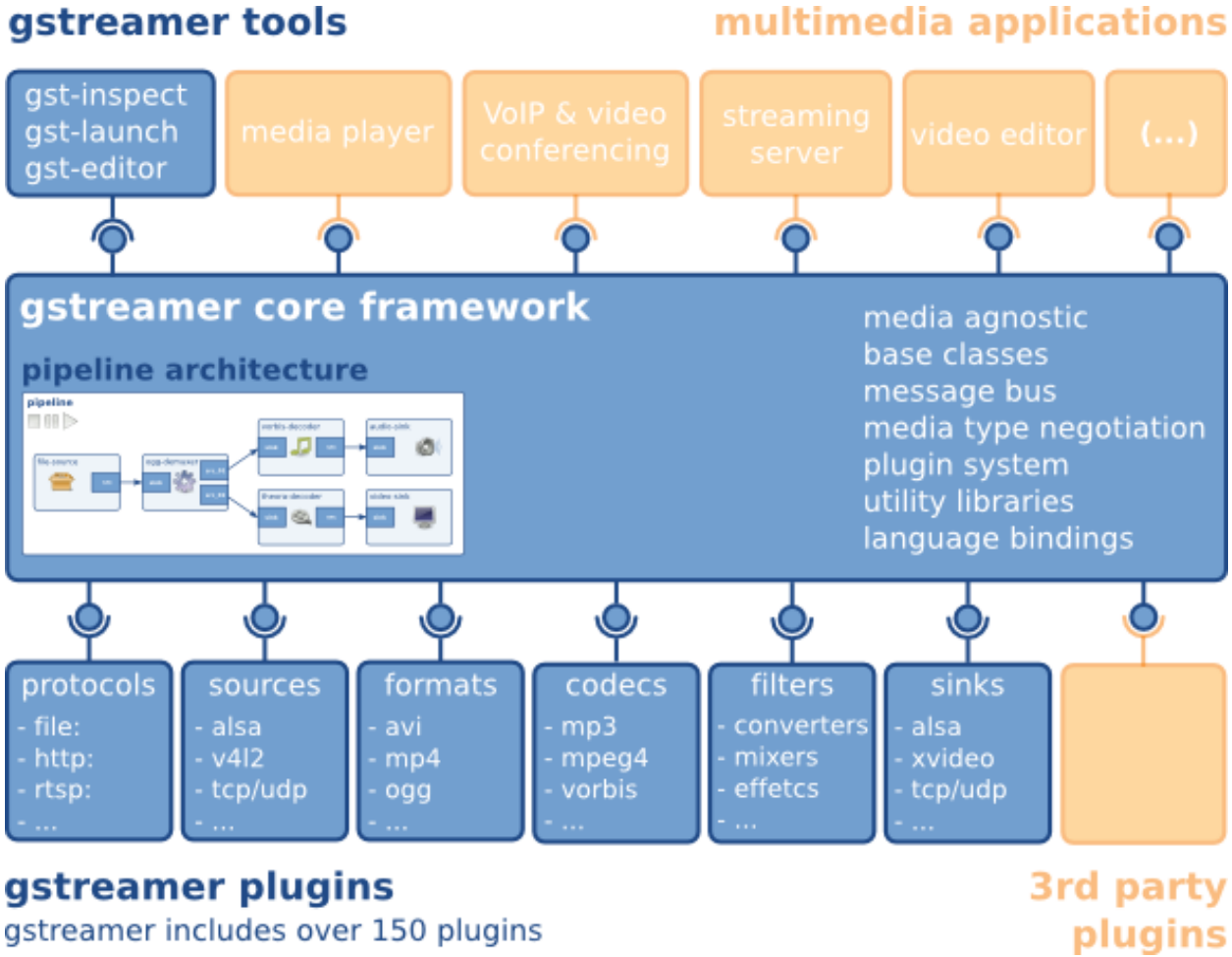
M1 OIM TP3

1. PRÉSENTATION DE GSTREAMER

Gstreamer est un « framework multimédia » (bibliothèque logicielle de gestion globale du son et de l'image) bien intégré dans certains composants Linux, mais il est aussi présent sous Unix, Windows, MacOS et Symbian.Il permet de gérer des données multimédia (son et vidéo) de bout en bout : de l'acquisition de la source (fichier, flux réseau, webcam, micro...) au traitement (effet vidéo, audio, encodage) à la diffusion (sur l'écran, dans un fichier, sur le réseau).

