Ardour3 - Video Integration

Robin Gareus
gareus.org, linuxaudio.org, CiTu.fr
Paris, France
robin@gareus.org

Abstract

This article describes video-integration in the Ardour3 Digital Audio Workstation to facilitate sound-track creation and film post-production.

It aims to lay a foundation for users and developers, towards establishing a maintainable tool-set for using free-software in A/V soundtrack production.

To that end a client-server interface for communication between non-linear editing systems is specified. The paper describes the reference implementation and documents the current state of video integration into Ardour3 and future planned development. In the final sections a user-manual and setup/install information is presented.

Keywords

Ardour, Video, A/V, post-production, sound-track, film

1 Introduction

The idea of combining motion pictures with sound is nearly as old as the concept of cinema itself.

Ever since the invention of the moving picture, films have entailed sound and music. Be it mechanical music-boxes accompanying candle-light projections, the Kinetoscope in the late 19th century; live-music supporting silent-films in the early 20th century to completely digital special effects in the 2000s.

Charlie Chaplin composed his own music for City Lights (1931) and many of his later movies. He was not clear whose job was to score the sound-tracks [Scaruffi, 2007]. That was the exception, and few film-makers would imitate him. In the 1930s, after a few years of experimentation, scoring film soundtracks became an art in earnest. By the mid-1940s, cinema's composers had become a well-established category.

Apart from the film-score (music), a film's soundtrack usually includes dialogue and sound effects.

One goal is to synchronize dramatic events happening on screen with musical events in the score - but there are many different (artistic) methods for syncing music to picture. With only a few exceptions - namely song or dance scenes - music composition and sound-design usually takes place after recording and editing the video [Wikipedia, 2011].

Major problems persisted, leading to motion pictures and sound recording largely taking separate paths for a generation. The primary issue was - and is - synchronization: pictures and sound are recorded and played back by separate devices, which were difficult to start and maintain in tandem. This stigma still prevails for most professional audio/video recordings for both creative and technical: ie. there is too much gear on camera already, more (budgetary) choices are available, unions define different job functions. . . .

Post-production requires to synchronize the original audio recorded on the set (on-camera voice) with the edited video. In the *analog days* visual cues (slate) have been used towards that end. With the advent of digital technology time-code (most commonly SMPTE produced by the camera) is commonly recorded along with the sound to allow reconstructing the audio-tracks after the video has been cut.

As you can imagine, the technical skills and details involved can become quite complex and neither composers nor sound-designers do want to concern themselves with that task. Creative technicians have come up with various tools to aid the process.

These days many Digital Audio Workstations

provide features to import EDL (edit decision lists) or variants thereof (AAF: Advanced Authoring Format, MXF: Material eXchange Format, BWF: Broadcast Wave Format, OMF+OMFI: Open Media Framework Interchange,...) and display a film-clip in sync with the audio in order to facilitate soundtrack creation.

There are very few to none free-software solutions for this process. However, Ardour [Davis and others, 1999 2012] provides nearly all relevant features for composition, recording as well as mastering. The underlying JACK audio connection kit allows for inter-application synchronization [Davis and others, 2001 2012] and suggests itself to be used in the context of film post-production.

Figure 1: Ardour 3.0-beta1 with video-timeline patch and video-monitor

2 Design

Other FLOSS projects are tackling the process of video production (NLE: non-linear editing), most notably Blender [Foundation, 1999 2012], lives [Finch, 2004 2012] and cinelerra [Ltd, 2002 2011]; however these emphasize on the visuals and often significantly lack on the audio side.

Other free-software digital audio workstations (DAWs) capable of the job include musescore [Schweer and others, 1999 2012] and qtractor [Capela, 2005 2011] amongst others, however with MIDI support arriving in Ardour3, continued cross-platform support and available high-end professional derivatives (Mixbus, Harrison) it is currently the most suitable and promising DAW from a developer's perspective.

2.1 Design-goals and use-cases

The aim is to provide an easy-to-use, professional workflow for film-sound production using free-software. More specifically: Integration of video-elements into the Ardour Digital Audio Workstation.

