Communication between desktop and web applications

Author: Manuel Rego Casasnovas
Contact: mrego@igalia.com
Date: 20/11/2009

Abstract: Nowadays, everybody uses web applications. Despite of their advantages, like being available at any place with Internet, they have some usability problems with regard to common desktop applications.

This article tries to analyze the possible solutions in order to interconnect desktop and web applications, centered in the GNOME platform and using RTM-GLib as example and study case.

Table of Contents

Introduction

There are a large amount of web services in the Internet. A general definition of web service said that it is a system which provides an interface to allow interaction between machines over a network.

As time passes more and more web applications provide some kind of API in order to access their services. This number is growing and it seems that will keep growing for some time. Some examples of these applications: Flickr, Facebook, Twitter, etc.

Web applications has some advantages compared with desktop applications:

- All the data is shared and in a centralized place.
- They do not need any special configuration or installation, just a simple web browser is enough.
- Access from anywhere.

Talking about desktop applications it is worth to highlight some strengths:

- They are faster.
- Usually they have better and richer user interfaces.

There are different trends nowadays, some focused on develop applications to run in a browser, and other related with the development of desktop or mobile applications that uses web services.

In this article we will talk about the latter group. The main problem here is the connection between desktop applications and these web services. That is not solved yet in a common way for all desktop applications.

In that moment, every desktop application develops its own solution to access to some of these remote web services. Having to fight again and again with the same issues related with the implementation.

Usually these web applications provide some API kit (or libraries) to make more comfortable the access to these web services from the desktop (or other web applications). That helps to share some code, but it is not a final solution thinking in the whole integration between desktop and web.

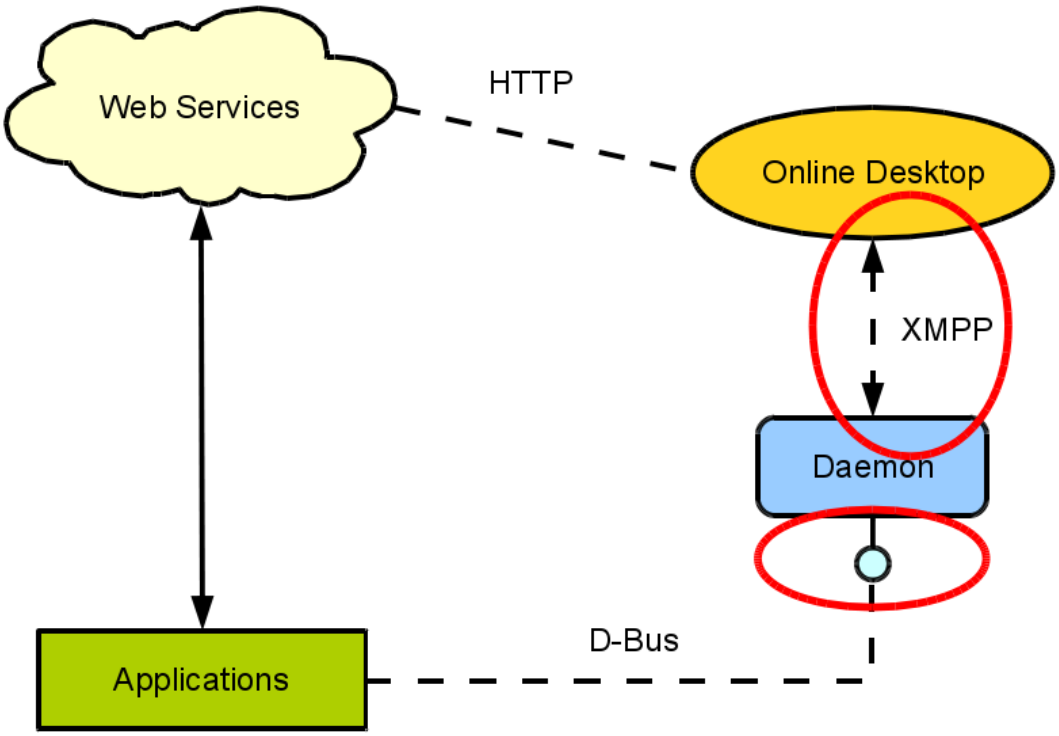
State of the art in the GNOME platform

There are different GNOME projects trying to fix the issues related with access to remote web services, and trying to provide a common solution for all the desktop applications. In this point, we are going to briefly review three of these projects: Online Desktop, Conduit and Mojito.

