Decibel: Short Concept Proposal

Aim of this paper is to describe the motivation and the concept of the Decibel project and related technologies.

Decibel

Motivation: In the near future, almost all communication is going to be computer based. While email and web browsing is a well-established technology, real time communication (like audio and video conferencing) are the next challenging communication techniques. While a user just needs one email application and one web browser concurrently, users have to face the unsatisfying reality that they need several applications simultaneously to be able to do voice and video communication. One usual solution to reduce the number of applications on the desktop were plugin-based tools. These applications integrate various protocols using plugins, thus just one application had to run. This solution looked like the solution to this problem, but it still had three major drawbacks:

1. Stability: If one plugin crashes, the whole Applications crashes, too.
2. Flexibility: Due to a relative static user interface (even if it is exendable by plugins), plugin based applications tend to be bloated.
3. Extensibility: Due to a missing interface standard, Plugins are not exchangeable between applications.

Decibel wants to clean up this situation by offering a desktop integrated concept, which will stop with chat applications as known today. Users are neither forced to use one or various applications nor they have to think about protocols if they just want to communicate with a distant person. Within Decibel, they will only choose a contact and a preferred communication type (text, audio, video). This decision will be made inside any application which will be able to handle contacts (address book, mail client, whatever..). Decibel will initiate the connection and chooses the right protocol for the job.
Providing an easy to use interface and a secure and solid connection, Decibel will ease communication and it will lower the amount of windows at the users desktop. Last but not least, the modular architecture of Decibel will make the framework flexible for expansions and result in a much more stable platform than plug-in based applications can offer.

Most of these improvements are possible by using the Tapioca framework. This Framework will be used and – if needed – extended to provide features to realize this motivation. The demonstration implementation will use KDE to show the feasibility of this concept.

First, we will describe some important aspects that take influence to the design of decibel. Then, we will describe components of Decibel and Tapioca.

Contacts and Presences

Contact information is needed to contact distant persons. Contact informations are usually stored into a database. Regarding to the underlying technology, it may be possible to discover, whether a contact is currently reachable or not. This kind of information is called “presence information”.

Remote Contact and presence information may be gathered from various sources. Some are related to the desktop and some may be stored on a remote server. The own contact information is stored locally, while the user has the choice whether he wants to show his presence to the public or not.

Contact informations are the central aspect to initiate new conversations. While the contact information itself is unique, one person may expose multiple contacts and presence information simultaneously. To solve this problem, Decibel will monitor all supported protocols for contacts and presence information. All of them will be stored and gathered within a central place to obtain one presence per communication type for each contact. To identify identical contacts a user feedback maybe necessary and should be included into the desktop interface.

Communication types

As described above, the technology (like protocols, codecs, …) should be covered by the framework. To consolidate different protocols a classification of communication types is needed. These communication types must stay simple enough to include many protocols, but strict enough to avoid intersections. For instance, a text type may include Jabber and IRC, while the type audio may contain VoIP, Skype and Google-Talk.
Decibel has to decide which supported protocol to use if the user initiates a new connection of one communication type. Therefore, a priority list of protocols will be specified. Of course, the user will have the right to reconfigure the priority of the list.

The following communication types may be used. The list is in random order and may be imperfect:

- video calls / video conferences
- voice calls / voice conference
- synchronous message exchange
- asynchronous message exchange

Hierarchical Configuration

Decibel will use a two level configuration hierarchy. This will help the framework to work with minimal user configuration and offer the opportunity to specify special preferences for dedicated users.
The two levels will consist of a system and a user configuration. The basic configuration will describe a working set of protocols, connection managers. Just PIM and login informations is missing to connect to a remote server, which is provided by KWallet and Akonadi (as listed below).
By reading the user configuration after the systems default, the settings are overloaded. Thus, a user can change the default configuration in a very easy manner.
To integrate new protocols easily, configuration additional information may be needed and added. A concept how Decibel will handle this without help from the user must be specified.

Components of Decibel

Decibel will be divided into different components that will use various frameworks. The following section will describe all modules in detail. Some of these components are part of related projects that fit well to Decibel. Additional modules will be added transparently into tapioca to add needed functionality. A short notice to external projects will be given within the description. Here is a brief overview of all components:

- Houston
- Tapioca
- Desktop user interface
- Client user interface
- Connection managers
- Kwallet
- Akonadi

Now let's take a closer look to these components.

Desktop User Interfaces

The desktop user interfaces are used to control the communication framework and to access internal data. It will be fully integrated into the Desktop and give the easiest available access to all known contacts to establish communication and to configuration settings.

Instead of having one big application, which shows all aspects of the framework, Decibel is accessible using specialized components which just show special views to it. While an applet may be perfect to show currently available presences, a configuration application may be perfect to change the configuration settings. Additionally, views to the framework will be available in other applications. For instance, the address-book and the mail application will provide the feature to communicate with a person who is currently selected.

To initiate a conversation, the user just selects the wished contact from the list and chooses the available communication types, provided by the framework. Decibel will try to establish the requested communication and may start a client user interface.

Last but not least, the user can specify his presence for each communication type through the desktop user interface.

Client user interface

While the Desktop User Interface will provide views and access to the framework, the client user interfaces will provide needed functionality that is necessary to handle an already established communication channel. This can be a text based chat-, a phone- or a videoconference component. Thus, the client depends on the protocol that is currently used.

Client user interfaces are exchangeable transparently if they belong to the same communication type. And they may be combined to integrate complex behavior. For instance, an ongoing videoconference may be handled by various text based chat components, by an audio and a video component. These components may be used in several widgets or integrated into one application, using the KDE component system.

