org/wiki/
[8] http://telepathy.freedesktop.org/doc/book/sect.connection.connection-manager.html

Telepathy provides a D-Bus API which abstracts away the specific connection managers allowing a UI to seamlessly support various protocols while only having little or no protocol specific knowledge. The fact that Telepathy is implemented as several D-Bus services makes it possible to integrate messaging and voice features throughout the system.

A good example of these are the chat, dialler and address book applications present on the Nokia N900 and N9 devices[9], which use Telepathy to support messaging, GSM and VoIP Calls using one single user interface, while at the same time providing ways for the user to choose which protocol they want for a given conversation.

Existing open-source connection managers support messaging through Jabber, GTalk, Facebook, Live Messenger, and more. Audio and video calls are also supported over Jabber, GTalk and SIP. See the Telepathy site[10] for more details. Before deciding on shipping any of these, however, it's important to verify whether any legal issues may arise, mainly related to trademarks.

As discussed before, Telepathy is pluggable, enabling a mix of closed and open-source connection managers to coexist. This enables OEMs to enable as many third-party services as desired, requiring for the technical side at most the creation of a new connection manager.

Collabora has been involved in consultancy projects to integrate various proprietary backends in the past and is ready to do so again if it is decided to include support for more protocols. More details about this will be included in reference produced during the development phase by the documentation team.

Traditional Telephony (GSM, SMS)

The system shall support making and receiving calls and sending/receiving text messages through a paired cell phone. Telepathy has a backend on top of oFono to make calls and send text messages, which makes it possible to easily have a single, integrated user interface for both regular phone calls and messages along with those of online services.

Telepathy is focused on messaging and calling; as such, it does not include GSM/UMTS-specific functionality like signal-strength, data connections, and so on. Those features are accessible through oFono directly.

Tethering from mobile devices

There are six main ways to hop on to a mobile device's Internet connection:

- WiFi connection for devices that support mobile hotspot

[9] http://techprolonged.com/index.php/2011/11/12/nokia-n9-a-complete-walk-through-meego-harmattan-software-and-user-interface-experience/#contacts-calling

[10] https://telepathy.freedesktop.org/components/

- Using the DUN Bluetooth profile, through oFono
- Using the PAN Bluetooth profile
- Using Ethernet over USB
- Using 3G USB modem
- Using device-specific proprietary protocols

A mobile hotspot feature is becoming more common on mobile devices and, in a way, taking the place once occupied by Bluetooth for tethering. It is a good connection method because it is very simple to set up.

The PAN profile is supported by ConnMan through BlueZ and DUN also needs these two components to works plus a extra one, which is oFono. This difference is due to the fact that the DUN profiles behaves like a modem and thus needs oFono to handle it. All interactions between BlueZ and the two other daemons, ConnMan and oFono, are performed using BlueZ's D-Bus interface, and as such should not cause problems in case is planned to replace BlueZ with a proprietary counterpart that implements the same interfaces.

Note that connecting a phone to the car for tethering over Bluetooth is a process that requires user intervention: the user needs to first pair the two devices. In most systems that support connections over the cell phone network the user is also asked to choose the plan they acquired from their provider from a list, which will also need to be done for the Apertis; only then the connection will be made available through ConnMan.

Ethernet over USB is supported by Linux using the usbnet driver. Among device-specific protocols, Apple devices in particular are important. Linux includes, since version 2.6.34, the ipheth[11] driver, which enables using Ethernet over the USB connection for Apple devices. In addition to the driver, pairing of the device by a user-space program is required. That pairing can be performed either by using the standalone tool provided at the project's web page or through the tools distributed by the **libimobiledevice** project, discussed in Media downloading.

Collabora believes the three main components discussed here, BlueZ, oFono and ConnMan are capable of supporting tethering to most mobile devices. Provided appropriate user interfaces are implemented, ConnMan is able to provide all requirements regarding having several different phones in the car, including prioritizing and selecting which one should be used.

