Contacts

Contents

This document outlines our design for address book contacts within the Apertis system. It describes the many sources the system can draw upon for the user's contacts, how it will manage those contacts, which components will be necessary, and what work will be needed in the middleware space to enable these features.

Contacts are representations of people that contain some details about that person. These details are often directly actionable: phone numbers can be called, street addresses may be used as destinations for the navigation system. Other details, such as name and avatar are purely representational.

We propose a contact system which uses the Folks contact aggregator to retrieve contacts from multiple sources and automatically match up contacts which correspond to a single person. This will give a thorough and coherent view of one's contacts with minimal effort required of the user.

Integrated Address Book Versus Alternative Solutions

The following design is based around the concept of a heavily-integrated address book which links together contacts from many contact sources, providing a common interface for applications to access these contacts. As presented below, the only available contacts which will not be fully-integrated into the common contact view will be contacts available on a paired Bluetooth device.

The level of contact source integration is flexible. If it is preferred to limit contact integration to the local address book and chat/Voice-over-IP contacts to, for example, isolate Facebook or Twitter contacts in their own address book(s), to be accessed by a special library, Collabora is ready and able to adjust this design.

Contact Sources

There are many potential sources for contacts, as people's contact details are frequently split over many services. The proposed system aggregates contacts from multiple sources in a way that is seamless to the user. See the Components section on Folks for more details of the components involved.

Local Sources

New contacts may be created locally by importing contacts from a Bluetooth-paired phone or a contact editor dialog (see User interfaces).

These local contacts may contain a wide variety of detail types, including (but not limited to):

- Full name
- Phone numbers
- Street addresses
- Email addresses
- Chat usernames on various services
- User-selected groups
- Notes

Bluetooth-paired phone

Synchronization

Contacts may be simply synchronized to a Apertis system by means of a SyncML[1] contact transfer from a phone paired with the Apertis system over Bluetooth. This operation is designed to intelligently merge fields added to contacts on the source phone to avoid creating duplicates.

[1] http://en.wikipedia.org/wiki/SyncML

To manage complexity, this function will only be supported from a phone to the Apertis system, not the other way around. Systems which support two-way contact synchronization have a number of issues to contend with, including:

- Contacts do not contain "last modified" time stamps, so it is rarely obvious how to resolve conflicts
- "Fuzzy-matching" fields for cases of equivalent names or phone numbers is not consistently implemented across different systems (if it is implemented at all)
- Even if equivalent fields are correctly matched, it is not clear which version should be preferred
- Because conflict resolution may not be symmetrical between the two directions of synchronization, the contacts in the two systems may never reach a stable state, potentially causing other side effects (such as duplicates on the phone)

By limiting synchronization from the phone to the Apertis instance (with a "source wins" conflict resolution policy), we can avoid the aforementioned issues and more. This simpler scheme will also be easier for users to understand, improving the user exeperience.

Synchronization will be performed automatically each connection of a phone to the Apertis system.

Each phone device will receive its own contact address book on the Apertis system which will be created upon first connection and re-used upon subsequent connections. This is meant to make it trivial to remove old address books based upon storage requirements.

Chat and Voice-over-IP Services

Most chat and some Voice-over-IP (VoIP) services maintain contact lists, so these are another potential source of contacts. We recommend supporting contacts from audio- and video-capable services, such as Session Initiation Protocol (SIP), Google Talk, and XMPP. These contacts and their services provide an alternative type of audio call which users may occasionally prefer to mobile phone calls for purposes of call quality and billing.

Additionally, contacts on some of these services may provide extended information, such as a street address, which the user might not otherwise have in their address book.

Our system will cache these contacts and their avatars from the service contact list. This will allow Apertis applications to always display these contacts. When the user attempts to call a chat/VoIP contact while offline, the system may prompt the user to go online and connect that account to complete the action.

From a user perspective, the configuration of chat and VoIP accounts within Apertis would be simple. In most cases, just providing a username and password

will add that user's tens or hundreds of service contacts to the local address book. For limited effort, this can significantly increase the ways the user can reach their acquaintances in the future.

