GRILO
Feeding applications with multimedia content

GUADEC, The Hague, July 2010

Iago Toral Quiroga
itoral@igalia.com

Index

Integrating Multimedia Content

Integrating Multimedia Content

- Media content available in many forms:
  - Youtube, Shoutcast, UPnP, Jamendo, Podcasts, Vimeo, Last.FM, iPod, local drives, etc.
- We are used to consume content from many of these sources every day.
- But usually we use various applications to do that: not convenient.
- Multimedia applications today are trying to integrate more and more of these services to provide a better user experience.

Integrating Multimedia Content

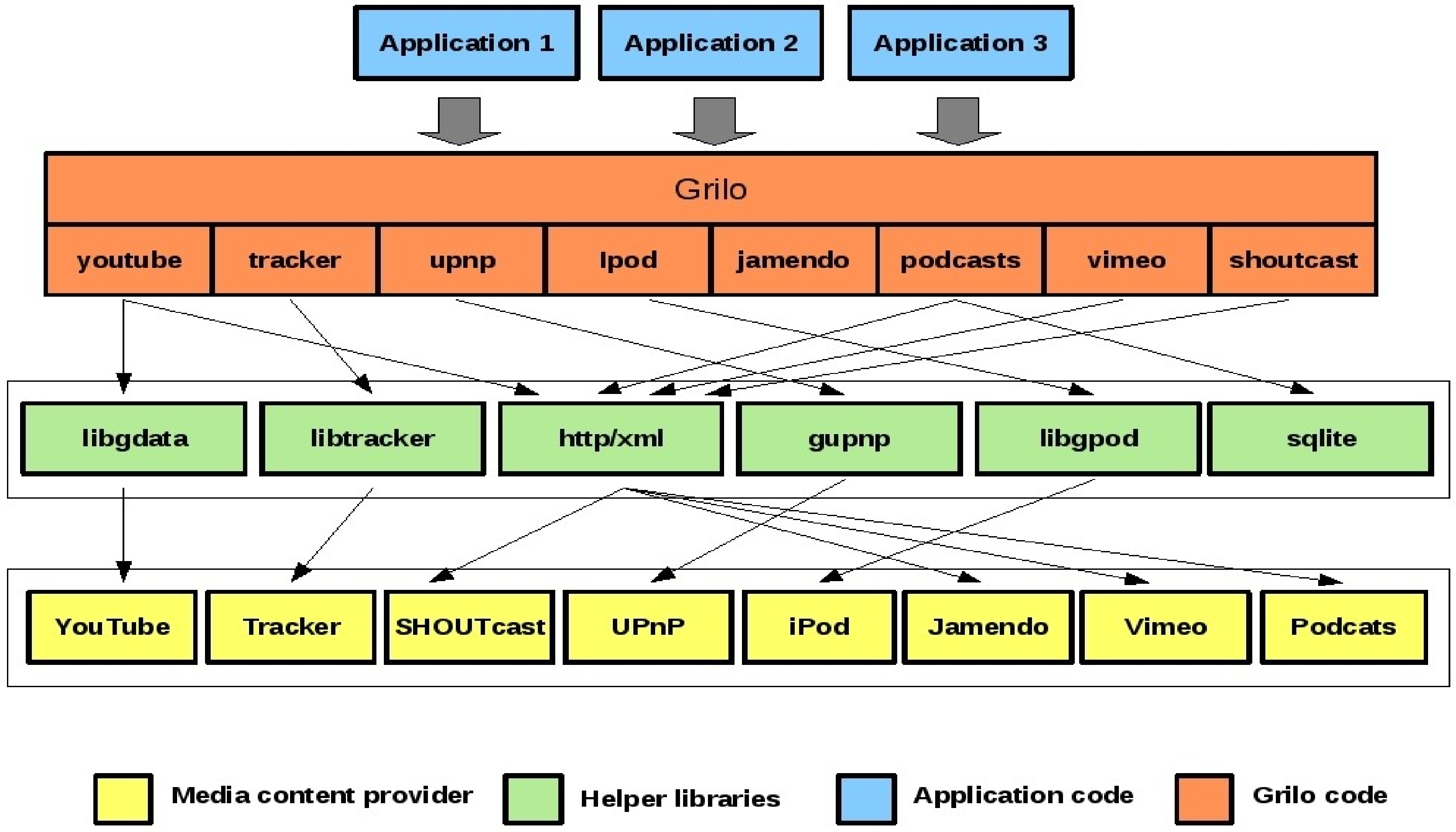
- However, these services expose different APIs / protocols, have different limitations / behaviors, etc.
- Integrating all these services in a single application requieres a lot of learning and coding.
- We have many applications doing this effort already:
  - Totem, Rhythmbox, Amarok, XBMC, etc
- But these solutions are application specific:
  - Developers cannot reuse this work directly in other projects.
  - Every application has to maintain its own solution.

