Project Proposal
Service Architecture for Multimedia Based Communication

Dipl.-Ing. Eva Brucherseifer
Dipl.-Ing. Stefan Eilers

02/01/06

Contents

1. The Situation

Current computer systems provide a wide range of options for communication. Text based approaches (like mail, web logs (today just called blogs) and text chat) are long established technologies, while the communication is changing towards audio and video based solutions.
As sending mails and web logs is based on one standardized protocol in each case, a wide range of stream based solutions using different protocols and codec’s, even if they are doing basically the same (from the user point of view).
For instance: Providing an IP-based telephone connection may be realized by various protocols, like SIP, H.323, Skype and more. While the protocols are different and don't allow to interact, even the encoding and decoding of the content stream (audio or video) may use different technologies. Thus, two SIP endpoints may be unable to communicate if they don't provide a common codec to handle the content stream in a correct manner!
Due to this situation, users are forced to use several applications simultaneous to gain access to various network protocols. Thus, power users face a situation which is nearly unbearable: While one mail client and one web browser is sufficient for accessing all mails and blogs (or other html-based services), he will need various chat clients (for instance a Jabber-, MSN-, AOL-Messenger and an IRC client), a minimum of two VoIP-Clients (Skype, SIP) to stay in contact with his friends and colleagues. And this tendency is increasing permanently.
There already exist several solutions to reduce the problem illustrated here. One idea is to use a protocol which is capable to integrate others via gateways, like the Jabber protocol does: Users just need one Jabber client to stay connected to all friends which may reside on any other network. This solution has a big drawback as most protocols uses peer-to-peer solutions to realize the channels for the content stream. Due to this fact, the gateway has to work as a proxy for all connected users, which causes a bottleneck and will lead to unacceptable latencies and the inablity to guarantee a minimum bandwith. Thus, these gateways only support text based chats which requires low bandwidth and prove to be very robust against package delays.
Other solutions integrate all protocols and codecs into one application by using a plugin infrastructure. This avoids bottlenecks and allows to use the full feature set of the protocols integrated.
There do exist various applications (Gaim for instance) which integrate different protocols this way. But they show several obstacles:

- To support a lot of protocol variations and features cause complex user interfaces which may not fit to every use case very well. The applications tend to grow bigger and bigger while the user may need just a minimum of the included features.
- Plugins are incompatible to foreign applications. This avoids synergetic effects.
- The available tools are limited to a special use case like instant messaging and don't provide a generic integration of communication services to extend missing features. For instance, it is impossible to add a video chat feature by adding a special plugin!

This leads to the vision, which is discussed in the next section.

2. The Vision

The vision of this project is to provide a generic infrastructure which integrates all communication protocols like a plugin based solution would do, without to write an application which has to provide everything in one user interface. The idea is to think in components and services, each optimized for a special task (or role). As a component will realize the user interaction, the services will provide the technology. A service-based architecture interconnects all of them to fulfill a given task.
For instance, a user wants to connect a distant person by selecting him within an address application. Depending on the contact information he selected (like telephone numbers or mail or instant messenger addresses), the address application is requesting a communication connection (see fig. 1), using OpenCDI. “CDI” stands for “Communication Desktop Interface” and describes the interface to connect lightweight components or complete applications to the service architecture.

A service manager takes the request and uses an internal index of available services and its dependencies to handle it. For instance, if the user selects a telephone number, it may allocate a SIP-Softphone and the right audio service (which provides one or more codecs to encode/decode the audio content stream) to establish the connection. Different backends would provide the needed functionality. Secondly, one or several user space components will be started to provide a visualization to show the ongoing operations and to interact with the user. Thus, the user is able to interrupt the process as the destination is currently unavailable.