The resulting interface must not be limited to the software at hand (Ardour, Xjadeo, icsd) but allow for further adaption or interoperability.

2.1.1 General

An important aspect of the envisaged project is modular design. This is a prerequisite to allow the project to be developed gradually and maintained long-term. With Interface definitions in place, the building blocks can be implemented individually. It is also crucial to cope with the current situation of video-codec licensing: Depending on the availability of licenses and user preferences, it should be possible to en/disable certain parts of the system without breaking overall functionality.

The utter minimum for both composing as well as producing sound-tracks is a video-player that synchronizes to an external time-source provided by the DAW.

To navigate around in larger projects a timeline becomes very valuable.

Session-management as well as archiving is important as soon as one works on more than one project. Project-revisions or snapshots come in handy.

Import, export and inter-operability with other software: Those are likely going to be the hardest part, yet also the most important.

Transcoding the video on import is necessary to provide smooth response times for forward/backward seeking. It may also be required for inter-operability (codecs). Demuxing is needed to import edited video-tracks along with the original set of audio tracks, aligned to timecode.

Exporting the soundtrack means aligning the materials and multiplexing the original video with the audio or to provide exact information on how to do that. Various formats - both proprietary and free - are in use which complicates the process.

Sometimes you (or the director) notices during sound-design, that some video edit decisions were not optimal; or that there is just a video frame missing for proper alignment. With digital

technology incremental updates of the video are not only feasible but relatively simple. Empowering the audio-engineer to quickly do minor video editing is a very useful but also dangerous feature.

2.1.2 Short-Term

Increase the usability of existing tools with focus on soundtrack creation (video-timeline, video-monitor).

In particular the learnability and efficiency for the workflow should be streamlined: video-transcoding, video-monitor and A/V-session management functionality should be accessible from a single application. The complexity of the whole process should be abstracted and focus on the use-case at hand while not limiting the actual system.

Practically this is mapped to adding support into Ardour3 to communicate with and control the Xjadeo [Gareus and Garrido, 2006 2012b] video-monitor. Furthermore a video-timeline axis is aligned to the Ardour-canvas. Lastly, the possibility to invoke ffmpeg from within Ardour's GUI to import and export audio/video is implemented. The user-interaction is kept to a minimum: Import Video, Show/Hide Timeline, Show/Hide video-monitor.

2.1.3 Mid-Term

Improve system integration. Configuration presets for multi-host setups. Streamline video-export, MUX/codec/format presets.

Stabilize video-session and edit API. Add interoperability with external video-editors. Import various EDL dialects.

2.1.4 Long-Term

Top Notch client-server distributed A/V non-linear editor, using Ardour3 for the sound, dedicated GUI for the video. Seamless integration.

2.2 Architecture

Ardour itself is separated in two main parts: the back-end (audio-engine, libardour) which manages audio-tracks, audio-routes and session-state and the front-end (gtk2-ardour) which provides a stateless graphical user interface to the back-end.

The idea is to construct the system such than intrusion in Ardour itself is minimal: Only Ardour's front-end GUI should be *aware* of video-elements.

The video-decoding and video-session management is done in a separate application. A client-server model is used to partition tasks. Ardour as client does not share any of its resources, but requests video-content from the server. The overall outline is depicted in figure 2.

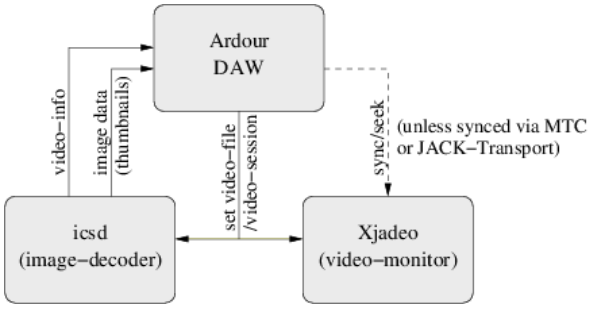

Figure 2: Overview of the client-server architecture

The video-server itself is a modular system. The essential and minimum system comprises a video-decoder and video-frame cache. Video-frames are referenced by frame-number. Time-code mapping is optional and can be done on the client and server-side (using server-side sessions). However, the server must provide information about the file's (or session's) frame-rate, aspect-ratio and start-time-offset. Furthermore, the server must be able to present server-side sessions as single file to the client.

