Gstreamer programming

Rajeev Tiwari
Sr Principal Architect

What's Gstreamer

- GStreamer is, in short, an engine used by other applications to play media files
- In more detail it is a framework based on graphs of filters which operate on media data. Applications using this library can do anything from the sound processing to real-time video playback, and handle almost everything that is related to multimedia. Its architecture based on plugins, you can easily add new data types or new treatment options.

License – LGPL

- GStreamer is released under LGPL (GNU Lesser General Public License).
- LGPL means that the GStremer software itself has copyleft rules which means that you have to keep the software free.
- But you are allowed to create your own software with your own copyright rules and link it with GStremer.
- For companies like **Nokia** this license gives a profitable basis for their own software. They can use GStremer as a multimedia framework, but own the copyright to their own software, because it is merely linked to GStreamer.

Applications

- GStreamer's most obvious use is creating a media player on top of it, because it supports many needed formats for this.
- GStreamer can also be used to create more complex programs like video or audio editing applications.
- We intend to use it to create an Internet radio.

Usage

- GStreamer is so called "pipelined" multimedia framework
  - Pipe-line consists of sequence of operations
- The basic building block of a pipeline is called an element
  - Pipe-line consists of chain of these elements and data flows through it.
- For example, a basic mp3 audio decoder
  - Element 1
    - Reads mp3 file from hard disk.
  - Element 2
    - Decodes the mp3 data to raw audio
  - Element 3
    - Sends the raw audio to soundcard