Web services

The growing number of web services with social networking is yet another source of contacts for many users. Some services may provide useful contact information, such as postal addresses or phone numbers. In these cases, it may be worthwhile to include web service contacts (since implementation for some services already exist within Folks and libsocialweb.

In the case of multi-seat configurations, it may also be worthwhile to support additional web services for entertainment purposes. Potential uses include playback of contacts' YouTube videos, reading through contacts' Facebook status updates, Twitter tweets, and other use cases which do not apply to a driver due to their attention requirements.

In general, web services require third parties access their content through a specially-issued developer key. In many cases, this will require to secure license agreements with the provider to guarantee reliable service as their terms of service change frequently (usually toward less access).

Our system will cache these contacts and their avatars from the service contact list. This will allow Apertis applications to always display these contacts, even when offline.

SIM Card

Contacts may be retrieved from a SIM card within a vehicle's built-in mobile phone stack. These contacts will be accessible from the Apertis contacts system. However, any changes to these contacts will not be written back to the SIM card. See Read-only operation for external sources.

Read-only Operation for External Sources

Modifications of contacts will be limited to Local sources. Depending upon the user interfaces created, users may be able to set details upon local contacts which may appear to affect external contacts such as web service contacts or Bluetooth-connected phone contacts. However, these changes will not actually be written to the corresponding contact on the external source.

Standard Behavior and Operations

Contact Management

Our proposed system will support adding, editing, and removing contacts. New contacts will be added to Local sources. Though the Components which will enable contact management already support these features, User interfaces needs

to be implemented to present these functions to the user. Similarly, contacts will need to be presented as necessary by end-user applications.

Contact Aggregation and Linking

Contacts will be automatically aggregated into "meta-contacts" which contain the sum details amongst all sub-contacts. The criteria for matching up contacts will be:

- **Equivalent identifier fields** – for instance, two contacts with the email address bob@example.com[2] or phone numbers "+18001234567" and "1-800-123-4567"
- **Similar name fields** – for instance, contacts with the full names "Robert Doe", "Rob Doe", and "Bob Doe" (which all contain variations of the same given name)

This system will be careful to avoid matching upon unverified fields which would allow a remote contact to spoof their identity for the purpose of being matched with another contact. In a real-world example, Facebook contacts may claim to own any chat name (even those which belong to other people). If we automatically matched upon this field, they could, theoretically, initiate a phone call and appear to the user as that other person.

The user will also be able to manually "link" together any contacts or, similarly, manually "anti-link" any contacts which are accidentally mismatched through the automatic process.

Linking and anti-linking will be reversible operations. This will avoid a user experience issue found in some contact aggregation systems, such as the one used on the Nokia N900.

Local Address Book Management

The Apertis contacts system will support adding and removing local contact stores in an abstract way that does not assume prior knowledge of the underlying address book store. In other words, to add or remove an underlying Evolution Data Server contact database, a client application will be able to use functionality within Folks and, indeed, not even need to know how the contacts are stored.

Search

This contact system will include the ability to search for contacts by text. Search results will be drawn from all available contact sources and will support support "fuzzy" matching where appropriate. For instance, a search for

[2] mailto:bob@example.com

the phone number "(555) 123-4567" will return a contact with the phone number "+15551234567" and a search for the name "Rob" will match a contact named "Robert."

Each type of contact detail field supports checking for both equality (for example, "Alice" "Carol") and equivalence (for example, the phone number "(555) 456 7890" is equivalent to "4567890"). This allows the contact system to add or change fuzzy matching for fields without needing to break API or treat certain field details specially based upon their names.

Sorting and Pagination

As a convenience for applications and potentially an optimization, the contacts system will support returning search results in sorted order (for example, by first name).

Furthermore, the search system will support returning a limited number of results at a time ("paginating" the result set). This may improve performance for user interfaces which only require a small number of results at once.

