GSTREAMER un cadre pour manipuler les objets multimédia

Pierre Mathieu, Frédéric Payan

GSTREAMER
un cadre pour manipuler les objets multimédia

Pierre Mathieu
Université de Nice Sophia Antipolis
IUT Nice Côte d'Azur - Département R&T
Laboratoire I3S
Email : Pierre.Mathieu@unice.fr

Frédéric Payan
Université de Nice Sophia Antipolis
IUT Nice Côte d'Azur - Département R&T
Laboratoire I3S
Email : Frederic.Payan@unice.fr

***Résumé*—La manipulation de données multimédia est souvent lourde et difficile car en dehors du mp3, connu du grand public, les flux de sons, d'images et de vidéos sont très souvent codés avec des fréquences, des codecs, des tailles qui diffèrent énormément selon l'application ou les utilisateurs.**

L'étude de ces flux et des codecs associés, et leur présentation lors d'un changement d'environnement sont donc souvent fastidieuses. Cela l'est d'autant plus lorsque cela doit être fait dans un cadre pédagogique, comme par exemple au sein d'un DUT ou d'une LP R&T.

GSTREAMER est un ensemble de bibliothèques dédié au codage de données multimédia. C'est un outil pratique, qui montre bien la notion de *flux* (*stream* en anglais). GSTREAMER représente donc une solution pour l'apprentissage du principe de codage/compression/transmission de données multimédia via les réseaux actuels, et ce pour différentes applications.

Dans cet article nous montrons comment fonctionne GSTREAMER, puis nous décrivons plusieurs mises en oeuvre possibles pour des séances de travaux pratiques dans le cadre d'un DUT ou d'une LP R&T.

I. INTRODUCTION

A l'heure actuelle, l'ensemble des médias utilisés au quotidien sont numériques. Malgré la forte augmentation des bandes passantes des réseaux filaires ou hertziens, on se heurte toujours à un besoin de qualité toujours plus grande qui induit des quantités de données à transmettre toujours plus volumineuses.

La nécessité pour l'audio et la vidéo de trouver des techniques de compression et de transmission toujours plus efficaces conduit à une compétition très vive au niveau des codecs, des conteneurs multimédia, voire des protocoles. Les organismes de normalisation, les éditeurs de logiciels, les éditeurs de contenus, les opérateurs réseaux et télécoms, le monde des logiciels libres, tous proposent leurs solutions dans l'espoir de remporter une part de cet énorme marché. Les grands opérateurs multi-cartes de l'Internet, comme Netflix, Apple, Google, ou encore Microsoft ont un environnement de production/compression/distribution relativement fermé et cohérent. Là où cela devient plus compliqué, c'est lorsque l'on veut sortir de leur *Business world* pour mélanger les sources, comparer les méthodes, bref ce que l'on veut faire d'un point de vue pédagogique.

GSTREAMER est un ensemble de bibliothèques logicielles qui permet d'accéder facilement et de manière unifiée aux traitements de flux multimédia, en gérant la synchronisation, le mélange, les codages et décodages, les espaces de couleurs, les échantillonnages, les traitements simultanés, les entrées/sorties multiples par fichiers ou réseaux, etc. Et tout cela sans avoir à programmer la moindre ligne de code, ni à créer de fichiers intermédiaires. Il est plus léger que `Matlab`, `Scilab`, plus facile et plus général que `avconvert`, `ffmpeg`, `vlc`, l'ancien `khoros` et de plus est gratuit, ce qui ne gâche rien [1]. C'est pour toutes ces raisons que nous utilisons depuis quelques années GSTREAMER dans le cadre de Travaux Pratiques au sein de notre LP R&T.

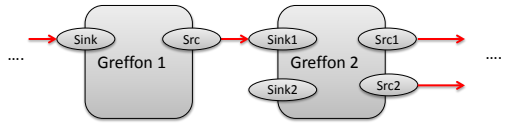

FIGURE 1. Description d'une mise en connexion de 2 greffons (*plugins*) à l'aide des *pads* d'entrée (*sinks*) et de sortie (*src*). Les flèches rouges représentent les flux (*streams*) qui transitent entre les greffons tout au long du *pipeline*.