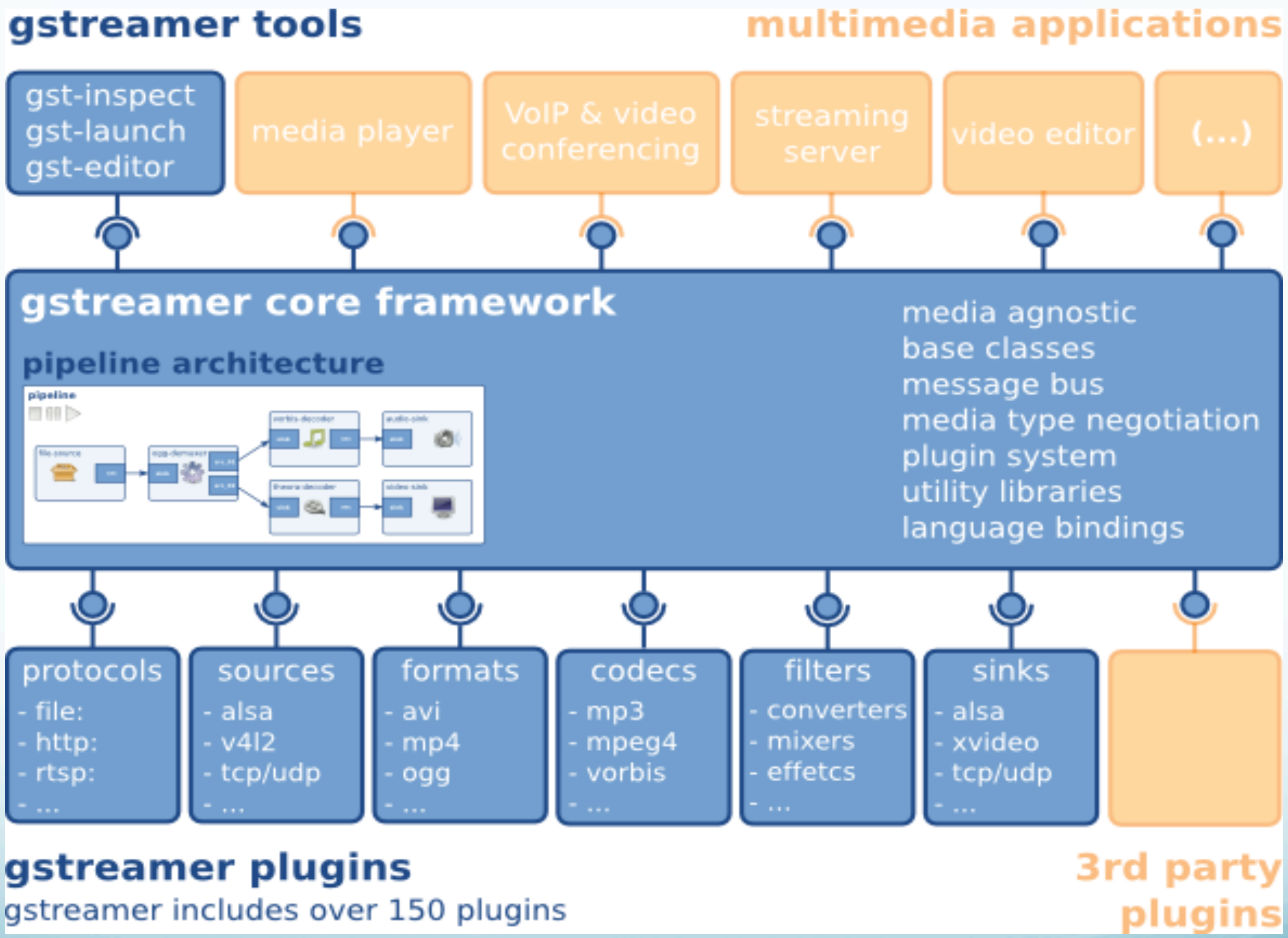
Usage

- Every element may also have a different number of pads
  - Sink (input) pad and source (output) pad.
  - Pad is an element’s link to outside world
  - Pads can also be dynamic so that a pad can be randomly created and/or destroyed.
- Interconnected elements form a bin
  - Is also element
  - Same operations that can be done to an element can be done to a bin also
  - Pipeline is a specialized type of bin so that the top level bin has to be pipeline

Usage

- A bus takes care of internal messaging in GStreamer.
- A callback function can be defined that takes care of EOS or other errors messages and acts upon them.
- Every pipe-line has a bus by default
  - The developer should create a callback function for the bus.
  - When the main loop is running the bus is checked for messages and when a new message is noticed the callback function will be called

Gstreamer Global Architecture



What GStreamer provides ?

- An API for multimedia applications
- A plugin architecture
- A pipeline architecture
- A mechanism for media type handling/negotiation
- Over 200+ plug-ins

GStreamer plug-ins Classification

- **Protocols handling**
- **Sources:** data input (involves protocol plugins)
- **Formats:** parsers, formaters, muxers, demuxers, metadata, subtitles
- **Codecs:** coders and decoders
- **Filters:** converters, mixers, effects, ...
- **Sinks:** data output (involves protocol plugins)

Gstreamer Packaging

GStreamer is packaged into :

- **gstreamer:** the core package
- **gst-plugins-base:** an essential exemplary set of elements
- **gst-plugins-good:** a set of good-quality plug-ins under LGPL
- **gst-plugins-ugly:** a set of good-quality plug-ins that might pose distribution problems
- **gst-plugins-bad:** a set of plug-ins that need more quality
- **gst-python:** the python bindings
- a few others packages

Gstreamer Tools
gst-inspect

- Liste all gstreamer elements installed on the platforme :

```
# gst-inspect
          flumpegdemux:  flutsdemux: MPEG Transport stream demuxer
          flumpegdemux:  flupsdemux: The Fluendo MPEG Program Stream Demuxer
          ........
          ........
          ........

          ........
```