Event Logging

Related to the contacts system, Collabora will provide an event logging which logs simple, direct communication between the user and their contacts. Supported events include VoIP and standard mobile phone calls, SMS messages, and chat conversations.

Events will include at least the following fields:

- **User Account ID** – e.g., "+15551234567", "alice@example.jabber.org[3]"
- **Contact service ID** – the unique ID of the contact involved
- **Direction** – sent or received
- **Event type** – call, text message
- Timestamp
- **Message content** – for text messages of any type
- **Success** – whether the call successfully connected, whether a text message was successfully sent

The contact service ID can be used by applications to look up extended information from the contacts system, such as full names and avatars. These details can then be displayed within the application to provide a consistent view of contacts when displaying their conversations.

Out of Scope

Email conversations will be out of scope due to their relatively large message sizes and their common use for indirect conversations (such as mailing list mes-

[3] mailto:alice@example.jabber.org

sages, advertisements or promotions, social networking status updates, and so on).

Messages exchanges with web service contacts will not be supported by default. However, the event logging service will allow third-party software to add events to the database. So events not logged by default by the middleware may be added by entirely third-party applications.

Caching

In general, contact sources will be responsible for maintaining their own cache in a way that is transparent to client applications.

Opportunistic Caching

It may be best to defer bandwidth-intensive operations (such as full contact list and avatar downloads) until the Apertis system can connect to an accessible WiFi network (such as the user's home or work network).

Open Questions

Will there be a general framework for libraries and applications to check whether network data should be considered "cheap" or "too expensive"? And should the contacts system factor that into its network operations?

Most bare contact lists (not including avatars) have trivial data length. For example, my very large Google contacts list of 1,600 contacts only contains 171 kilobytes of data. Common contact lists are substantially smaller than that.

When factoring in avatars (for the first contact list download), contact list sizes can potentially reach a few megabytes in the worst case. This could be an unacceptable amount of data to transfer on a pay-as-you-go data plan. But at the same time, this is a relatively small amount of data and will only get relatively smaller as data service plans improve.

Considering the factors above, would it be worthwhile for the contacts system to support opportunistically caching remote contact lists on bandwidth-limited networks?

Components

Folks

Folks[4] is a contact management library (libfolks) and set of backends for different contact sources. One of Folks' core features is the ability to aggregate meta-contacts from different contacts (which may come from multiple backends). These meta-contacts give a high-level view of people within the address book, making it easy to select the best method of communication when needed.

[4] http://telepathy.freedesktop.org/wiki/Folks

For instance, the driver could just as easily call someone by their SIP address as their mobile phone if they prefer it for call quality or billing reasons.

The actively-maintained Folks backends include:

- – **Telepathy** – Chat and audio/video call contacts, including Google Talk, Facebook, and SIP
  – **Evolution Data Server (EDS)** – Local address book contacts
  – **libsocialweb** – Web service contacts, including YouTube and Flickr

Many of these backends have associated utility libraries which allow client software to access contact features which are unique to that service. For instance, the Telepathy backend library provides Telepathy contacts, which may be used to initiate phone calls.

Bindings

The Folks libraries have native bindings for both the C/C++ and Vala programming languages. There is also support for binding any languages supported by GObject Introspection (including Python, Javascript, and other languages), though this approach has less real-world testing than the C/C++ and Vala bindings.

Required work

As described in Contact aggregation and linking, our system will support automatic linking of contacts as well as anti-linking (for mismatched automatic links). Folks currently supports recommending links but does not yet act upon these recommendations automatically, so this would need to be implemented.

Along with this, Folks will need the ability to mark contacts specifically as non-matches (by anti-linking them). There is preliminary code for this feature, but it will need to be completed for this functionality.

In order to enable display of chat/VoIP contacts while offline, we will need to implement a chat/VoIP contact list cache within Folks. This will be similar to existing code for caching avatars, but simpler.

Similarly, we will need to implement a web service contact cache to display web service contacts while offline.

Search functionality in Folks is nearly complete but still needs to be merged to mainline[5].