There is also a page at live.gnome.org (GNOME Integration with Online Services) that tries to aggregate all the information about the different technologies that allows to integrate desktop and web applications. The idea is to collect all the different libraries to define a common way to access web services in the future GNOME 3.0. When people expect a better communication of the whole desktop with the most used web services.

Online Desktop

Online Desktop was an important initiative that tried to join the desktop applications and the web services. The main idea is use desktop applications to access the different web applications, instead of a just a simple web browser.



Online Desktop architecture overview

One important part is the a central web server, written in Java, which manages the access to the different web services. This main server would be in charge to interact with the web applications, implementing the needed interfaces to do this work. This main server also store some user data like web services accounts, configuration of desktop applications, etc. Even it could end up saving passwords of web applications.

This is a great idea, because all the work needed to connect with any web application would be shared in the central server. Then, when some web application changes its API, ideally, just this server should be modified. Moreover, from the final user point of view, the most important feature is that he just need to authenticate one time (against the central server) and then all his data and services would be available. Thus, the user should not care about which computer is using, because of all the information will be stored in the main server.

From the point of view of desktop applications, *Online Desktop* provides a daemon, which defines some D-Bus methods and signals, to be used by them in order to interact with the different web services. This daemon is connected to the main server using the

XMPP protocol, which means that there is a two-way communication between the daemon and the central server. This allow to the server send signals to desktop applications when some data has changed. Even, the main server could poll the different web services from time to time, and then send signals to desktop applications.

Some advantages of this solution:

- It is really comfortable for desktop applications, because they just need to know a D-Bus interface to interact with web applications.
- The logic to interact with web service is just implemented once in the main server.

Some drawbacks:

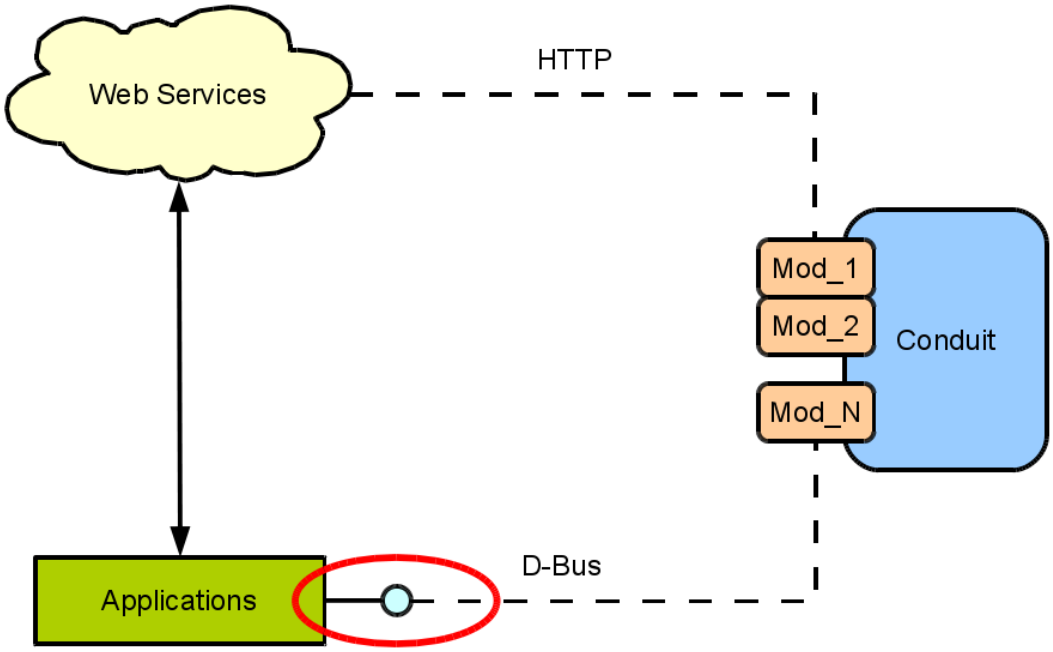
- The central server could become a bottleneck if there are a lot of users. Perhaps, some scalability problems could appear.
- Security issues that could appear because of a lot of user information is stored in the main server.
- The project is currently abandoned.

Online desktop was started by the middle of 2007 and just lasted around one year. It was started by some important GNOME hackers (Havoc Pennington, Owen Taylor and Colin Walters) working in Red Hat in that time. By the middle of 2008 the project started to decrease its activity, and it is stopped since one year ago.

Conduit

Conduit is a generic synchronization application for GNOME. It is not just a bridge between desktop and web applications. For example, it allows you to synchronize: two folders on your system, your music library with your iPod, your contacts with your mobile phone, etc. *Conduit* allows to create different modules for each kind of synchronization.