Counting bytes and getting information about bandwidth

ConnMan provides an API called **Counters** that is used for tracking how much traffic has gone through a given connection, and can be used by the network connections management UI to inform the user about the quantity of data that

[11] https://github.com/torvalds/linux/blob/master/drivers/net/usb/ipheth.c

has been transmitted. The counters are per-connection and are automatically updated with the information by ConnMan.

oFono also provides the **CallMeter** API for tracking how much conversation time is still available for a GSM phone, using data from the SIM. oFono is able to emit a warning when the limits are close to be reached.

For measuring bandwidth there is no convenient API at the moment. Clients can register a counter and specify an update interval, but ConnMan advises against using that API for tracking time. A more robust and correct implementation would be to have applications and services that care about that information track the RX/TX bytes and run a timer of their own to estimate how much bandwidth is being used at a given point in time.

It is important to note that tracking connection quality taking in account used bandwidth (and possible other variables as connection latency and available bandwidth) is not a easy task. Usually those variables doesn't give enough information to decide which connection has the better quality.

Providing Internet connectivity to other devices

In case the Apertis has Internet connectivity itself, it should be able to share it with other devices through either Bluetooth or WiFi.

ConnMan supports sharing the current Internet connection by using the WiFi interface in master mode or via Bluetooth PAN profile, becoming an access point that other devices can connect to. This is done by turning on tethering mode on WiFi or Bluetooth.

> See the tethering properties at the bottom of https://git.kernel.org/pub/scm/network/connman/connman.git/tree/doc/technology-api.txt?id=HEAD

As is the case with other features, this needs proper UI to be created to let the user turn the tethering on as well as specify the desired SSID and pass-phrase for WiFi, or to pair the Bluetooth devices. In order for this feature to be provided, the driver for the wireless chip used in the development board needs to support the master mode.

A web server to provide information

Apertis will have a web server running internally to provide information about the system and the car for access by smart phones. Collabora's working assumption is the server will be available to devices that connect to the WiFi or Bluetooth hotspot provided by the system, regardless of whether it is being used to provide Internet connectivity to the devices or not. Libwebsockets can be used to write a solution for web server.

For users to access the web server, the manual of the device will contain a

specific URI, and the DNS server provided by the device will resolve the name to the address the system has in the address space used by its DHCP.

Bluetooth support

The automotive space is by far the biggest user of Bluetooth for communications between the car and external devices, such as phones, tablets, notebooks, and so on. Plans have been stated to acquire a proprietary solution and supplement BlueZ in the apertis.

That solution sits in between applications that use BlueZ and the BlueZ daemon, and adapts requests to make sure specific device quirks are satisfied.

BlueZ is currently a fairly complete Bluetooth stack, and has support for all of the major Bluetooth profiles[12]. It's important to note, however, that applications need to be written to use the Bluetooth infrastructure for connecting to mobile devices for music playing, remote control, file transfer, downloading of contacts and tethering.

For other Bluetooth profiles, the ones supported by BlueZ will be provided, development of support for more profiles is out of scope for this project. The list of Bluetooth profiles bellow was extracted from the Apertis Feature List document, and information is provided on the general level of support provided by BlueZ. For a more detailed list of existing support and gaps, Collabora would require a more detailed list of requirements.

When it comes to pairing support BlueZ supports both Legacy Pairing (PIN entry, many old devices only support this type of pairing) and Secure Simple Pairing (Numeric Comparison).

Bluetooth 3.0, 4.0, High Speed, L2CAP, SDP, RFCOMM

BlueZ currently supports the both Bluetooth 3.0 and 4.0 core specification. However, High Speed support through 802.11 AMP is still under development, so it is not currently supported. The Bluetooth core support provided by BlueZ includes Logical Link and Control Adaptation Protocol **(L2CAP),** Service Discovery Protocol **(SDP)** and Radio Frequency Communications **(RFCOMM).**

SPP

The Serial Port Profile **(SPP)** 1.1 is supported by BlueZ. The Serial Port Profile allows emulation of serial ports over a Bluetooth link.