Additionally, the ability to perform "deep" searches will require support for search-only backends[6].

The search functionality will also need to support sorting and pagination as described in Sorting and pagination before it can be merged upstream.

[5] https://bugzilla.gnome.org/show_bug.cgi?id=646808
[6] https://bugzilla.gnome.org/show_bug.cgi?id=660299

Folks external contact sources will need the ability to be designated as "synchronize-only" or "keep-remote". Contact sources designated as synchronize-only will be automatically synchronized as necessary (such as when a phone is connected over Bluetooth). Keep-remote sources will not be synchronized to the Apertis system and will only be accessible while the remote source is available (whether over a local or Internet connection).

For Folks to access contacts stored on a vehicle's built-in SIM card, we will need to write an oFono backend to retrieve the contacts from that hardware.

Abstract contact address book creation and deletion within Folks will require new work.

In case Opportunistic caching is required for the contacts system, this will need to be added as a new feature to Folks and its Telepathy and libsocialweb backends.

Support for storing arbitrary data in contacts has not yet been implemented in Folks, but has already been discussed[7] and will be implemented.

Out of scope

We recommend application logic for synchronizing an entire address book from a Bluetooth-paired phone be implemented in a new library or application on top of SyncEvolution (which we will provide in our Reference images). The contacts created in this process will automatically be stored as any other local contact.

Speech-based search has been identified as a major use case for the address book software in Apertis. The text-based search portion of this use case will be supported by Folks; however, the parsing of audio data into a text for searching will be the responsibility of specific software above the middleware. Global search in general will be covered in the upcoming document "Apertis Global Search".

Collabora recommends to implement the voice search in whole or in part as a service daemon started automatically upon boot. This would allow dependent functionality, including Folks, to be initialized in advance of user interaction. This will be necessary to minimize latency between voice search and the display of results.

Support for contact caching for abstract third-party backends certainly would be possible and would likely take the form of a vCard contact store. However, at this time, Collabora recommends not implementing this feature. We would much prefer to delay this until there exist at least two third-party Folks backends with which to test this functionality during development. This is primarily due to the risks involved with committing to an API. Once officially released, this API will need to be kept stable. So it is critical that the API be tested by multiple independent code bases before finalization. Furthermore, at this time,

[7] https://bugzilla.gnome.org/show_bug.cgi?id=641211

there exist no known third-party Folks backends. In the meantime, third-party backends could still implement opaque contact caches suited to their own needs and migrate to a centralized implementation if and when it is created.

Telepathy

The Telepathy[8] communications framework, which Collabora created and maintains, retrieves contacts for many types of chat services, including Google Talk, Facebook, XMPP, and most other popular chat services. It also supports supports audio and video calls over SIP, standard mobile phone services, and the previously-mentioned chat services (depending upon provider).

Evolution Data Server (EDS)

Evolution Data Server is a service which stores local address book contacts and can retrieve contacts stored in Google accounts or remote LDAP contact stores. Contacts may contain all defined and arbitrary[9] vCard[10] attributes and parameters, which is a common contact exchange format in address book systems. This allows Folks to store and retrieve contacts with many types of details.

EDS is the official address book store for the Gnome Desktop and has been used in Nokia's internet tablet devices and N900 mobile phone. It has been the default storage backend for Folks since Gnome 3.2, which was released in September, 2011.

libsocialweb

In the case that we support web service contacts, libsocialweb will be the component that provides these contacts through its Folks backend. Note that exactly which web services can be used depends upon both implementation in libsocialweb and license agreements with those services. See Web services for more details.

SyncEvolution

SyncEvolution[11] is a service which supports synchronizing address books between two sources. While it supports many protocols and storage services, it best supports synchronizing contacts from a SyncML client over Bluetooth to Evolution Data Server, which will be our primary contact store. Many mobile phones support the SyncML protocol as a means of contact synchronization.