By the way, *Conduit* supports the SyncML protocol, a synchronization standard for mobile devices. That allows to share your contacts between your computer and your mobile phone.



Conduit modules overview (from the point of view of desktop and web communications)

As it has a wider scope than just web services, it is a completely different approach. However, it adds a new idea to fix the problem related with web synchronization.

In the case of synchronize applications and web services it uses the D-Bus interfaces provided by desktop applications, or just use the files stored by these applications to get the information. Which means that for every application it should know its D-Bus interface or how are formated the files it uses. So, if something changes in the application implementation, the *Conduit* module should be modified.

Desktop applications do not know anything about *Conduit*, so they do not know if somebody is doing a synchronization with their data. This can be an important source of problems, moreover the functionality that *Conduit* could provide is not the as powerful as if the proper applications directly access to web services.

If we talk about desktop and web synchronization the main advantages:

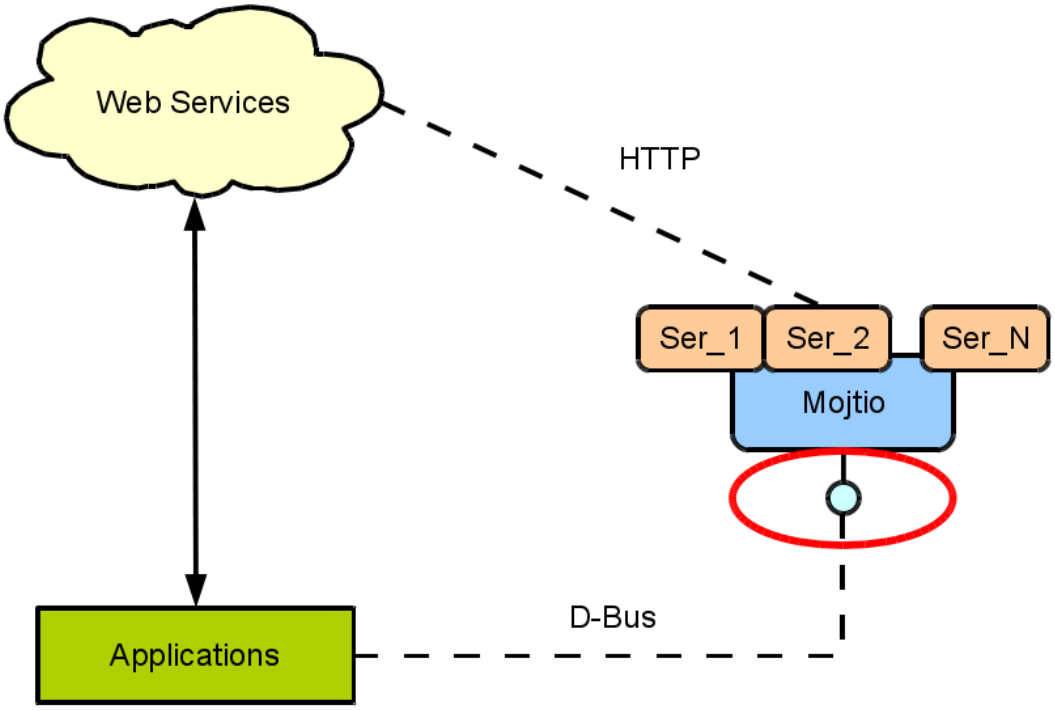
- There are already some source code to access to web services APIs in different modules.

Among the disadvantages:

- It is not thought for this specific purpose of connecting desktop and web applications.
- Depends on D-Bus interfaces or file formats of desktop applications, that could change in any moment.
- Desktop applications do not manage the interaction with the web services.

Mojito

Mojito is a project that tries to bring social networks to the desktop. It is new project and is part of the Moblin project developed by Intel.



Mojito structure overview

For the moment it is just a social network aggregator, that pulls in content from some web services. *Mojtio* provides a D-Bus interface that desktop applications can use to get data from these web applications. Moreover, it also provide a method to update your status on web services that supports it. That is something similar to the *Online Desktop* approach.

To sum up, you can say that it is a centralized place to manage all the interaction of the desktop with different web applications. All the logic needed to connect to these web services is implemented in *Mojito*. Furthermore, it polls the different web service time to time, and send D-Bus signals when something changes.

Advantages:

- From the point of view of desktop applications that wants to interact with some web service, it is quite similar to the *Online Desktop* approach.
- All the logic related with web services is implemented in the same application, so some common code to access the web could be easily shared.