L'objectif de cet article est de présenter GSTREAMER à la communauté R&T, et de montrer à travers plusieurs exemples son fonctionnement. Le reste de l'article est organisé de la manière suivante. Les sections II et III présentent le logiciel GSTREAMER, son fonctionnement et ses avantages. Ensuite, les sections IV, V, VI, VII et VIII donnent des exemples d'applications que l'on utilise pour inculquer certaines notions liés au codage auprès des étudiants de LP. Enfin nous concluons dans la section IX.

II. DESCRIPTION DE GSTREAMER

A. Les éléments de base

GSTREAMER manipule ce que l'on appelle des *streams*, c'est-à-dire des flux de données. Ces flux seront organisés dans des *pipelines*, qui peuvent être indépendants, synchronisés, convergents ou divergents. Dans chaque *pipeline* s'agencent des *greffons* (*plugins* en anglais) qui auront des fonctions bien spécifiques. Ces greffons disposent d'entrées/sorties appelées usuellement *pads*.

Ce sont ces *pads* que nous connectons les uns aux autres pour construire un *pipeline* complet : voir Figure 1. Les *pads* de sorties se nomment des sources (*src*) et les *pads* d'entrées se nomment des *sinks*. *Sinks* signifie lavabos en anglais. Ce terme illustre bien l'idée que le flux va se déverser dans le greffon qui le videra au fur et à mesure de son traitement. Certains greffons peuvent avoir plusieurs *sinks* et/ou *src*, tandis que d'autres ne pourront avoir que des *sinks* (un greffon de visualisation par exemple), ou que des *src* (la source d'un flux, comme une url, par exemple).

B. La synthèse

La concaténation des greffons et donc l'exécution d'un *pipeline* est mise en œuvre par l'application `gst-launch` que l'on tapera dans un terminal selon la logique suivante :

```
gst-launch-x.x greffon1 ! greffon2 ! ...
```

les caractères `x.x` correspondent à la version de GSTREAMER. Dans tous les exemples de cet article, nous utilisons la 1.0 de GSTREAMER. Pour la plupart des exemples de cet article, nous utilisons xubuntu 14.04.1. L'avantage de cette distribution est que GSTREAMER ainsi que de nombreuses librairies associées sont installées par défaut. La prise en main peut donc se faire très rapidement.
Les greffons possèdent évidemment de nombreux paramètres (*element properties*). On les positionne à la suite du nom du greffon de la manière suivante :

```
gst-launch-x.x greffon1 \
    elemProp1=param1 \
    elemProp2=param2 ... ! ...
```

Pour connaître les paramètres de chaque greffon, il suffit d'utiliser la commande

```
gst-inspect-x.x nomGreffon
```

qui donne une description complexe de tous les greffons de GSTREAMER. GSTREAMER peut aussi fonctionner d'une autre manière : au lieu d'utiliser `gst-launch` pour synthétiser la fonction désirée, il est possible de faire les appels de fonction au niveau d'un code C et donc d'utiliser une véritable application multimédia. Cette option n'est pas utilisée actuellement lors des TP.

C. Un exemple simple

Une caméra IP Axis délivre une séquence d'images jpeg que l'on souhaite visualiser à distance. Pour cela, il nous faut capturer le flux réseau (http) en temps réel, le décomprimer, ajuster l'espace de couleur et l'afficher sur notre écran. Le caractère `'!'` fait la concaténation entre les greffons. Cela s'écrit donc de la manière suivante :

```
gst-launch-1.0 souphttpsrc \
   location=http://...131/video.mjpg \
   ! jpegdec ! videoconvert ! osxvideosink
```

où
- `souphttpsrc` est un greffon permettant de capturer des données multimédia à partir d'une URL ;
- `location` indique l'URL concernée ;
- `jpegdec` est un greffon permettant de décoder des images au format jpeg ;
- `videoconvert` est un greffon permettant de convertir une vidéo d'un espace de couleur à un autre ;
- `osxvideosink` est un greffon permettant d'afficher des séquences d'images.

On peut voir le *pipeline* ainsi obtenu représenté en diagramme bloc sur la figure 2.