[8] http://telepathy.freedesktop.org/wiki/
[9] http://www.ietf.org/rfc/rfc2426.txt
[10] http://en.wikipedia.org/wiki/VCard
[11] http://syncevolution.org/

This method requires Bluetooth OBEX[12] data transfer support, which is widely supported by most Bluetooth stacks, including BlueZ[13].

Zeitgeist

Zeitgeist[14] is open source event-tracking software that will serve as the Event logging service for Apertis. It is a flexible event store and uses external services to store their events in a central location. So, by its nature, it supports third-party applications without prior knowledge of them.

Zeitgeist is committed to API stability in part because Ubuntu's Unity user interface depends upon it.

Required Work

A simple service to monitor and send Telepathy chat and VoIP call events to Zeitgeist is in progress, so this work will need to be finished and merged upstream.

Architecture

In our recommended architecture, contacts applications will use libfolks directly. Libfolks, in turn, will use its Telepathy backend for chat and VoIP service contacts; Evolution Data Server backend for local contacts, and its libsocialweb backend for web service contacts.

Not pictured in is the optional linking between the application and each backend's utility library (for accessing service-specific contact features).

Accessibility of Contacts By Source

Contacts within this system are accessible on two levels: Meta-contacts, representing an entire person, are available for all contacts in the system. Each meta-contact contains at least one contact. For many use cases, applications can work entirely with meta-contacts and ignore the underlying contacts. For use cases requiring service-specific functionality, such as initiating an audio call with a Telepathy contact, applications can iterate through a meta-contact's sub-contacts.

Additionally, applications can access contacts for each user account. Each account has a corresponding contact store containing only the contacts for that account. So, an application could be written to display only contacts from single account or service provider at a time (ignoring any parent meta-contacts if it instead wishes to work in terms of service contacts).

[12] http://en.wikipedia.org/wiki/OBEX
[13] http://www.bluez.org/
[14] http://zeitgeist-project.com/

User interfaces

As Folks and Telepathy are a set of libraries and low-level services, they do not provide user interfaces. There exist a few open source, actively-maintained applications based upon Folks and Telepathy:

- **Gnome Contacts** – an "address book" application which supports contact management and searching
- **Empathy** – a chat application which provides a chat-style contact list and both audio/video call and chat handler programs

Together, these components provide most contact functionality including:

- Adding new contacts
- Editing or removing contacts
- Browsing/searching through contacts
- Importing contacts from a Bluetooth-paired phone
- Initiating and accepting incoming phone calls

However, these applications are designed for use on a typical desktop environment and do not suit the needs of an in-vehicle infotainment user experience. We recommend to examine these applications as real-world examples of contact applications which use the components we recommend for the Apertis contacts system.

Multiple Users

Each user in the system will have their own contacts database, chat/VoIP accounts, and web service accounts. Changes by one user will not affect the contacts or accounts of another user.

Storage considerations

The storage requirements for our proposed contacts system will be very modest. Storage of local address book contacts should be under a few megabytes for even large sets of contacts with up to several megabytes of storage for contacts' avatars.

These storage requirements do not factor in files received from contacts.

Abstracting Contacts Libraries

In general, Collabora discourages direct, complete abstractions of libraries because the resulting library tends to have fewer features, more bugs, and gives its users less control than the libraries it's meant to abstract. Particularly, when abstracting two similar libraries, the resultant library contains the "least common denominator" of the original libraries' features.

However, partial-abstraction "utility" libraries which simplify common use patterns can prove useful for limited domains. For instance, if many applications

required the ability to simply play an audio file without extended multimedia capabilities, a utility library could dramatically simplify the API for these applications.

As such, Collabora recommends against abstracting Folks or Zeitgeist on a per-component basis as they are designed to be relatively easy to integrate into applications. But, for example, it would make sense to create a library or two which provide widgets based upon these libraries. This could create a contact selector widget based on top of Folks, allowing applications to prompt the user to pick a contact with only a small amount of code.

Another recommended widget to add to such a library is a "type-ahead" contact selector as is common in many email applications. As the user types into a "To:" entry field, the widget would the Folks search capabilities to return a list of suggestions for the user to select from.