Minimal configuration that needs to be shared by the server and client is the server's access URL and document-root.

Even though tight integration is planned, the prototype architecture leaves room for side-chains. In particular this concerns extraction of audio-tracks from video files as well as final multiplexing and mastering the final video. While interfaces are specified to handle these on the server-side, tight user-interface integration and rapid prototyping motivates a client-side implementation. This is accomplished by sharing the underlying file-system storage.

3 Interface

For read-only (no video-editing) access to video information, all communication between the client (here: Ardour) and the video-server is via HTTP.

This is motivated by:

- HTTP is a well supported protocol with existing infrastructure.
- with client-side caching: throughput is more important than low latency. HTTP overhead is reasonably small[1]
- web-interface possibility
- persistent HTTP connections are possible
- possibility to make use of established proxy and load-balancing systems

3.1 Protocol: Request Parameters

HTTP verbs can be taken into account for specific requests e.g. `HEAD`: query file-information, `DELETE`: remove assets, regions, files. The vast majority will bet `GET` and `POST` requests.

The server URL must conform to RFC3986 specifications. It must be possible to have the server in a sub-path on the server's root (e.g. http://example.org/my/server/). The URI must allow path and file-names to be added below that path (e.g. http://example.org/my/server/status.html). A default endpoint handler needs to catch access to the doc-root and provide service for the following requests URLs, which are described in further detail below:

/status print server-status (user-readable, version, etc) must return *200 OK* if the server is running.

/info return file or session info

/frame query image-frame data

/ generic handler to above according to request-parameters

/admin/flush_cache (optional, `POST`) reset cache

/admin/shutdown (optional, `POST`) request clean shutdown of server

/stream (optional) server-side rendering of video chunks - export and preview

/index/* (optional) file and directory index

/session/* (optional) server-side project and session management

The file extension defines the format of the returned data. For textual (non-binary) data, valid requests include `.html`, `.htm` , `.json`, `.xml`, `.raw`, `.edl`, `.null`. Valid image file extensions are `.jpg`, `.jpeg`, `.png`, `.ppm`, `.yuv`, `.rgb`, `.rgba`. For video streaming, the extension defines the container format e.g. `.avi`, `dv`, `.rm`, `.ogg`, `.ogv`, `.mpg`, `.mpeg`, `.flv`, `.mov`, `.mp4`, `.webm`, `.mkv`, `.vob`, ... The extension is case-insensitive.

Alternatively the format can be specified using the `format=`*`FMT`* query parameter which overrides the extension, if any.

3.1.1 File and Session information

The `/info` handler returns basic information about the file or session.

Request parameters:

file file-name relative to the server's document root.

Reply:

version reply-format version. Currently `1`, may change in the future

framerate a double floating point value of video frames-per-second

duration long integer count of video-frames

start-offset double precision - time of first video-frame; specified in (fractional) seconds

aspect-ratio floating point value of the width/height geometry ratio including pixel and display-aspect ratio multipliers

width (optional) integer video width in pixels

height (optional) integer video height in pixels

length (optional) video-duration in seconds

size (optional) video-size in bytes

The *raw* (un-formatted) reply concatenates the values above in order, separated by unix-newlines ('\n', ASCII 0x0a). Optional values require all previous values to be given: i.e. length requires width and height.

json or *xml* formatted replies must return an associative array with the key-name as specified

[1] RGB24 frame sizes, thumbnail 160x90: 337kiB, 768x576/PAL:10.1MiB, 1920x1080/full-HD: 47.5MiB. HTTP request/response headers are typically between 200 and 800 bytes.

```
#curl -d file=tmp/test.avi \
      http://localhost:1554/info
1
25.000
15262
0.0
1.833333
```

Figure 3: Example request of file information

in the reply format list. *html* or *txt* replies are intended for user readability and may differ in formatting, but should include the required information.