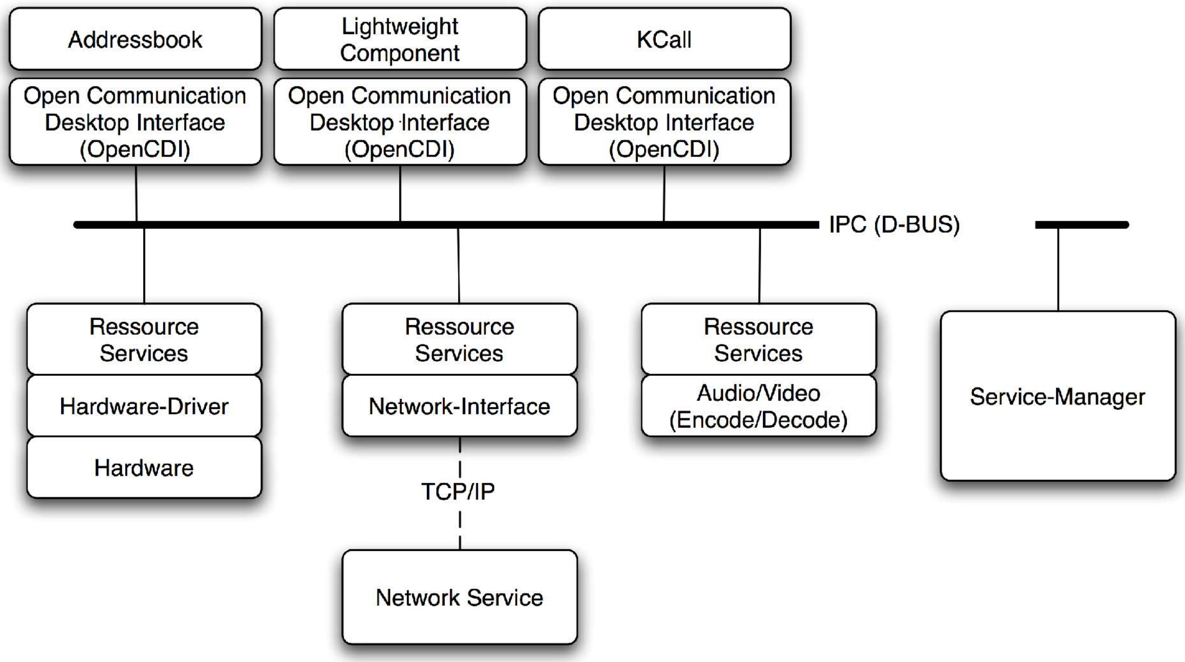


Figure 1: Final Architecture

The important aspect of this idea is to permit the user to decide which visualization components or applications he likes and which may fit into his workflow, but to decouple the underlying protocols and ressources by using an open architecture as a platform which integrates everything. The hope is to start a process which consolidates available solutions and uses synergetic effects without forcing the user to take one big application.
In this concept, all services and components are connected to an interprocess communication framework like DBUS as it provides the ability to transmit the signalling and the service requests between user and kernel space backends. This allows a maximum flexibility. The content stream will be transmitted via socket based communication channels which are created between the components directly on demand (peer-to-peer concept).

To summarize the motivation or aim of this project:

- Whether a user just needs a simple dial pad to telephone, whether he want to use a more comfortable address component or a real telephone (connected via USB or network) to call a distant person, should not minimize the capabilities and services he is able to use[1]! The service manager just needs the right services and components to fulfill a request. Or it will reject the request if it is not solvable, as a video call does not make sense if no video-camera service is available or no component is available to show the video content.
- If a new service is added, its features will be available for every tool, immediately.
- Additionally, developers need simple but efficient interfaces to integrate their services or components easily, otherwise the open-source community will not accept it.
- In the future, this service architecture should solve everything, which is related to communication.

To realize this challenging task, a first example implementation is needed to start the community process. This implementation has to be as small as possible to show the idea. But it has to work successfully to start an open discussion and to get people to work for this project. The focus will be the integration of computer based telephony, as stated in the next section.

[1] Ok, he has to use something different than a numeric dial pad to enter a jabber address!

3. Related Projects

Various projects exist which are related to this project. The following section describes the Telepathy project which provides a working proof of concept implementation to start with and a dedicated community. Following sections introduce projects which provide useful key technologies.

3.1. Telepathy

Project Homepage: http://ipcf.freedesktop.org/wiki

The aim of this freedesktop project is to provide a D-BUS based framework that unifies all forms of real time conversations. Thus, the ambition of this project is very close to the vision stated in this paper, but they are focused on soft-phones and gnome. They provide a proof of concept implementation based on python and a D-BUS specification that looks like a good base for further extensions.