PAN and DUN

As discussed in sections Network management and Tethering from mobile devices, BlueZ provides support for the Personal Area Networking **(PAN)** profile

[12] http://www.bluez.org/profiles/

both in the NAP role (BlueZ acting as connection provider) or PANU role (BlueZ using a internet connection over Bluetooth).

There is support for the DUN profile. The Client role is implemented by an extra oFono's daemon and can be used to connect to devices providing internet connection.

There is also support for the server role of the Dial-up Networking **(DUN)** profiles, which can be used with oFono and ConnMan to provide Internet connection to an external device. However current implementation only supports sharing a DUN connection only if the device has a GPRS data connection active. Collabora thinks that lack the support for DUN server won't be a problem. DUN is rapidly being replaces by the PAN profile.

GOEP, OBEX

The Generic Object Exchange Profile **(GOEP)** and Object Exchange Protocol **(OBEX)** are also supported by BlueZ. They enable file exchange between Bluetooth-capable devices and the Apertis system.

PBAP, MAP, OPP, SYNCH

The Phone Book Access Profile **(PBAP)** is supported by BlueZ, and can be used for downloading contacts from the external devices. There is also support for Object Push Profile **(OPP)**, used for transferring vCards and vCalendars. Finally, BlueZ also supports the 1.0 version of the Message Access Profile **(MAP)** version 1.0 in the client role**,** which can be used to download SMS messages and email from a phone onto the Apertis system, however support to upload delete and mark messages as read/unread is lacking at the moment.

Note that, although BlueZ includes support for these profiles, it's up to applications on the system to make use of the framework to provide the actual features. For instance, the contacts application needs to talk to BlueZ to perform the phone book download.

There is currently no support for the Synchronization Profile **(SYNCH)** profile. Collabora's current understanding is this does not pose a problem for the use cases planned, since only support for download of the contacts is required.

For more information on the specific use-cases, problems and proposed solutions regarding contacts in the Apertis system refer to the Contacts design document prepared by Collabora.

AVRCP, A2DP, VDP

These profiles are used to communicate with devices that are able to reproduce multimedia content and/or control media playing remotely.

BlueZ supports the Audio/Video Remote Control Profile 1.0 **(AVRCP)** in the controller role, but it only supports the two commands at the moment: Volume

Down and Volume Up. Collabora recommends the development of the missing features for AVRCP 1.0 version and also support for the 1.4 version, which is not yet supported by upstream. This would provide metadata information about the media and folder browsing support.

When acting as a controller, an application needs to provide the user with an interface for inputting commands. Collabora will provide sample code for an application acting on the controller role.

Also included is support for acting as sink for the Advanced Audio Distribution Profile **(A2DP)** version 1.2, using PulseAudio to provide audio routing. When the device starts to send an A2DP stream to Apertis PulseAudio will automatically make it available as an output device.

PulseAudio has a module that can automatically redirect streams to new output devices. However, for systems with complex requirements for audio routing it's probably a better idea to have a system daemon or application managing that; the car system interface D-Bus daemon is one viable candidate.

The Video Distribution Profile **(VDP)** is not yet supported

HFP

The Hands-Free Profile **(HFP)** version 1.6 is supported by BlueZ. Hands-free is the technology that allows making phone calls with voice commands, and having audio routed from the phone to a different device, such as the Apertis system which can then play it to the car speakers, for instance. The BlueZ framework, along with oFono, can be used to add hands-free support to the system. Wide Band Speech –high quality audio for calls, though, is not yet supported by BlueZ.

After a SIM-enabled device in Audio Gateway mode has been paired with BlueZ, PulseAudio will be able to use it as source and sink through its Bluetooth module and route streams from the car's microphone to the phone and the audio from the call to the car's speakers. In this case BlueZ acts as in the Hands-free role.