Il se base sur une architecture modulaire. Il est composé d'un cœur (GStreamer) et de plugins (Base, Good, Ugly, Bad). Il est développé en C mais il existe de nombreuses librairies pour appeler GStreamer à partir de logiciels développés dans d'autre langages : C, Java, Python...



Voici la liste des composants GStreamer :

- gstreamer : le cœur
- gst-plugins-base : un ensemble d'éléments indispensables
- gst-plugins-good : un ensemble de greffons de bonne qualité sous licence LGPL
- gst-plugins-ugly : un ensemble de greffons de bonne qualité qui peuvent poser des problèmes sous certaines distributions
- gst-plugins-bad : un ensemble de greffons qui nécessiteraient plus de qualité

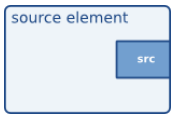
– gst-python : les liaisons avec python

Pour vérifier que GStreamer est bien installé sur linux, vous pouvez demander le listing des greffons et focntionnalités disponibles :

```
$ gst-inspect
gio:  giosink: GIO sink
gio:  giosrc: GIO source
...
Nombre total : 219 greffons (1 élément de liste noire not shown),
1111 fonctionnalités
```

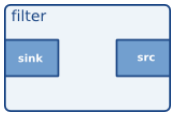
Un pipeline est constitué d'elements qui sont connectés entre eux. Les différents éléments (`elements`) que l'on peut retrouver dans le pipeline sont :

– les éléments sources (source element) :



– les éléments de sorties (sink element)

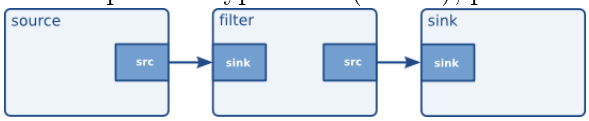
– les éléments de filtrage (filter) :



Les connecteurs (`pads`) sont les entrées/sorties des éléments. Ils servent à interconnecter les éléments entre eux. Ils peuvent inclure des fonctions de vérification afin de s'assurer que le format des données correspond à ce qu'il attend. On peut distinguer deux sous ensembles de pad :

– les pads de type "sink", permettant de faire entrer des données dans un élément

– les pads de type "src" (source), permettant de faire sortie des données d'un élément



Il est possible d'associer plusieurs pads sinks et srcs à un même élément. Les pads sont de loin les objets les plus complexes dans GStreamer. Ils contiennent un ensemble de paramètres (statique ou dynamique) permettant de définir les données attendues ou générées.

Les `bins` sont des pipelines prédéfinis que l'on peut inclure comme de simple éléments dans une nouveau pipeline. Celà permet de simplifier grandement certaines actions complexe. `rtpbin`, qui permet d'assurer des transports sur les réseau en est un bon exemple.

2. UTILISATION GSTREAMER

2.1. **Chaîne de traitement simple.** Nous allons essayer GStreamer sur un exemple simple : lire une vidéo en utilisant le lanceur `gst-launch` :

```
$ gst-launch filesrc location=/users/minfg/farinas/M1_OIM/TP2/trailer_400p.ogg \
! oggdemux ! theoradec ! xvimagesink
```

Une fenêtre s'ouvre et la vidéo est lue :

Vous remarquerez que les commandes (greffons, plugin en anglais) s'enchaînent à la manière des tuyau Unix : le caractère séparateur est ici « ! ».