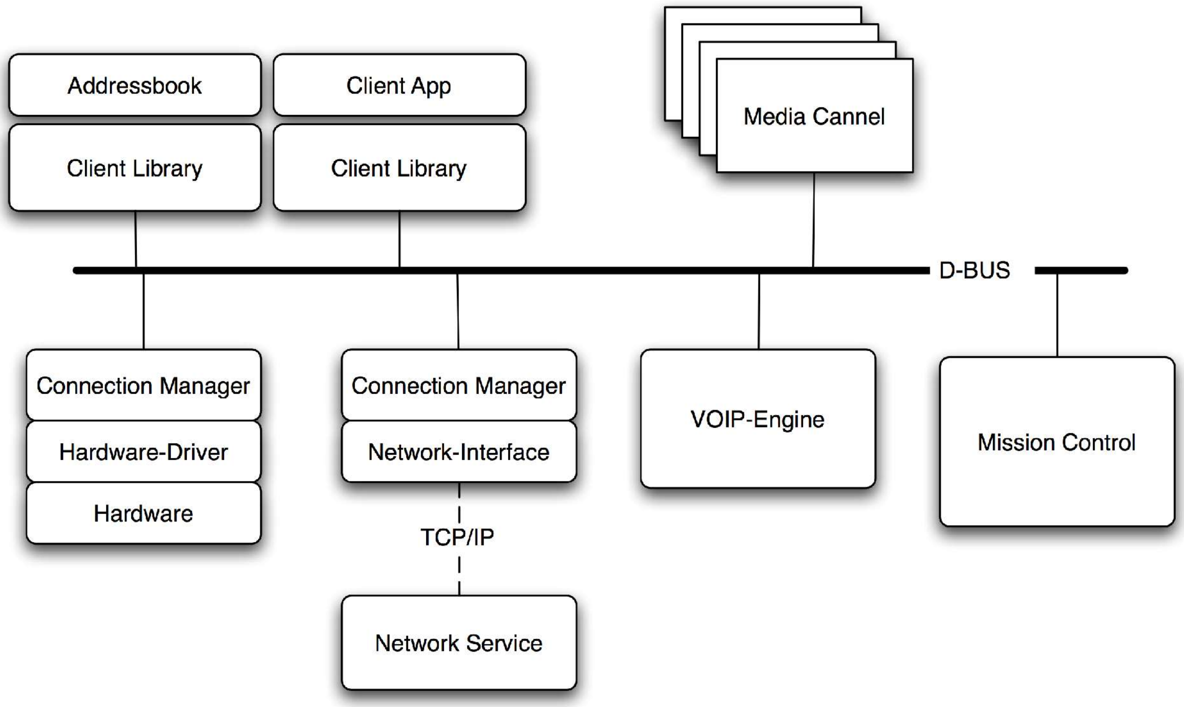


Figure 2: Telepathy

Telepathy is already supported by an active community and leaded by Robert McQueen who is very interested in our assistance. The existing D-BUS specification will be used and extended to meet our requirements. As we would provide a C++/KDE integration by designing a Qt based client library (called OpenCDI) and a C/C++ backend interface (for connection managers), they would benefit from our effort. Connection managers created by the OpenCDI project will be available to the community and therefore usable within the Telepathy framework without further modifications.

3.2. MOTUIM/DesktopIntegrationSIPIM

Project Homepage: https://wiki.ubuntu.com/MOTUIM/DesktopIntegrationSIPIM

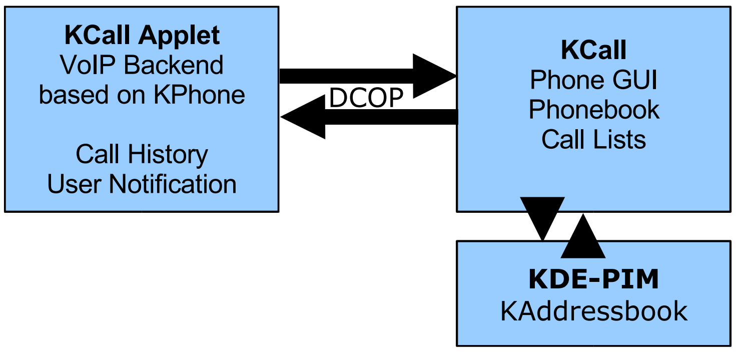
This project is in an early state, thus it does not provide any sources or draft specifications. Its motivation is similar to ours and it is closely connected to the Telepathy project. Their first milestone is to integrate SIP/IM into the (K)Ubuntu Desktop.