Grilo: Overview

Grilo: Overview

- A framework for easing access to multimedia content.
- Application developers want to browse / search content from many services...
- ...but they don't want to know how they work internally (APIs, protocols, technologies, limitations, ...)
- Single API to access media content, hiding differences among media providers.
- Same idea as GStreamer and media formats.
- Application developers write their solution once and it will work for any service supported in Grilo.

Grilo: Overview



Grilo for application developers

Grilo: Application Developers

- ➔ Looking up available sources (signals):

```
GrlPluginRegistry *r = grl_plugin_registry_get_instance ();
g_signal_connect (r, "source-added",
                  G_CALLBACK (source_added_cb), NULL);
g_signal_connect (r, "source-removed",
                  G_CALLBACK (source_removed_cb), NULL)

/* Use one of these to load all plugins, plugins
   in a specific directory or just a specific plugin */
grl_plugins_registry_load_all (r);
grl_plugin_registry_load_directory (r, path);
grl_plugin_registry_load (r, path);
```

Grilo: Application Developers

- ➔ Looking up available sources (source id):

```
GrlPluginRegistry *r = grl_plugin_registry_get_instance ();
GrlMediaPlugin *p = grl_plugin_registry_lookup_source (r, “grl-jamendo”);
GrlMediaSource *s = GRL_MEDIA_SOURCE (p);
...
```

- ➔ Looking up available sources (capabilities):

```
GrlPluginRegistry *r = grl_plugin_registry_get_instance ();
GrlMediaPlugin **p =
    grl_plugin_registry_get_sources_by_operations (
        r, GRL_OP_BROWSE | GRL_OP_SEARCH, TRUE);
for (gint i = 0; p[i] != NULL; i++) {
   GrlMediaSource *s = GRL_MEDIA_SOURCE (p[i]);
   ...
}
```

Grilo: Application Developers

➔ Browsing

```
GList *keys = grl_metadata_key_list_new (GRL_METADATA_KEY_TITLE,
                                         GRL_METADATA_KEY_CHILDCOUNT,
                                         NULL);
guint browse_id =
  grl_media_source_browse (source,         // MediaSource object
                           container,      // Container to browse
                           keys,           // Metadata keys to retrieve
                           offset, count, // Page info
                           flags,          // Operation flags
                           browse_cb,      // Callback for results
                           user_data);     // User data for callback

static void browse_cb (GrlMediaSource *s, guint opid, GrlMedia *m,
  guint remaining, gpointer user_data, const GError *error)
{
  if (media) {
    const gchar *title = grl_media_get_title (media);
    ...
  }
}
```

Grilo: Application Developers

- Searching

```
GList *keys = grl_metadata_key_list_new (GRL_METADATA_KEY_TITLE,
                                         GRL_METADATA_KEY_CHILDCOUNT,
                                         NULL);
guint search_id =
  grl_media_source_search (source,        // MediaSource object
                           text,          // Search text
                           keys,          // Metadata keys to retrieve
                           offset, count, // Pagination info
                           flags,         // Operation flags
                           search_cb,     // Callback for results
                           user_data);    // User data for callback

static void search_cb (GrlMediaSource *s, guint opid, GrlMedia *m,
  guint remaining, gpointer user_data, const GError *error)
{
  if (media) {
    const gchar *title = grl_media_get_title (media);
    ...
  }
}
```

Grilo: Application Developers

- Operation flags
  - GRL_RESOLVE_NORMAL
  - GRL_RESOLVE_FULL
  - GRL_RESOLVE_IDLE_RELAY
  - GRL_RESOLVE_FAST_ONLY

Grilo: Application Developers

- Other APIs
  - Query
  - Multiple Search
  - Cancel
  - Get / Set metadata
  - Resolve
  - Store / Remove

Grilo: Application Developers

- Youtube
- Vimeo
- Jamendo
- Apple Trailers
- Flickr
- Podcasts
- SHOUTCast
- UPnP
- Bookmarks
- Filesystem
- Last.fm album art
- Metadata store
- Gravatar

Grilo: Application Developers

- GNOME Media Server Spec:
  - http://live.gnome.org/Rygel/MediaServerSpec
- Targets:
  - Separate media providers in a separate process.
  - Consume media content over D-Bus.
  - No need for language bindings.
- Rygel-Grilo (name subject to change)
- Plugins for Totem and Rhythmbox