Disadvantages:

- For the moment it just support a few functionalities, and it is not sure that this will expand its current scope.
- It is still in an early stage and it is not part of the GNOME project yet.

Goals

After review the current situation related with this topic, the idea is to create a GNOME based library in order to provide access to the Remember The Milk web application. And after that, try to provide through *Moijto* (via D-Bus) all the methods needed to interact with this web service. Making easy the access to Remember The Milk for any application.

The idea to develop a library for this service (and not any other), it is mainly because of there was not exist a good library written in C to access to this web application. Moreover, during the Master on Free Software, a desktop client for Remember The Milk has been developed. This application could end up using the library (directly or through *Mojito*) to make the interaction with the web service more simple.

Remember The Milk is a task and time management web application. It tries to improve the user experience regarding this issue. It provides a lot of features allowing to users to create multiple task lists, organize these tasks by tags, specify the location of a task, share tasks between groups of people, etc. Almost all these features are also available from his API web.

The library: RTM-GLib

The library that has been created is called *RTM-GLib*. This library provides access to the API of Remember The Milk.

The library webpage was created at GNOME wiki (http://live.gnome.org/RTMGlib). There you can find all the information about the project, e.g., latest news, release notes, roadmap, etc.

It is worth pointing out that after the announcement of the first release of *RTM-GLib* (version 0.1.0), that was made both in GNOME and Remember The Milk mailing lists, the library has become one of the recommended API kits at Remember The Milk webpage.

Dependencies

RTM-GLib just depends on *librest*, a library designed to make easier the access to web services that claims to be REST, like the Remember The Milk one.

This library was written using *libsoup* and *libxml* with the aim to make very simple the access to any web service. Without the need to manually process the response from the web application when you do a request. *librest* is part of the Moblin project and has been developed by Rob Bradford and Ross Burton. The authors of *Mojito* project, that also uses *librest*.

The main reason to use this library, instead of the common *libsoup* (or *libcurl*) and *libxml* combination, is its simplicity. Thanks to this library the web services requests

are really simple to process, therefore all the effort is put in the implementation of the particular stuff of Remember The Milk. Futhermore, it fulfills all the needs to develop *RTM-GLib*.

Development

RTM-GLib is written in C, as most of the GNOME libraries and applications.

As any GNOME library it uses GLIb and GObject as base. It defines some classes that represent the different objects needed for an application that wants to interact with Remember The Milk.

Besides, the library uses Check (a unit testing framework for C), in order to check the behaviour of the different classes defined by the library.

In addition GTK-Doc was used to define the API reference, in order to allow that anybody can be able to start to use the library without any problem.

Furthermore, during the development of *RTM-GLib* a Git repository (http://gitorious.org/rtm-glib/) has been used. And two releases has been done: the first release (0.1.0) and a maintenance release (0.1.1), fixing some bugs. Trying to follow one of the free software maxims, "release early, release often".

License

The license chosen for *RTM-GLib* is the GNU Lesser General Public License version 2.1. The main reason to select this license is just because most of the GNOME libraries use this license, so in some way it is more GNOME-compliant.

Moreover, this kind of license (non-copyleft) has a lot of sense if you are developing a library, or toolkit, and your idea is to just spread the use of your library. Without worrying if public or privative applications are developed on the top of it.

Roadmap

RTM-GLib is currently usable to access to the main part of Remember The Milk API methods (some of them are still pending to be implemented).

Anyway, as in any software project a lot of improvements could be done, the most important in the short term could be:

- *Finish the implementation of all the methods provided by Remember The Milk.*
- *Add asynchronous calls support to the library. Currently the library just provide synchronous methods.*
- *Add GObject introspection support, in order to make easy the creation of bindings for the library.*

Usage example

In this section, two examples are shown in order to understand better how to use this library. The first example explains how to manage the authentication against the Remember The Milk web service. In the second example, there is just a simple call to one of the library methods that returns a list of tasks.

For more information, you can review a full example in the source code (inside the `examples` folder at file `rtm-glib-example.c`).