3.3. KCall

Project Homepage: http://kcall.basyskom.org

KCall is a telephone GUI offering CTI functionality and is integrated in KDE's Kontact (http://kontact.kde.org). The software accesses KDE's addressbook and thus groupware servers. KCall comes together with a KDE applet containing a SIP softphone implementation (kphone).

KCall and the applet communicate by using DCOP (KDE's Desktop Communication Protocol). The API for this is considered as functional and was the first testcase for further development. Currently KCall doesn't use the OpenTAPI framework, because OpenTAPI is not ready for use yet. KCall is prepared for that though.



KCall was implemented by Mike Hauth during his internship at basysKom. The sourcecode can be found in KDE's SVN in trunk/kdenonbeta/kcall.

3.4. OpenTAPI Prototype

Currently we have a working prototype of an OpenTAPI framework. This prototype was implemented by Malte Böhme as a “summer of code” project which was initiated by Google. It will be the base for the future development of OpenCDI.

It contains a SIP client at the moment, based on Malte's own implementation of the SIP specification. It was driven towards a framework for softphones. Right now there is no GUI using the framework.

The sourcecode is available in KDE's SVN in branch/kdepim/kcall

4. OpenCDI Project: Description and Bias

The aim of this initial project is to start with a platform independent telephony framework for the implementation of computer telephony integration (CTI) and telephony applications. It provides a simple method to access conventional PSTN hardware as well as IP telephony setups. The applications use OpenCDI as a desktop service interface. Thus it represents the interface to a local communication middleware.

The main infrastructure of OpenCDI will be implemented with libraries that are part of the upcoming LSB 3 standard and that are available on other platforms like Windows and Linux Embedded as well. This includes libraries such as libstdc++, glib and Qt. We intend to create a framework that is compatible with Telepathy as much as possible, so that we can share communities. We want use the existing IPC specification as it is available on freedesktop.org and want to enable the use of already existing Telepathy backends. Where necessary we will extend the existing framework and specifications and discuss this with the Telepathy project.

The communication protocol used by the applications will be DBUS using the protocol specification provided by the Telepathy project. The architecture of the framework will consist of a plugin structure which can be extended easily. These plugins (called backends) provide resource services which are handled by the service manager. One of them will provide an adapter which accesses PBX systems (CTI 3rd party control). Others contain a softphone implementation (CTI 1st party control) or a remote control of a hardphone (CTI 1st party control). Additional backends will provide services for audio subsystems, audio codecs, video codecs, media protocols, etc. The backend interfaces will be discussed with the Telepathy project, so that we can make sure Telepathy backends work with OpenCDI and vice versa.

The backends we would like to be available are the following. The crucial ones are marked with a star:

3rd Party Control:
- Asterisk Connector
- Gigaset Connector using the gigaset kernel driver

Signalling for Softphone, Connection Manager:
- SIP *
- H.323

Audio Backends:
- OSS, alsa, jack, etc
- gstreamer? *
- maybe multimedia framework of the different desktops

Audio Codecs:
- G.711a *
- GSM
- speex

Video Codecs:
- theora

RTP:
- RTP *
- SRTP

Instant Messaging connections:
- Jabber
- ICQ
- ...

UI Applications using the framework:
- KCall *
- KPhone
- GnomeMeeting
- Kopete
- KAddressbook
- ...

4.1. Roadmap

In order to implement the OpenCDI framework, we want to proceed with the following work packages:

- Analysis of the Telepathy implementation in comparison to the existing OpenTAPI implementation
- analysis of other related solutions, such as the asterisk project
- implementing the basis framework using C++, Qt and D-BUS
- implementing user-space APIs
- adapt and implement tests and example implementations such as KCall and backends
- integrate existing backends and maybe further backends
- test and debug
- document the backend and application APIs

The major goal of this project is, to have a working infrastructure ready, usable by applications and with the first backends available. The subset of functionality which will be available at least includes the usage of softphone backends and 3rd party CTI control.