Grilo for plugin developers

Grilo: plugin developers

- Plugin types:
  - Media Providers (MediaSource)
    - Provide media content
    - Youtube, Jamendo, Shoutcast, etc.
    - browse, search, query, store, remove, etc
  - Metadata Providers (MetadataSource)
    - Provide extra metadata
    - Album art, ratings, imdb, etc
    - resolve

Grilo: plugin developers

- Creating a MetadataSource plugin (class_init)

```
static void
grl_lasftfm_albumart_source_class_init (GrlMetadataStoreSourceClass * klass)
{
  GrlMetadataSourceClass *metadata_class =
      GRL_METADATA_SOURCE_CLASS (klass);

  metadata_class->supported_keys =
    grl_lastfm_albumart_source_supported_keys;
  metadata_class->key_depends =
    grl_lastfm_albumart_source_key_depends;
  metadata_class->resolve =
    grl_lastfm_albumart_source_resolve;

  ...
}
```

Grilo: plugin developers

- ➔ Creating a MetadataSource plugin (supported_keys)

```
static const GList *
grl_lastfm_albumart_source_supported_keys (GrlMetadataSource *source)
{
  static GList *keys = NULL;
  if (!keys) {
    keys = grl_metadata_key_list_new (GRL_METADATA_KEY_THUMBNAIL,
                                      NULL);
  }
  return keys;
}
```

Grilo: plugin developers

- Creating a MetadataSource plugin (key_depends)

```
static const GList *
grl_lastfm_albumart_source_key_depends (GrlMetadataSource *source,
                                        GrlKeyID key_id)
{
  static GList *deps = NULL;
  if (!deps) {
    deps = grl_metadata_key_list_new (GRL_METADATA_KEY_ARTIST,
                                      GRL_METADATA_KEY_ALBUM,
                                      NULL);
  }

  if (key_id == GRL_METADATA_KEY_THUMBNAIL) {
    return deps;
  }

  return  NULL;
}
```

Grilo: plugin developers

- Creating a MetadataSource plugin (resolve)

```
static void
grl_lastfm_albumart_source_resolve (GrlMetadataSource *source,
                                    GrlMetadataSourceResolveSpec *rs)
{
    artist = grl_data_get_string (GRL_DATA (rs->media),
                                  GRL_METADATA_KEY_ARTIST);
    album = grl_data_get_string (GRL_DATA (rs->media),
                                 GRL_METADATA_KEY_ALBUM);

    thumb_uri = /* Use album & artist to get thumbnail info */

    grl_data_set_string (GRL_DATA (rs->media),
                         GRL_METADATA_KEY_THUMBNAIL,
                         thumb_uri);

    rs->callback (rs->source, rs->media, rs->user_data, NULL);
}
```

Grilo: plugin developers

- Creating a MediaSource plugin (class_init)

```
static void
grl_podcasts_source_class_init (GrlPodcastsSourceClass * klass)
{
  GrlMediaSourceClass *source_class = GRL_MEDIA_SOURCE_CLASS (klass);
  GrlMetadataSourceClass *metadata_class = GRL_METADATA_SOURCE_CLASS (klass);

  source_class->browse = grl_podcasts_source_browse;
  source_class->search = grl_podcasts_source_search;
  source_class->metadata = grl_podcasts_source_metadata;

  metadata_class->supported_keys = grl_podcasts_source_supported_keys;
  ...
}
```

Grilo: plugin developers

- Creating a MediaSource plugin (search)

```
static void
grl_podcasts_source_search (GrlMediaSource *source,
                            GrlMediaSourceSearchSpec *ss)
{
  GList *medias =
    query_pocasts_by_text (ss->text, ss->keys, ss->skip, ss->count);

  if (!medias) {
    ss->callback (ss->source, ss->operation_id,
                  NULL, 0, ss->user_data, NULL);
  } else {
    guint n = g_list_length (medias);
    while (medias) {
      ss->callback (ss->source, ss->operation_id,
                    GRL_MEDIA (medias->media), --n,
                    ss->user_data, NULL);
      medias = g_list_next (medias);
    }
  }
  ...
}
```

Demo

Grilo: Resources

- Wiki:
  - http://live.gnome.org/Grilo
- Git repositories:
  - git://git.gnome.org/grilo
  - git://git.gnome.org/grilo-plugins
- IRC:
  - grilo @ GIMPNet
- Mailing list:
  - http://mail.gnome.org/mailman/listinfo/grilo-list
- Bugzilla:
  - http://bugzilla.gnome.org
  - Category: Other, Product: grilo

?