Authentication

- *Create a new `RtmGlib` object:*

  ```
  RtmGlib *rtm = rtm_glib_new (YOUR_RTM_API_KEY, YOUR_RTM_SHARED_SECRET);
  ```

  You will need to request an API key at Remember The Milk website.
- *Get the authentication frob:*

  ```
  gchar *frob = rtm_glib_auth_get_frob (rtm, NULL);
  ```
- *Request the login URL:*

  ```
  gchar *url = rtm_glib_auth_get_login_url (rtm, frob, NULL)
  ```

  Then the user should access to the returned URL and accept the requested permissions by the application.
- *Get the authentication token:*

  ```
  gchar *auth_token = rtm_glib_auth_get_token (rtm, frob, NULL);
  ```
- *Check that the user is properly logged:*

  ```
  gchar *username = rtm_glib_test_login (rtm, auth_token, NULL);
  ```

List of tasks

- *Call method `rtm.tasks.getList` with an `RtmGlib` object already authenticated:*

  ```
  GList *glist = rtm_glib_tasks_get_list (rtm, NULL, NULL, NULL, NULL);
  for (GList *item = glist; item; item = g_list_next (item)) {
          RtmTask *task = (RtmTask *) item->data;
          g_print ("%s", rtm_task_to_string (task));
  }
  ```

Future ideas

Mojito

The idea, from the Goals section, to integrate in some way *RTM-GLib* and *Mojito* has not been achieved yet. The main issue is related with the scope of *Mojito* project, and

if it wants to provide to the desktop applications methods to access to any web service through its D-Bus API.

Nowadays, *Mojito* is mainly centered in social networks, for example, they are adding Facebook support right now. However, it could maybe work properly for any web service. Anyway, this should be discussed with the project adevelopers (still without answer from their side).

In order to provide access to Remember The Milk it is needed to implement a new *Mojito* service. There are already a dummy service that could be used as base for the development of the new plugin, that would use *RTM-GLib* as base library.

In my opinion, the integration of *RTM-GLib* in *Mojito* is something that should remain as main goal for the future of the library, at least if it fits in the project scope after som debate with the authors. This will make easier to any application interact with Remember The Milk web service. And it would be important for the library, because this would mean that these applications will be indirectly using *RTM-GLib*, that could increase the feedback about bugs, new features, and so on. Unfortunately, for the moment, the lack of time has made impossible to advance more in this study case.

Tracker miner

Tracker is a project with the aim to collect information and meta-information about personal data. This information is indexed in order to can be easily and quickly searched. It provides both a search tool and a storage system for all this information.

From the point of view of a final user, *Tracker* is just a small applet that allows to insert any word related with something that he likes to found, e.g., some words of a document or the artist of a song. But, actually some other applications uses *Tracker* to ask about, for example, the video files in the system.

There is a daemon that is running in the computer and indexes every change that happens in the file system, storing the meta-information related with the different files in the *Tracker* store. These kind of daemons are called a **miners**.

Currently, there are only one official miner, the file system miner. However, the idea is that in the future more miners could appear, and each of them will have the responsibility to index different data sources. For example, there is already a person (Adrien Bustany) working in a Facebook miner written in Vala, that also uses *librest* to interact with the the web service. The source code of this new miner is in a branch of the *Tracker* development repository.

When more miners are available, the user could configure which miners are currently running and enable or disable any of them from a common applet. This applet will also show information about the progress of the indexation process.

Talking about the implementation details, in order to develop a new miner, it is just needed to extends an abstract class (`TrackerMiner`) and implement some methods to perform common operations such as start, stop, pause and resume the miner.

For example, a new miner to index the tasks from Remember The Milk could be developed using *RTM-GLib* as base library.

EDS backend

Evolution Data Server (EDS) provides a common backend for programs that have to work with contacts, tasks and calendar information. At the beginning it was part of *Evolution* (the default email client of GNOME project), but it is currently used for other external applications.

One interesting example, from the point of view of *RTM-GLib*, is that the calendar applet in the GNOME panel uses *EDS* to provide a list of your appointments and tasks. It would be great to have these tasks synchronized with your tasks at Remember The Milk website.

In order to develop this idea it would be necessary to create an *EDS* backend. This backend could be based in the Google Calendar backend already implemented in *EDS*.

Conclusion

Nowadays, web applications are in really good moment. For what most of people do in a computer, the wide range of available applications in the Internet are more than enough.

It seems that there is an interesting future related with the integration between desktop and web applications. Nowadays, several desktop applications are starting to interact with different web services. There are several approaches to make this communication possible, and there are a lot of technologies trying to fix this issue in some way.

The development of the library was an interesting practice in order to know how to deal with this kind of interaction problems. It also helps to improve the knowledge about the GNOME platform and how its internals works. Moreover, it is a simple example about how to fix this issue in the GNOME world. Where some ideas like proposed by *librest* or *Mojito* could become in the standard.

Just to conclude, there are certainly exciting times ahead regarding the communication between desktop and web.