5. License

KCall, OpenTAPI and Telepathy are free software and are released under the LGPL. KPhone is released under GPL as well as Kontact.

6. Involving the Community

First of all, we are going to coordinate our work tightly with the Telepathy project to stay compatible and guarantee interoperability.

After implementing the service architecture and extending OpenTAPI towards OpenCDI the project can be extended in two ways:

- One way is by providing telephony backends. This can be softphone implementations, connectors to conventional telephony systems (e.g. PBXs) or VoIP systems.
- The second way is to use OpenCDI by other GUI applications and special applications.

6.1. Backends

The Telepathy project provides only few backends right now, consisting of a SIP backend and a video framework.

We are in contact with the developers of KDE's messaging client Kopete as they are planning a rewrite of their protocol backend structure. They are aware of the Telepathy project and might want to cooperate here with regard to the backends, mainly for instand messaging and a Skype connector. They already factored out the kimproxy daemon that provides presence information to applications in the same way we want to provide telephony services with OpenCDI. Telepathy specifies a similar functionality.

Regarding new backends, we also plan to contact KPhone and GnomeMeeting as soon as we have the framework ready in order to integrate the Kphone SIP implementation and a H.323 backend. KPhone is right now being moved to sourceforge and a more open development style is being implemented. This seems to be a good chance to cooperate with them.

Someone who wants to use KCall and OpenCDI to control his mobile phone has also contacted us.

6.2. GUI Applications

Most VoIP implementations also support chatting (like SIP) and a later version of OpenCDI can provide that functionality. Kopete is interested in making use of these VoIP/Chat protocols. Right now they only provide Skype through Skype's DBUS interface.
We are also interested in talking with gnomemeeting to share implementations. As gnomemeeting is already planning to work with the Telepathy framework, we will benefit from first synergy effects.

Besides opensource software there is also commercial interest in OpenCDI. basysKom has been in contact with ISVs looking for a TAPI replacement when porting their applications from Windows to Linux. Usually this is about CRM or groupware applications. Specialized software for business processes might also need CTI functionality.
Furthermore OpenCDI can be of interest for CallCenter applications. We hope to get companies involved once the framework is ready.

Once the basic framework is released, we hope to be able to convince commercial users to further support this project. By integrating Asterisk into this framework this architecture will hopefully generate additional attention for commercial use cases.

6.3. Community Platform, SVN

In order to involve the community we want to provide a community platform including documentation, mailinglists, access to the source code repository, etc. This should be synchronized with the needs of the Telepathy developers, as a coherent presentation to the community is desired. We will consider moving OpenTAPI out of KDE's SVN to a new location outside like freedesktop. This makes it possible for people without KDE SVN account to contribute.

6.4. Community Events

In order to grow the OpenCDI & Telepathy community we want to take part in several community events and also want to organize hackatons:

Fosdem, February
taking part in VoIP track, meeting with Mark Spencer, invite Telepathy people, try to get in contact with Gnome people, talk to KDE people (PIM, Multimedia)

KDE Multimedia Meeting in the Netherlands, May
maybe send someone there, to discuss the codec/audio/video backends

Hackaton in Darmstadt, June/July?
Once the first implementation is ready, we would like to invite other projects and mainly the telepathy people to integrate and test first backends and to discuss possible framework extensions.

Conferences we can present at:
Linuxtag, May (Deadline passed already, maybe too early)
Guadec (GNOME conference), May? (maybe too early)
akademy (KDE conference), October?

7. Summary

OpenCDI will enable usage of CTI functionality on Linux and will be extended in further projects to a service architecture for multimedia based communication. By integrating our results into the Telepathy project, we are going to strengthen an already existing community process. Additionally, both projects will benefit from synergy effects as the common D-BUS specification will permit interaction and intercommunication between Gnome,KDE and its backend services.
We propose to Stichting NLnet to support the implementation and advertisement of the OpenCDI project. See enclosed document concerning financial details and expected time frame.

Darmstadt, 03/27/2006

Eva Brucherseifer,
Stefan Eilers