D. Les capacités

Il y a toujours une négociation entre un *pad* d'entrée et un *pad* de sortie car ils doivent obtenir réciproquement des informations les uns des autres. Ces informations que l'on nomme *caps* (pour *capabilities* en anglais), sont transmises sous forme de type *mime* étendus. On observe par exemple sur la figure 2 que l'on a en entrée de `jpegdec` un type mime simple, assez logiquement *image/jpeg*. Par contre, à sa sortie, on a une image brute de format YUV/I420 (voir figure 3), qui est ensuite transmise au greffon `videoconvert`. Enfin, le greffon `osxvideosink` (disponible uniquement sur MacOSX) ne pouvant prendre en entrée qu'un format UYVY (voir figure 4), demande au greffon `videoconvert` de lui envoyer les images sous ce format-là. Ce dernier connaît donc l'espace de couleur d'entrée, mais aussi celui de sortie qu'il doit générer. Il est ainsi possible de lever des ambiguïtés aussi bien par une extrémité du *pipeline* que par l'autre, et les greffons finalement s'adaptent.

```
Pad Templates:
  SRC template: 'src'
    Availability: Always
    Capabilities:
      video/x-raw
                 format: { I420, RGB, BGR, RGBx, xRGB, BGRx, xBGR, GRAY8 }
                  width: [ 1, 2147483647 ]
                 height: [ 1, 2147483647 ]
              framerate: [ 0/1, 2147483647/1 ]
```

FIGURE 3. Capacités (*caps*) du greffon `jpegdec` obtenues via la fonction `gst-inspect-x.x jpegdec`.

```
Pad Templates:
  SINK template: 'sink'
    Availability: Always
    Capabilities:
      video/x-raw
              framerate: [ 0/1, 2147483647/1 ]
                  width: [ 1, 2147483647 ]
                 height: [ 1, 2147483647 ]
                 format: UYVY
```

FIGURE 4. Capacités (*caps*) du greffon `osxvideosink` obtenues via la fonction `gst-inspect-x.x osxvideosink`.

Quand la chaîne a déterminé les bonnes caractéristiques d'entrée et de sortie de tous les greffons, le *pipeline* peut entrer en action, et dans cet exemple, la vidéo s'affiche sur notre ordinateur en temps réel.

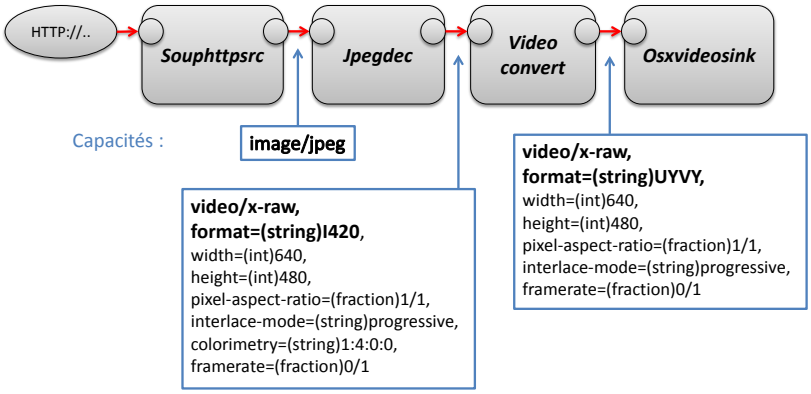

FIGURE 2. Lecture et affichage d'une séquence d'images Jpeg.

III. PLUSIEURS PIPELINES, MAIS PAS UNE USINE À GAZ

De nombreux décodeurs ou encodeurs sont fournis, mais retrouver l'encodeur qui convient pour une séquence particulière est souvent complexe. Une solution intéressante - et pratique pour certains de nos TP - que fournit GSTREAMER est l'automatisation de la reconnaissance des entrées et sorties au moyen des *caps*, pour ensuite mettre en place automatiquement le bon *pipeline*.