- Le premier greffon est `filesrc` : il prend en entrée un fichier multimédia (dont le nom est passé par le paramètre `location`).

Il est possible d'avoir une documentation précise du plugin et de ses paramètres :

```
$ gst-inspect filesrc
```

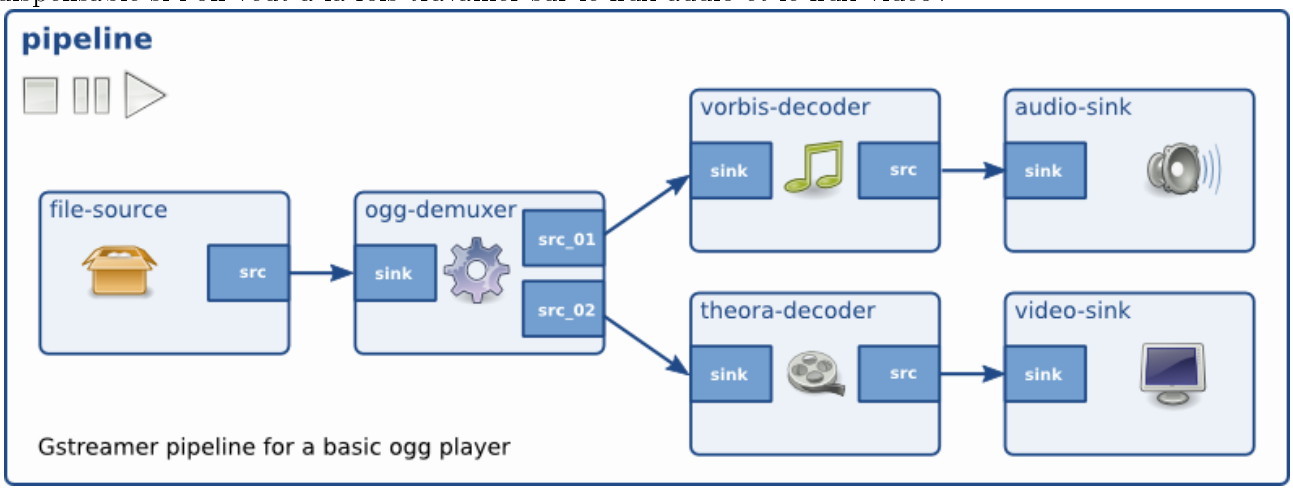
- Le deuxième greffon est `oggdemux` : il prend en entrée un flux vidéo OGG et qui permet de le démultiplexer. OGG est ici un conteneur (comme les fichiers .AVI), il encapsule le contenu. Ce greffon permet d'accéder à ses composants.
- Le troisième greffon (`theoradec`) permet d'effectuer le décodage de la vidéo dont le format est Theora (format de vidéo ouvert et sans brevets, rivalisant avec les formats MPEG4, WMV et RealVideo).
- Le dernier greffon (`xvimagesink`) prend en entrée un flux vidéo décodé (format RAW) et l'affiche sur l'écran en utilisant Xv (XFree86 video).

La logique d'enchaînement des greffons est assez intuitive. Si l'on souhaite par exemple redimmensionner l'imageavant de l'afficher, il suffit d'ajouter un greffon (`videoscale`) :