Houston

The glue between all components will be called Houston (but we don't have a problem). Houston controls nearly everything within Decibel so it becomes the most important part. Houston will be based on the Tapioca APIs and become a persistent component.

For some reasons, it is possible that the requested connection can't be established. Decibel will notice this and will try to obtain a fallback solution. The desktop user interface will notify the user about this new situation and will ask for acknowledge. A small example may clarify the workflow.

Example: A user wants to make a video call and requests it from Decibel. Decibel tries to establish a video call connection, but the camera of the remote contact is currently busy. (The camera is in use by another application outside of Decibel!) Decibel notices the problem, checks fallback communication types and notifies the user about the problem by asking for acknowledge. Decibel may offer to establish a voice call for compensation. If the user accepts, Decibel will try to establish a voice call connection.

To summarize the tasks of Houston:

- Decide whether a connection manager must be started or stopped for any specified protocol. Store information about it's communication type.
- Validate if a connection is possible to establish, initiate it and give a fallback solution if a connection can not be started for some reasons. Fallbacks maybe useful if bandwidth of network connection gets lower too. Houston should find a clever way to decide when a fallback has to be suggested to the user.
- Initiate startup of client user interfaces or applications and monitor if they get closed. When a user closes an application it maybe necessary to stop some connection managers to free resources.
- Listening to all presence signals, provided from active connection managers. Store all these information and match them with the possible communication types of the connection managers. Publish the centralized contact and presence information to the desktop user interface if requested.
- Communicate with Kwallet and Akonadi to feed PIM and login to the connection managers if they are needed. **A secure communication over the Dbus is needed here!**
- Act as agent between connection managers and client user interfaces. Providing Login and PIM data in a secure way, establish and controlling the connection and cleanup all resource at the end.

Even if we will provide a KDE-based example implementation of Decibel, Houston will be implemented independently to any specific desktop environment. Thus, Housten will be available for GNOME based environments. Future implementations for Windows and MacOS-X will be kept in mind, too.

Tapioca

Tapioca is a project that already realized a Jabber / Google Talk client that uses a DBUS based backend. Tapioca has an easy to use API that abstracts the backend. This included features like startup of the needed connection managers for a given protocol and the abstraction of the communication between client and connection manager.

Its core API for developers will be used (and may be exteded) to:

- startup clients and connection managers for a specified protocol
- stop components that are no longer needed to free resources
- abstract communication between Houston, Connection Managers, Clients and the desktop user interface

It is very important to communicate with the tapioca developers to ensure that the Tapioca API will obtain all features, needed by Decibel. On the other side, Tapioca clients should work with Decibel, too. As Decibel clients will mainly communicate with Houston, we will use the Tapioca APIs as the common interface between Decibel and Tapioca. Thus, Tapioca clients will still work alongside with Decibel.

Connection managers

The connection managers belong to the tapioca framework and integrate the protocol implementation to the external communication partners. Thus, they abstract the functionality of the communication protocol to the framework.

They communicate with servers or directly with other clients (peer to peer). Dependent to the underlying protocol, they provide contact and presence information to the framework, which will be stored in Houston. Local presence information is exposed to the network as configured by the user.

Kwallet

Kwallet is a KDE application that stores sensible data within the KDE desktop. Due to the fact that Decibel needs a way to store logins and passwords in a secure manner, Houston needs a .
KDE currently integrates Dbus into it's desktop. This means communication for many KDE compotes will get included. As soon as Kwallet is attached to the Dbus communication with Houston can get included with small effort.

Communication with the Kwallet developer must be initiated to clear Dbus specifications. Kwallet won't be needed at the initial development process.

Akonadi

Akonadi is currently in development and will be used to access PIM data, which is stored in a database. It is already attached to the DBUS, thus it will be accessible with a minimal effort. Currently, the developers of Akonadi have shown some examples that their concept is working properly. Communication with the developers is planned.

Use Cases

To clearify the attached use case diagrams, we will provide some textual written use cases. They describe the following three situations:

1. Startup of the framework
2. Outgoing connection
3. Incoming connection

Section: Initial framework startup and running state

- Framework startup: Houston, connection managers, tapioca and desktop user interface will get started.
- The connection managers log into their communication servers using login and password provided by Houston. Presence information will be sent out and received.
- The connection managers notify Houston about every presence information that comes in. There communication type will be delivered to Houston too.
- Houston stores and collects all presence information and its assigned communication type.
- Houston informs the desktop user interface about presence and communication changes.
- The desktop user interfaces receives presence information and communication types. It displays them in clearly arranged way to inform the user.

Outgoing connection – the user wishes to establish an connection

- The user selects a contact and communication type to startup a new conversation with an external contact.
- The desktop user interface will signal the connection wish to Houston and awaits the establishment of the connection.
- Houston receives the connection wish and validates the possibility to establish the requested connection.
- If Houston accepts the connection wish and no fallback must be provided the connection will be initiated. Connection manager (if not running) and client(s) will be started.
- Houston emits the connection object to the client and connection manager so they can communicate with each other.
- Now the user can use the client user interface to communicate with the external contact through the connection manager.

Incoming connection – an external user wants to communicate with us

- An external contact wants to establish a communication with the user. A connection manager emits the connection wish to Houston.
- Houston receives the external connection wish and validates whether the connection can get established at the moment.

- Houston may ask, by help of the desktop user interface, if the user agrees to the external connection wish.
- If the connection can be established Houston will create a connection and starts the needed client user interface(s). The connection object will be send to the client and connection manager.
- Now the user can communicate with the external contact through the connection manager.

Use case diagrams

There are two use-case diagrams attached to this document. On shows the incoming and the other the outgoing connection. Using the description above, it should be easier to understand the details.