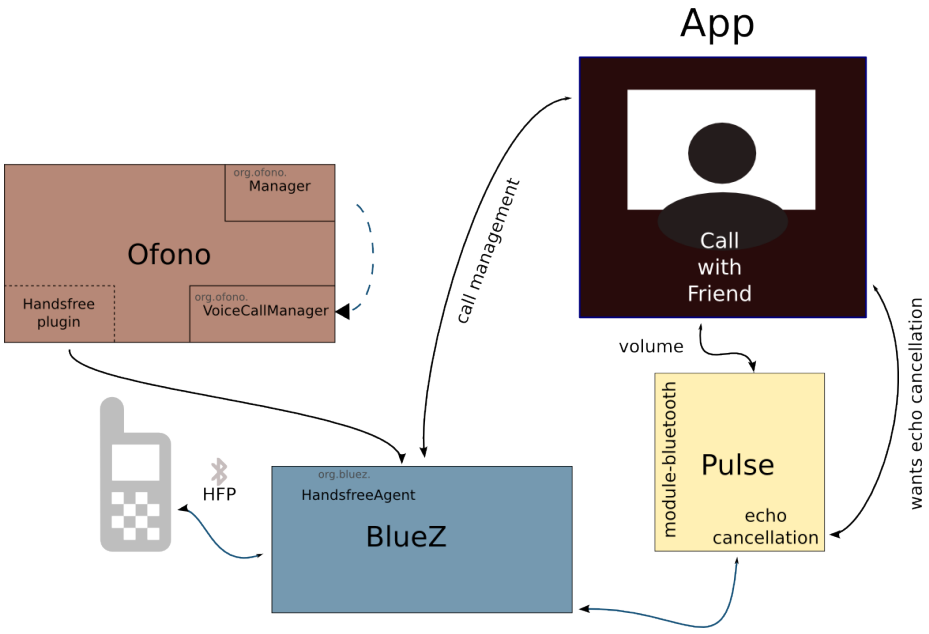
The application which handles the calls can use PulseAudio APIs to control the volume of the source and sink streams, and should set the **filter.want** property of the PulseAudio streams[13] to let PulseAudio know echo cancellation should be used.

This will cause PulseAudio to automatically load the echo cancellation module. The echo cancellation module can also contain a noise cancellation sub-module. PulseAudio ships with an Open Source sub-module based on speex for echo cancellation, but it can be replaced by custom or proprietary modules if required, which was the course chosen by Nokia for its phones, for instance. The same goes

[13] http://freedesktop.org/software/pulseaudio/doxygen/proplist__8h.html#a87c586045175fa05e28e6ee1cbaac4de

for the noise cancellation sub-module, it can be easily replaced by an proprietary noise cancellation solution just by rewriting the sub-module.

The diagram in Illustration shows how the various pieces of such a set-up are related.



HSP

Currently BlueZ has no support for the Headset Profile **(HSP)**. Collabora recommends it be supported, but that would require development of the feature. VoIP applications rely on HSP to operate calls over Bluetooth, It is also important to mention that older phones only support the HSP profile for phone calls.

GSM 07.07 AT-commands

oFono has support for most of the GSM 07.07 AT command set. It is through the AT command set that we control the phone in the HFP profile.

GNSS

The Global Navigation Satellite System Profile is currently not supported by BlueZ, nonetheless adding support would be simple in the BlueZ side. For this profile, BlueZ would only provide the Bluetooth connection handling, all the navigation specific data would be passed to the GPS specific application to handle it.

Global Positioning System (GPS)

The Apertis platform provided by Collabora will include the GeoClue geolocation framework. GeoClue[14] provides a D-Bus service that can be queried to establish the current location of the system with configurable accuracy. GeoClue is able to use the GPS from the system to provide very accurate location information.

This technology will be used, for instance, to power the existing GeoLocation implementation of the WebKit-Clutter library. The service can be made available for use by store apps, potentially including selective restrictions on the accuracy each app can query. This should be discussed and specified during the development phase.