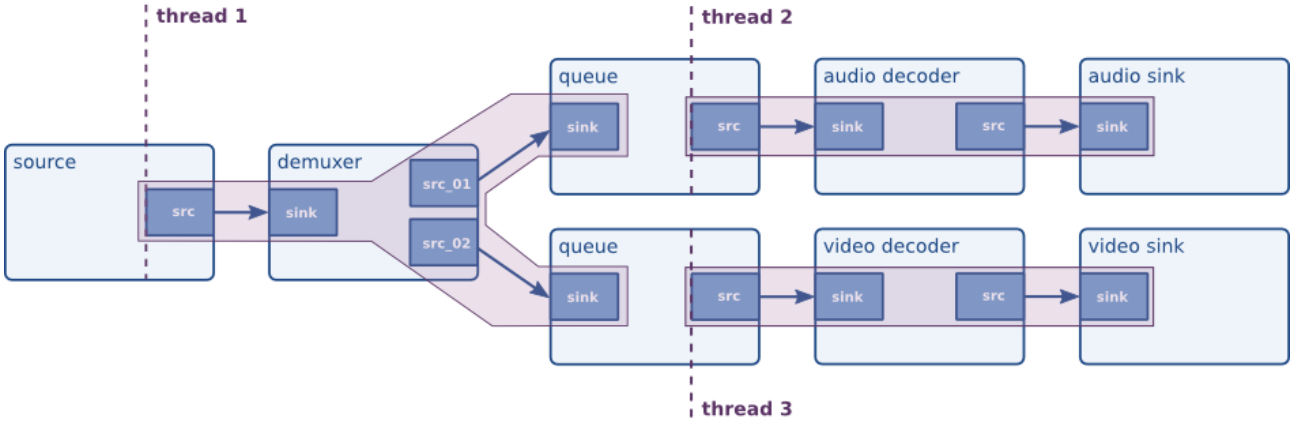
```
$ gst-launch filesrc location=/users/minfg/farinas/M1_OIM/TP2/trailer_400p.ogg \
! oggdemux ! theoradec ! videoscale ! video/x-raw-yuv,height=240 ! xvimagesink
```

- Le greffon `videoscale` adapte la taille du flux à l'élément suivant. En indiquant `video/x-raw-yuv,height=240` on contraint le flux à ces propriétés. Si l'on n'avait pas rajouté ces propriétés la fenêtre pourrait être redimentionnée à la volée avec la souris.

2.2. **Chaîne de traitement audio-vidéo.** Nous n'avons traité qu'un seul flux dans l'exemple précédent, mais il est également possible de travailler en paralèle sur deux flux : ça sera en particulier indispensable si l'on veut à la fois travailler sur le flux audio et le flux vidéo !



gst-launch implémente un formalisme pour pouvoir écrire le pipeline.



Voici un exemple d'un lecteur audio-vidéo :

```
# gst-launch filesrc location=/users/minfg/farinas/M1_OIM/TP2/trailer_400p.ogg
! oggdemux name=demux \
demux. ! queue ! vorbisdec ! audioconvert ! audioresample ! alsasink \
demux. ! queue ! theoradec ! xvimagesink
```

On peut améliorer le lecteur en laissant GStreamer sélectionner lui même les sorties audio et vidéo (autodetect) :

```
# gst-launch filesrc location=/users/minfg/farinas/M1_OIM/TP2/trailer_400p.ogg
! oggdemux name=demux \
demux. ! queue ! vorbisdec ! audioconvert ! audioresample ! autoaudiosink \
demux. ! queue ! theoradec ! autovideosink
```

2.3. **Exercices.**

(1) Ecrivez une chaîne de traitement qui permette de lire un fichier audio MP3. Vous pourrez tester sur le fichier
`/users/minfg/farinas/M1_OIM/TP2/drlp090_d027.mp3`

(2) Ecrivez une chaîne de traitement qui permette de convertir un fichier son encodé en MP3 vers un fichier on encodé en OGG.

(3) Ecrivez une chaîne de traitement qui permette de convertir la bande annonce de Big Bug Bunny en un format .AVI avec la vidéo au format MPEG2 redimensionnée par la méthode des plus proches voisins en 320 x 200 et le son encodé en MP3 en ayant baissé le volume de 25%.

Références

- GStreamer site officiel, Freedesktop, `http://gstreamer.freedesktop.org/`
- Introduction à GStreamer, le framework multimedia, Nicolas Largo, `http://blog.nicolargo.com/2009/01/introduction-a-gstreamer-le-framework-multimedia.html`