- Get information about an element :

```
# gst-inspect  <element name>
```

Example

```
#include <gst/gst.h>



int

main (int   argc,

      char *argv[])

{

pipeline = gst_pipeline_new ("audio-player");

  source = gst_element_factory_make ("filesrc", "file-source");

  parser = gst_element_factory_make ("oggdemux", "ogg-parser");

  decoder = gst_element_factory_make ("vorbisdec", "vorbis-decoder");

  conv = gst_element_factory_make ("audioconvert", "converter");

  sink = gst_element_factory_make ("alsasink", "alsa-output");

  g_object_set (G_OBJECT (source), "location", argv[1], NULL);



bus = gst_pipeline_get_bus (GST_PIPELINE (pipeline));

gst_bus_add_watch (bus, bus_call, loop);

gst_object_unref  (bus);



 gst_bin_add_many (GST_BIN (pipeline),

                                                    source, parser, decoder, conv, sink, NULL);



  gst_element_link (source, parser);

  gst_element_link_many (decoder, conv, sink, NULL);

  g_signal_connect (parser, "pad-added", G_CALLBACK (new_pad), NULL);

}
```

Example

Creating elements

```
#include <gst/gst.h>


int

main (int   argc,

      char *argv[])

{

pipeline = gst_pipeline_new ("audio-player");

  source = gst_element_factory_make ("filesrc", "file-source");

  parser = gst_element_factory_make ("oggdemux", "ogg-parser");

  decoder = gst_element_factory_make ("vorbisdec", "vorbis-decoder");

  conv = gst_element_factory_make ("audioconvert", "converter");

  sink = gst_element_factory_make ("alsasink", "alsa-output");

  g_object_set (G_OBJECT (source), "location", argv[1], NULL);


bus = gst_pipeline_get_bus (GST_PIPELINE (pipeline));

gst_bus_add_watch (bus, bus_call, loop);

gst_object_unref (bus);


 gst_bin_add_many (GST_BIN (pipeline),

                    source, parser, decoder, conv, sink, NULL);


  gst_element_link (source, parser);

  gst_element_link_many (decoder, conv, sink, NULL);

  g_signal_connect (parser, "pad-added", G_CALLBACK (new_pad), NULL);

}
```

Example

```
#include <gst/gst.h>


int

main (int   argc,

      char *argv[])

{

pipeline = gst_pipeline_new ("audio-player");

  source = gst_element_factory_make ("filesrc", "file-source");

  parser = gst_element_factory_make ("oggdemux", "ogg-parser");

  decoder = gst_element_factory_make ("vorbisdec", "vorbis-decoder");

  conv = gst_element_factory_make ("audioconvert", "converter");

  sink = gst_element_factory_make ("alsasink", "alsa-output");
```

Creating elements

```
  g_object_set (G_OBJECT (source), "location", argv[1], NULL);


bus = gst_pipeline_get_bus (GST_PIPELINE (pipeline));

gst_bus_add_watch (bus, bus_call, loop);

gst_object_unref (bus);


 gst_bin_add_many (GST_BIN (pipeline),

                                                source, parser, decoder, conv, sink, NULL);


  gst_element_link (source, parser);

  gst_element_link_many (decoder, conv, sink, NULL);

  g_sig                                    CK (new_pad), NULL);

}
```

Setting filesrc element properties

Example

Creating elements

```
#include <gst/gst.h>

int
main (int   argc,
      char *argv[])
{
pipeline = gst_pipeline_new ("audio-player");
  source = gst_element_factory_make ("filesrc", "file-source");
  parser = gst_element_factory_make ("oggdemux", "ogg-parser");
  decoder = gst_element_factory_make ("vorbisdec", "vorbis-decoder");
  conv = gst_element_factory_make ("audioconvert", "converter");
  sink = gst_element_factory_make ("alsasink", "alsa-output");
```

Adding the callback function for bus

Setting filesrc element properties

```
  g_object_set (G_OBJECT (source), "location", argv[1], NULL);

bus = gst_pipeline_get_bus (GST_PIPELINE (pipeline));
gst_bus_add_watch (bus, bus_call, loop);
gst_object_unref (bus);

 gst_bin_add_many (GST_BIN (pipeline),
                                                source, parser, decoder, conv, sink, NULL);

  gst_element_link (source, parser);
  gst_element_link_many (decoder, conv, sink, NULL);
  g_sig                              ACK (new_pad), NULL);
}
```

