Contribution ID: **63** Type: **not specified**

Streaming with Python, Twisted and GStreamer

Tuesday, 10 July 2007 14:00 (30 minutes)

Flumotion is a GPL streaming media server written in Python. It is distributed and component-based: every step in the streaming process (production, conversion, consumption) can be run inside a separate process on separate machines.

Flumotion uses Twisted and GStreamer. Twisted enables the high-level functionality, distributing components over the network. GStreamer, through the Python bindings, enables the high-speed low-level functionality: actual media processing.

Flumotion uses a central manager process to control the complete network; one or more worker processes distributed over machines to run actual streaming components; and one or more admin clients connecting to the manager to control it.

Flumotion is under very active development. In its latest stable release (0.4.2), it already supports the following features:

- various sources: webcams, soundcards, TV cards, Firewire cameras, looper
- various codecs: Vorbis, Theora, mulaw, JPEG, smoke
- various containers: Multipart, Ogg
- synchronized capturing across machines
- username/password authentication
- overlaying, colorbalance
- HTTP streaming, disk archiving
- administration GUI with a wizard for the most basic scenario
- ncurses-based administration
- code distribution from a central location
- local caching of the distributed code space
- strong focus on ease of use and usability
- improved support for network failure and reconnection
- multiplex all component feeds through only one port on the worker
- sharing HTTP port among streamers
- support for GStreamer 0.10

The current design allows for the following future features:

- any number of sources/containers/codecs/effects/protocols to be added
- completely centralized code upgrades
- complete code upgrades with minimal downtime

- any kind of authentication mechanism (key exchange, challenge/response, ...)
- any number of possible scenarios for actual content production and distribution
- manager failover and state replication
- loadbalancing streams over different servers or from different locations
- internal stresstests

Some of the features are only possible or easily implementable thanks to using a high-level language like Python:

- sending GUI code for the wizard and component administration from the manager to the admin client, making the admin GUI a lightweight shell
- sending component code to any of the workers over the wire at startup
- rebuilding modules and reloading code on the fly while running; allowing for a distributed code upgrade without losing clients
- rapid development of new components, allowing to catch up with and eventually to keep one step ahead of the competition
- easy networking code thanks to Twisted

In this project, we came up with solutions to specific problems presented to us that would be interesting to share with others.

- all code is stored centrally and partitioned into "bundles" which are cached by clients who need them. Versioning and dependencies are correctly handled, and to the code being run this is handled transparently. Code can still import as if it were one big file tree.
- the manager sends GUI code to an admin client and component code to a worker. The GUI code running on the admin machine then controls the behaviour of code running on the worker machine by going through the manager machine
- state of components is automatically replicated one level deep to the manager, and two levels deep to all connected admin clients
- authentication of any service in the network is handled by creating keycards which can be exchanged between all processes, implementing any type of challenge/response authentication as securely as possible.
- Twisted's Perspective Broker was extended to use this generic keycard concept instead of the current (limited) username/password credentials.
- the open-ended nature of Twisted and GStreamer is difficult to harness into a usable GUI. The wizard provides a good way of crystallizing all possibilities into a sensible task-based presentation. The design of the wizard also incorporates the flexibility of the network distribution and the dynamic code distribution by pulling in the necessary GUI code for the next step based on previous choices.
- moving to run-time checks of functionality as opposed to compile/configure-time. Since the server can be distributed over any number of machines, and actual components have different run-time needs, all checks for features (devices, required libraries, versions, permissions)

Python also presents some specific challenges when used in a large project as compared to more low-level languages. When handled right, these can actually be turned into project engineering advantages. Python's weak argument typing forces developers to document the API correctly. Python's dynamic nature (running code from received chunks, extensive subclassing) and Flumotion's design (componentized functionality, lots of small pieces of code sent back and forth) forces us to aggressively write unit testing for all functionality.

Primary author: Mr VANDER STICHELE, Thomas (Fluendo)

Presenter: Mr VANDER STICHELE, Thomas (Fluendo)

Session Classification: Python Language and Libraries