A. En cas de sorties multiples

Chaque greffon dans le *pipeline* et chaque *pad* ont un nom. Il est donc possible de relier finement les entrées aux sorties, mais ce n'est souvent qu'une source d'erreurs car la négociation entre les greffons au moyen des pads est souvent beaucoup plus juste : prenons par exemple le greffon `matroskademux`, un démultiplexeur vidéo, audio et sous-titres. Si on lui connecte un greffon qui ne lit que de la vidéo (comme `videoconvert`), `matroskademux` ne créera que le *pad* vidéo nécessaire.

B. Decodebin

Certains greffons comme `decodebin` permettent de simplifier énormément la taille apparente du *pipeline*. Ils ont la possibilité d'analyser le type de flux, de reconnaître au besoin le type de conteneurs, de générer tous les *pads* intermédiaires et ensuite d'aller chercher les *parseurs*, puis les décodeurs, pour chacun des signaux et de sortir les flux voulus à la condition qu'un *pad* de sortie adapté soit fourni. En TP cela permet de masquer temporairement certains problèmes complexes en faisant que cela "tombe en marche" sans avoir à tout expliquer aux étudiants dès la première utilisation.

C. Rtspsrc

Un autre exemple de greffon très intéressant : le greffon `rtspsrc`. Il permet de mettre en place une session `rtsp` pour faire du streaming qui gère correctement les flux `rtp` et les messages `rtcp`. On peut tout laisser automatique mais on peut aussi éliminer `rtcp` et évaluer l'amélioration due au contrôle.

Il n'est pas évident d'utiliser directement l'accès `udp`, mais les essais faits dans les différentes conditions convainquent facilement de l'intérêt d'utiliser les protocoles temps-réels complets.

IV. GESTION D'UN FLUX AUDIO

Nous allons maintenant montrer comment mettre en place des chaînes de traitement de données audio à l'aide de GSTREAMER.

A. Manipulation d'un fichier mp3

On souhaite lire en local un fichier audio compressé au format mp3. Avant de l'écouter, il faut donc le décompresser. Si on ne connait pas encore le greffon permettant de décompresser un mp3, il suffit de taper `gst-inspect-x.x | grep mp3` dans le terminal : il apparaît alors la liste de tous les greffons disponibles liés au mot-clef *mp3* : voir figure 5.

```
fred@fred-VirtualBox:~$ gst-inspect-1.0 | grep mp3
flump3dec:  flump3dec: Fluendo MP3 Decoder (liboil build)
typefindfunctions: application/x-id3v2: mp3, mp2, mp1, mpga, ogg, flac, tta
typefindfunctions: application/x-id3v1: mp3, mp2, mp1, mpga, ogg, flac, tta
typefindfunctions: application/x-apetag: mp3, ape, mpc, wv
typefindfunctions: audio/mpeg: mp3, mp2, mp1, mpga
mpg123:  mpg123audiodec: mpg123 mp3 decoder
mad:  mad: mad mp3 decoder
libav:  avdec_mp3: libav MP3 (MPEG audio layer 3) decoder
libav:  avdec_mp3float: libav MP3 (MPEG audio layer 3) decoder
libav:  avdec_mp3adu: libav ADU (Application Data Unit) MP3 (MPEG audio layer 3) decoder
libav:  avdec_mp3adufloat: libav ADU (Application Data Unit) MP3 (MPEG audio layer 3) decoder
libav:  avdec_mp3on4: libav MP3onMP4 decoder
libav:  avdec_mp3on4float: libav MP3onMP4 decoder
libav:  avmux_mp3: libav MP3 (MPEG audio layer 3) formatter (not recommended, use id3v2mux instead)
lame:  lamemp3enc: L.A.M.E. mp3 encoder
```

FIGURE 5. La commande `gst-inspect-x.x | grep mp3` permet de connaître tous les greffons dédiés au format mp3.

Après avoir inspecté la fonction `mad`, on voit que celle-ci fait office de décompresseur. La commande pour lire notre fichier mp3 va donc être

```
gst-launch-1.0 -v \
    filesrc location=sample.mp3 \
    ! mad ! autoaudiosink
```

avec

- `-v` une option permettant de voir en détails ce qu'il se passe durant l'exécution ;
- `filesrc` le greffon permettant de capturer un fichier local, dont la position est indiquée grâce au paramètre `location` ;

– `mad` le greffon qui va décompresser le fichier mp3 (mime audio/mpeg) pour en faire un fichier brut (mime audio/raw) ;
– `autoaudiosink` un greffon qui va détecter automatiquement un lecteur audio adapté aux cartes son présentes sur le système utilisé.