Example

Adding the callback function for bus

Putting elements to a bin

Creating elements

Setting filesrc element properties

```
#include <gst/gst.h>


int

main (int   argc,

      char *argv[])

{

pipeline = gst_pipeline_new ("audio-player");

  source = gst_element_factory_make ("filesrc", "file-source");

  parser = gst_element_factory_make ("oggdemux", "ogg-parser");

  decoder = gst_element_factory_make ("vorbisdec", "vorbis-decoder");

  conv = gst_element_factory_make ("audioconvert", "converter");

  sink = gst_element_factory_make ("alsasink", "alsa-output");
```

```
  g_object_set (G_OBJECT (source), "location", argv[1], NULL);


bus = gst_pipeline_get_bus (GST_PIPELINE

gst_bus_add_watch (bus, bus_call, loop);

gst_object_unref (bus);


 gst_bin_add_many (GST_BIN (pipeline),

                                     source, parser, decoder, conv, sink, NULL);


  gst_element_link (source, parser);

  gst_element_link_many (decoder, conv, sink, NULL);

  g_sig                                        ACK (new_pad), NULL);

}
```

Example

```
#include <gst/gst.h>


int

main (int   argc,

      char *argv[])

{

pipeline = gst_pipeline_new ("audio-player");

  source = gst_element_factory_make ("filesrc", "file-source");

  parser = gst_element_factory_make ("oggdemux", "ogg-parser");

  decoder = gst_element_factory_make ("vorbisdec", "vorbis-decoder");

  conv = gst_element_factory_make ("audioconvert", "converter");

  sink = gst_element_factory_make ("alsasink", "alsa-output");
```

```
  g_object_set (G_OBJECT (source), "location", argv[1], NULL);


 bus = gst_pipeline_get_bus (GST_PIPELINE

 gst_bus_add_watch (bus, bus_call, loop);

 gst_object_unref (bus);


 gst_bin_add_many (GST_BIN (pipeline),

                                  source, parser, decoder, conv, sink, NULL);


  gst_element_link (source, parser);

  gst_element_link_many (decoder, conv, sink, NULL);

  g_sig                                                ACK (new_pad), NULL);

 }
```

Adding the callback function for bus

Putting elements to a bin

Creating elements

Linking elements together

Setting filesrc element properties

Example

Callback funtion for error handling

```
static gboolean bus_call (GstBus     *bus,   GstMessage
      *msg,

        gpointer    data)

{
  GMainLoop *loop = (GMainLoop *) data;

  switch (GST_MESSAGE_TYPE (msg)) {
    case GST_MESSAGE_EOS:
      g_print ("End-of-stream\n");
      g_main_loop_quit (loop);
      break;
    case GST_MESSAGE_ERROR: {
      gchar *debug;
      GError *err;

      gst_message_parse_error (msg, &err, &debug);
      g_free (debug);
      g_print ("Error: %s\n", err->message);
      g_error_free (err);

      g_main_loop_quit (loop);
      break;
    }
    default:
      break;
  }
  return TRUE;
}
```

Function for dynamic pad creation

```
static void
new_pad (GstElement *element,
         GstPad     *pad,
         gpointer    data)
{
  GstPad *sinkpad;

  /* We can now link this pad with the audio decoder */
  g_print ("Dynamic pad created, linking parser/
      decoder\n");

  sinkpad = gst_element_get_pad (decoder, "sink");
  gst_pad_link (pad, sinkpad);

  gst_object_unref (sinkpad);
}
```

Resources

- ❑Gstreamer : http://www.gstreamer.org

- ❑Fluendo plugin : http://core.fluendo.com/gstreamer/src/gst-fluendo-ismd/

- ❑Gstreamer API : http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html/libgstreamer.html