3.1.2 Image (preview, thumbnails)

Request parameters:

file file-name relative to the server's document root.

frame the frame-number (starting at zero for the first frame).

w (optional) width in pixels - default -1: auto

h (optional) height in pixels - default -1: auto

format (optional) image format and/or encoding

If neither width or height are specified, the original size (in pixels, not scaled with pixel or display-aspect ratio) must be used. If both width and height are given, the image must be scaled disregarding the aspect-ratio. If either width or height are specified, the returned image must be scaled according to the movie's real aspect ratio.

The default format is `raw` RGB, 24 bits per pixels.

3.1.3 Request video export, rendering

The `/stream` handler is used to encode a video on the server.

Request parameters:

file file-name relative to the server's document root or session-id.

frame (optional) the frame-number (starting at zero for the first frame) where to start encoding. the default is 0, start at the beginning.

duration (optional) number of frames to encode (minus one), negative values indicate no limit. A value of 0 (zero) must encode exactly one video frame. the default value is -1: until the end.

w (optional) width in pixels; default -1: auto

h (optional) height in pixels; default -1: auto

container (optional) video format; default : avi

nosound (optional) if set (1) only video is encoded - default: unset, 0

3.1.4 Server administration

Since the only way to communicate with the server is HTTP, specific interfaces are needed for administrative requests. The current standard defines only two; namely `re-start` (or cache-flush) and `shutdown`.

These commands are mostly for debugging and development purposes: The *clean shutdown* was motivated to check for memory-leaks; but comes in handy to launch temporary servers with each Ardour-session.

Cache-flush is useful if the video-file changes. Cached images must be cleaned and the decoders must release open file-descriptor and re-read the updated video-file. Future implementations should actually monitor the file for modification and do this automatically.

3.1.5 The Session API

Session API replicates `/info`, `/frame` and `/stream` request-URLs below the `/session/` namespace. Each request to those handlers must specify a `session` query parameter. The `file` parameter - if given - can be used to identify chunks inside a session.

Additional request handlers are needed to provide access to editing functionality, in particular asset-management and clip arrangement.

The detailed description of the session API is beyond the scope of this paper and will be published separately [2].

3.2 Extensions

3.2.1 Web GUI

The `/gui/` namespace is reserved for an end-user web-interface which is out of the scope of this paper. Currently a prototype using XSLT and icsd2's XML function is prototyped in XHTML

[2] A prototype of the session API for research is implemented in `libs/libsodan/jvsession.c` and `src/ics_sessionhandler.c`

and JavaScript and available with the source-code (see figures 4, 7).

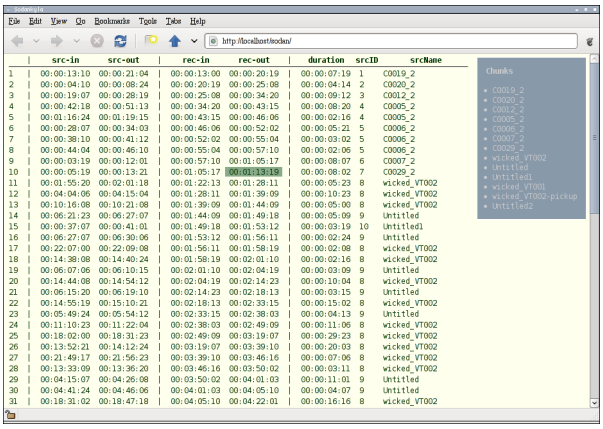

Figure 4: icsd2: JavaScript+XHTML EDL editor

3.2.2 Out-of-band notifications

Since it is not possible to use HTTP to send out-of-band notifications, it is not a suitable protocol for concurrent editing.

Both XMPP (Extensible Messaging and Presence Protocol) and OSC (OpenSoundControl) are options.