Dans le cadre d'un TP sur la compression audio, on peut vouloir montrer l'impact d'une compression forte sur la qualité sonore du fichier. Pour cela, on peut utiliser le greffon `audioresample`, qui permet de convertir/filtrer un fichier audio. En inspectant ce greffon, on voit qu'il existe un paramètre `quality` qui permet de régler la qualité désirée en sortie du greffon :

```
gst-launch-1.0 -v \
    filesrc location=sample.mp3 \
    ! mad ! audioresample quality=0 \
    ! pulsesink
```

La valeur 0 indique que l'on veut la qualité minimale. A l'écoute, on se rend compte que l'on ne perçoit pas la différence (il est toujours plus difficile de mettre en évidence la dégradation sonore d'un fichier audio. Ceci est plus facile avec une image car nous sommes plus sensibles à une dégradation visuelle). Par conséquent, toujours dans ce même TP, nous souhaitons mettre en évidence l'impact sonore d'une réduction du débit. On peut vouloir par exemple montrer la différence entre le débit d'un téléphone GSM (8kHz), par rapport à un son Hi-Fi (44,1kHz). Pour cela, on va utiliser les capacités du greffon `audioresample`, voir figure 6 :

```
Pad Templates:
  SRC template: 'src'
    Availability: Always
    Capabilities:
      audio/x-raw
                 format: { F32LE, F64LE, S32LE, S24LE, S16LE, S8 }
                   rate: [ 1, 2147483647 ]
               channels: [ 1, 2147483647 ]
                 layout: { interleaved, non-interleaved }
```

FIGURE 6. La commande `gst-inspect-x.x audioresample` permet de connaître les capacités (`caps`) du greffon `audioresample`.

On voit que la capacité `rate` existe pour le pad `src`. On va donc modifier le débit initialement à 44kHz en sortie du greffon grace à la commande :

```
gst-launch-1.0 -v \
    filesrc location=sample.mp3 \
    ! mad ! audioresample \
    ! "audio/x-raw,rate=8000" ! pulsesink
```

On observe que l'utilisation d'une capacité nécessite une syntaxe bien particulière, différente des paramètres des greffons, puisque l'on indique la conversion en sortie du greffon `audioresample`, comme si cette capacité était un greffon distinct. D'où l'utilisation du " !" après `audioresample`.

B. Gestion de données audio oggvorbis

Dans cette section, nous montrons comment lire un fichier audio `oggvorbis`. Cette manipulation est intéressante pour montrer aux étudiants la différence entre un conteneur (format ogg) et un codec (format vorbis), deux notions qui sont souvent confondues par les étudiants. Pour mémoire, `vorbis` est un codec audio - moins populaire mais plus performant que le mp3 - qui doit être absolument encapsulé pour être transmis. La ligne de commandes qu'il faut utiliser est donc la suivante :

```
gst-launch-1.0 -v souphttpsrc
    location="http://.../Exemple.ogg" \
    ! oggdemux ! vorbisdec ! pulsesink
```

Cette commande illustre bien le besoin d'utiliser une étape de désencapsulation avant décodage, et donc par conséquent la notion de conteneur.

V. GESTION D'UN FLUX VIDÉO AVEC SON

Jusqu'à présent, tous nos exemples ne traitent qu'un seul flux, nécessitant la mise en place d'un seul *pipeline*. Prenons maintenant l'exemple d'une lecture suivie de l'affichage d'une vidéo+son sous format mp4. Avec ce format, il faut démultiplexer la source (avec le greffon `qtdemux`) afin de distinguer le flux audio et le flux vidéo. Ensuite, il faut créer deux *pipelines* pour traiter séparément ces deux flux, comme l'illustre la figure 7. Pour cela, il est nécessaire d'utiliser le paramètre `name` du greffon `qtdemux` qui permet de renommer ce dernier. Ainsi nous pouvons utiliser les `src` de ce greffon à différents endroits dans notre ligne de commandes, comme on peut le voir ci-dessous :

```
gst-launch-1.0 -v souphttpsrc \
    location=https://.../video.mp4 \
    ! qtdemux name=flux \
flux.video_0 ! queue ! decodebin \
    ! videoconvert ! ximagesink \
flux.audio_0 ! queue ! decodebin \
    ! audioconvert ! autoaudiosink
```