The connectivity considerations document discusses using GPS data for predicting connectivity conditions, and pre-emptively switching to a different connection before entering an area with bad coverage, for instance. This seems to be a risky strategy unless very up-to-date and very extensive data are accessible to the system at all times. In any case, the GeoClue framework could serve the purpose of providing the location information from the GPS.

One additional advantage of using GeoClue is it supports using different providers for its information like cell towers through oFono-based gsmloc, WiFi networks through integration with ConnMan, IP addresses through HostIP, and so on.

Note that HostIP is essentially useless for mobile use cases, since it tries to use the IP address as an indication of the location, but that is not very accurate in general and for mobile specifically

> A somewhat outdated list: https://gitlab.freedesktop.org/geoclue/geoclue/-/wikis/home

Those could be useful to provide location information with coarse accuracy for applications such as the web browser with no need for turning the GPS on or for systems with no GPS hardware. If only GPS matters, and tighter control is required, Collabora can support using the GPS service directly through gpsd[15] or gypsy[16].

If different accuracy levels want to be defined that the store application can use, different permissions for GPS access can be created representing the different levels of accuracy. These permissions would be specified in the application' s manifest and prompted to the user at the moment the user authorizes the installation of an application. Refer to the Applications design for more details on this.

[14] http://www.freedesktop.org/wiki/Software/GeoClue
[15] https://savannah.nongnu.org/projects/gpsd
[16] http://gypsy.freedesktop.org/

Media Downloading

This chapter discusses how communication with various devices is made to provide the Apertis system with ways of downloading media from them. For more detailed information on use cases, requirements, problems and solutions refer to the Media Management design document (sometimes called the Media and Indexing design) prepared by Collabora.

In general media will be brought into the system through USB sticks, mobile devices and online sources. USB sticks are mounted using the USB mass-storage support. Most USB sticks use the VFAT file system, which is, unfortunately, patent-encumbered and has been used in the past by Microsoft to promote law suites against companies shipping devices that support the file system, see http://www.groklaw.net/article.php?story=20090401152339514.

When considering communication with specific devices the ones that stand out are the Apple devices, which are both very well-known and have widespread usage. This document discusses the Open Source tools that are currently the state of the art for communicating with Apple devices but does not specifically recommend their usage.

The libimobiledevice[17] suite is the state of the art on Open Source libraries and tools for accessing Apple devices. It implements the protocols used for communication with Apple's iPhone, iPad, iPod Touch and TV products, covering almost all available functionality, including downloading of music and video, when used in conjunction with libgpod of the gtkpod project22. Its pairing tool is also a requirement for using the ipheth driver mentioned before.

The project is a community effort and although it does not require the devices to be jail-broken, it's not supported by Apple, which means the protocol is reverse-engineered and often lags behind recent Apple releases. As an example, iOS 5 support has only recently (22/03/2012) seen the light of day in a release. Despite these shortcomings, the suite would provide the technical means for writing the applications that interact with Apple products.

Microsoft has also developed a protocol for media exchange called Media Transfer Protocol (MTP). This protocol has been standardized and is currently published by the USB implementers forum[18]. A LGPL-licensed library exists that supports the *Initiator* side of the communication, meaning it is able to access media on devices that support the MTP R*esponder* side: libmtp[19]. The libmtp library is currently shipped as a part of Ubuntu and can be provided in the Apertis middleware platform to be used to implement applications. Note that as is the case for Apple devices, Collabora is unable to provide legal counselling about the use of libmtp.

[17] http://www.libimobiledevice.org/
[18] https://usb.org/sites/default/files/MTPv1_1.zip
[19] http://libmtp.sourceforge.net/

UPnP

Universal Plug and Play (UPnP[20]) is a protocol used for discovering and browsing multimedia content made available by media centers. This protocol will be supported by the Apertis middleware platform using the gupnp library. For more information about this please see the Media Management design document (sometimes referred to as Media/Indexing design).

[20] http://www.upnp.org/