On peut y observer que le greffon `qtdemux` a été nommé `flux` : cela nous permet par la suite d'utiliser dans deux *pipelines* différents les `src video_0` et `audio_0` qui fournissent, respectivement, les flux vidéo et audio du fichier initial. On remarquera qu'il n'y a pas de " !" après l'utilisation du greffon `qtdemux` puisque le reste de la commande concerne un autre *pipeline*. Idem après l'utilisation du greffon `ximagesink` qui affiche la vidéo.

Cet exemple permet aussi de présenter un greffon très important, `queue`, que l'on utilise à deux reprises ici. Sans ce greffon, l'affichage de la vidéo+son ne se ferait pas correctement, car les deux *pipelines* se bloqueraient l'un l'autre. Ce greffon permet de créer un *thread* (tâche en français) par *pipeline* et donc de les exécuter en parallèle. Chacun ayant sa propre horloge, synchronisée par le *pipeline* global, ils s'exécutent de manière synchrone.

VI. CRÉATION D'UN MUR DE VIDÉOSURVEILLANCE

La vidéosurveillance est l'un des mots clefs de certaines LP R&T. GSTREAMER peut être utilisé pour mettre en place un "mur de surveillance". Là encore l'utilisation de plusieurs *pipelines* facilite la création de ce mur : voir figure 8. Il suffit de récupérer le flux vidéo de plusieurs caméras via plusieurs *pipelines* en parallèle, redimensionner ces vidéos grâce au greffon `videoscale` - pour faciliter leur traitement et leur affichage en simultané - et enfin faire une mosaïque de ces vidéos dans une seule et même fenêtre graphique grâce au

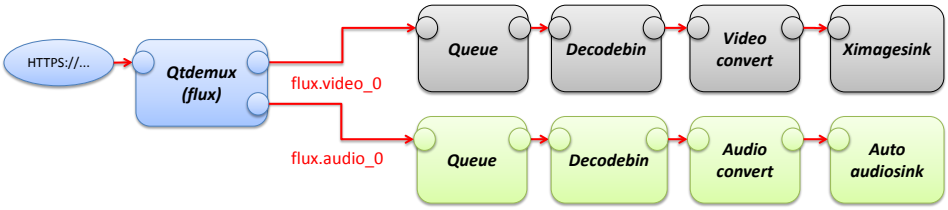

FIGURE 7. Lecture d'une vidéo avec son et affichage, grâce à la création de plusieurs *pipelines* et à l'utilisation du greffon `queue` qui sépare et parallélise les traitements.

greffon *videomixer*. On remarque au passage que GSTREAMER propose des greffons permettant de faire du traitement vidéo (redimensionnement), mais aussi du traitement audio, image, etc. La ligne de commandes sera la suivante :

```
gst-launch-1.0 -v videomixer name=mix \
    sink_1::ypos=240 ! videoconvert \
    ! ximagesink \
rtspsrc \
    location=rtsp://...131:554/media.amp \
    ! rtpmp4vdepay ! avdec_mpeg4 \
    ! videoscale \
    ! "video/x-raw,width=320,height=240" \
    ! mix. \
rtspsrc \
    location=rtsp://...133:554/media.amp \
    ! rtpmp4vdepay ! avdec_mpeg4 \
    ! videoscale \
    ! "video/x-raw,width=320,height=240" \
    ! mix.
```

VII. TRAITEMENT EN PARALLÈLE D'UN MÊME FLUX

Nous allons maintenant montrer comment on peut observer, en même temps, la représentation temporelle et le spectre d'un fichier mp3, et cela tout en l'écoutant. Ceci est possible puisque nous savons maintenant manipuler plusieurs *pipelines*. Une fois le fichier `sample.mp3` décodé, il suffit de l'envoyer dans trois *pipelines*, un premier pour afficher la représentation temporelle grâce au greffon `wavescope`, un deuxième pour afficher le spectre (grâce au greffon `spectrascope`) et enfin un troisième pour écouter le signal avec `pulsesink`. En nous inspirant de ce que l'on fait section V pour traiter en parallèle l'image et le son d'une vidéo, nous testons la commande suivante :

```
gst-launch-1.0 -v filesrc \
    location="sample.mp3" ! mad name=q \
q. ! queue ! audioconvert \
    ! wavescope ! ximagesink \
q. ! queue ! audioconvert \
    ! spectrascope ! ximagesink \
q. ! queue ! pulsesink
```

Malheureusement cela ne marche pas ! La raison est simple : GSTREAMER ne peut pas connecter un pad `src` avec plusieurs `sinks`. Pour permettre cela, on va utiliser un greffon nommé `tee`, qui permet de dériver le flux d'un pad `src` dans plusieurs `sinks` : voir figure 9.

La commande qui fonctionne est la suivante :

```
gst-launch-1.0 -v \
    filesrc location="sample.mp3" \
    ! mad ! tee name=q \
q. ! queue ! audioconvert \
    ! wavescope ! ximagesink \
q. ! queue ! audioconvert \
    ! spectrascope ! ximagesink \
q. ! queue ! pulsesink
```

VIII. LE RÉSEAU SUR ÉCOUTES

Il peut être amusant d'enregistrer le flux d'une conversation ou d'une visioconférence avec un utilitaire tel que `tcpdump`. Il suffit, sur un switch, de dupliquer le flux vers un autre port et de l'envoyer au moyen d'une machine espion dans un fichier `.pcap`, compatible avec `wireshark`. Nous conseillons fortement l'utilisation de `tcpdump` car le format de fichier obtenu avec `tshark` ou `wireshark` pose des problèmes au greffon `pcapparse`. Considérons que la machine espionnée ait pour adresse 10.4.110.132, que l'audio soit en G722 et la vidéo en H264.

Commençons par récupérer le flux dans un fichier :

```
tcpdump -i eth0  -w /tmp/unFluxComplet.pcap \
   host 10.4.110.132
```

Un examen avec `wireshark` montre que ce fichier contient quatre flux différents : deux vidéos et deux audios dans les deux sens. Pour distinguer ces flux, il va falloir identifier les ports sources par exemple, et donner quelques informations supplémentaires au greffon. En effet, si la capture est lancée avant le début des échanges SIP, `wireshark` utilise les informations de SIP pour identifier les paramètres dynamiques qui sont fournis en RTP et les afficher de manière explicite. Le greffon n'a pas ce niveau de développement, il faut donc le renseigner. Examinons la commande qui extrait le son du fichier de capture et le sauvegarde au format `opus` dans un conteneur `matroska` :

```
gst-launch-1.0 -v filesrc \
    location= /tmp/unFluxComplet.pcap \
  ! pcapparse src-ip=10.4.110.132 \
    src-port=2326 \
  ! "application/x-rtp,payload=9,
      clock-rate=8000" \
  ! rtpg722depay ! avdec_g722 ! opusenc \
  ! matroskamux ! filesink \
    location=/tmp/son.mkv
```

Le greffon `pcapparse` utilise ici `src-ip` pour sélectionner la machine source et `src-port` pour sélectionner le flux audio. Ce paramètre ne peut qu'être lu dans le fichier `unFluxComplet.pcap` avec `wireshark`. Maintenant que le flux est identifié de manière unique, il faut indiquer le type de charge utile (*payload* en anglais) et l'horloge. Ici on utilise les capacités pour que le greffon `rtpg722depay` interprète correctement son flux d'entrée : c'est du `rtp` avec une charge

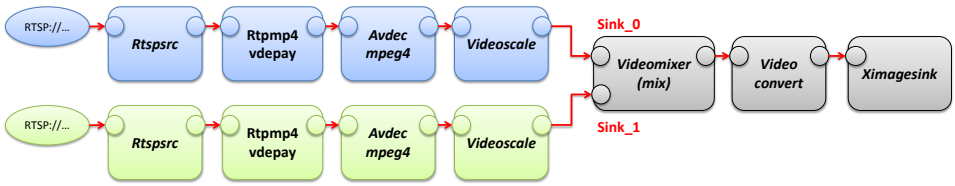

FIGURE 8. Mise en place d'un mur de vidéosurveillance : lecture de plusieurs flux vidéos et mixage dans une seule image (avec redimensionnement des vidéos).

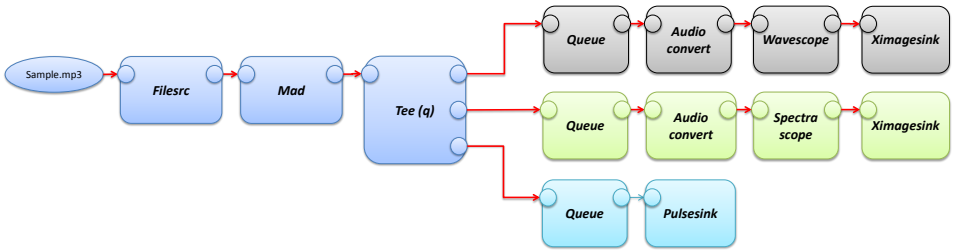

FIGURE 9. GSTREAMER ne peut pas connecter un pad `src` avec plusieurs `sinks`. Pour permettre cela, il faut dériver le flux à l'aide du greffon nommé `tee`.

G722 (le numéro 9 a cette signification), à 8000 Hz. Ces valeurs sont présentes dans la partie SDP (*Session Description Protocol*) de l'invite SIP ou mieux, dans celle de la réponse à l'invite.

Ensuite, on enlève l'encapsulation RTP tout en interprétant les paramètres de l'entête RTP, puis il est décodé en format brut. Pour pouvoir le sauvegarder dans un conteneur `matroska`, il faut le ré-encoder. Ici, c'est l'encodeur `opus` qui a été choisi. On aurait pu écouter directement l'audio sans le sauvegarder dans un fichier.

Il est tout à fait possible de faire un pipeline semblable pour la vidéo. La seule différence est que le numéro de charge utile est dynamique et doit donc obligatoirement être recherché dans les échanges SIP. Par contre, vu le volume de données à manipuler, il n'est pas possible de visualiser la vidéo en temps réel.

IX. CONCLUSION

Dans cet article, nous avons présenté GSTREAMER, un ensemble de librairies gratuites permettant de manipuler des flux de données multimédia. Précisément, nous avons donné des exemples pour manipuler du son, des séquences d'images, des vidéos (nécessitant le traitement simultané du son et des images), incruster dans une même fenêtre plusieurs vidéos, etc. Nous aurions pu aussi vous montrer comment se brancher sur sa box pour regarder la télévision, décoder et ré-encoder un flux, changer le format d'un conteneur, ajouter des sous-titres à une vidéo, faire du traitement de signal (filtrage, etc.)... GSTREAMER est un outil extrêmement puissant qui n'a comme limite que votre imagination. Pour vous en persuader, de nombreux exemples sont aussi disponibles sur le site de Nicolas Hennion [2], ou encore de Patrice Ferlet [3] dont nous nous sommes inspirés pour faire certaines de nos manipulations.

Actuellement, ces manipulations sont effectuées lors de TP de Licence Professionnelle dans un module intitulé *Codage et compression de données multimédia*. A l'avenir, nous envisageons d'utiliser aussi GSTREAMER dans des modules de DUT tels que *M1107 - Initiation à la mesure du signal*, *M1108 - Acquisition et codage de l'information* ou encore *M2108 - Chaîne de transmission numérique*. GSTREAMER nous semble être une bonne alternative à des logiciels de traitement de signal - payants ! - tels que *Labview* ou encore *Matlab*.

RÉFÉRENCES

[1] Gstreamer team. (2014) Gstreamer features. [Online]. Available : http ://gstreamer.freedesktop.org/features/

[2] N. Hennion. (2014) Le blog de nicolargo. [Online]. Available : http ://blog.nicolargo.com/ ?s=gstreamer

[3] P. Ferlet. (2014) Metal3d.org. [Online]. Available : http ://www.metal3d.org/ticket/2012/08/13/didacticiel-gstreamer