OSC has less overhead, yet only OSC over TCP provides reliable transport. XMPP has the advantage that by spec processing between any two entities at the server must be in order. Furthermore XMPP is a protocol intended for publish/subscribe, and capable to notify multiple clients. Authentication methods have also been defined for XMPP, whereas both authentication and Pub/Sub would require custom development on top of OSC.

3.2.3 Authentication and Access Control

Both HTTP as well as XML offer means of authentication below the application layer.

While some requests motivate access-control as part of the protocol - e.g. user-specific read-only access to certain frame-ranges or chunks - these can in general be handled by URI filtering, much as a .htaccess files does in apache.

3.3 Server Internal API

This section covers the planned interface for server-side sessions and non-linear video-editing. The back-end is partially implemented in `icsd2` and there is a basic web-interface to it.

3.3.1 Database and persistent session information storage

The server keeps a database for each session. A session consists of one or more chunks (audio or video sources) and a map how to combine them. Furthermore the session-database includes configuration and session-properties in a generic key-value store.

An extended EDL table is used to for mapping the chunks. It provides for (track) layering of chunk-excerpts (in-point, out-point). The data-model does not yet provide for per edit or per chunk attributes (zoom, effect automation) but transitions between chunks can be specified in EDL style. The current database schema can be found in source-code.

3.3.2 Server-side video monitoring

An important feature is to be able to to physically separate audio (Ardour) and video (server and monitor). This is motivated by various factors: All video-outputs of the client computer may be needed for audio, leaving none available for a full-screen video projector. It also preempts resource conflicts between audio processing and video-decoding.

To provide server-side video monitoring (video-display), two approaches are currently prototyped:

- monolithic approach: share the decoder and frame-cache back-end between the HTTP-server and the video-monitor using shared-memory.
- modular approach: share the source-code/libraries: HTTP-server and video-monitor are independent applications.

The former obviously requires less resource and puts less strain on the server (memory and CPU usage) at the cost of potential instability (resource conflicts) since maintaining cache-coherence complicates the code.

A middle-ground - building a video-monitor on top of the existing HTTP `frame/` interface - is unsuitable for low-latency seeks. Even though it performs nicely in linear playback it effectively purges the cache and introduces a penalty for time-line use.

With computing resources being easily available (and cheap compared to video-production in

general), a modular approach - using independent video-monitor software - is the way to follow.

3.3.3 Server to Server communication

A cache-coherence protocol MOESI (Modified, Owned, Exclusive, Shared, Invalid) is envisaged to distribute load among servers. An algorithm based on file-id and frame-number can be used to dispatch requests to servers in a pool. This can be done directly in the client implementation or by a proxy-server (http load balancer).

Server-to-server communication will use publish/subscribe to synchronize and distribute updates among servers in the pool.

4 Implementation

Historically things went a bit backwards. After the groundwork (Xjadeo, Ardour1) had been laid, different endeavours (gjvidtimline, xj-five) culminated in the video-editing-server software (vcsd) for Ardour3. The recurring need to focus on movie-soundtracks motivated the short-term goals of the current implementation: Ardour3 [Davis and others, 1999 2012] provides the DAW as well as the GUI. The video-server (icsd) project is named Sodankylä [Gareus, 2008 2012] after its place of birth.

4.1 client – Ardour3

The patch[3] can be broken out into small parts.

- Video-timeline display
- HTTP interaction with the video-server
- Xjadeo remote control
- ffmpeg interaction
- Dialogs (the largest part)
- Support and helper functions

The implementation is completely additive[4], and all video related code is separated by `#ifdef`s.

4.1.1 Video timeline

`gtk2_ardour/video_timeline.cc` defines the timing and alignment of the video-frames depending on the current scroll-position and zoom-level of the Ardour canvas.

The video-timeline is a unique "ruler" in Ardour (similar to bar/beat or markers) and globally accessible via `ARDOUR_UI::instance()->video_timeline`.
The video-timeline contains video-frame images: `gtk2_ardour/video_image_frame.cc`.

The timeline is populated by aligning the first video-frame. The zoom-level (audio-frames per pixel) and timeline-height * aspect-ratio (video-frame width in pixels) define the spacing of video-frames after the first frame. Images on the current page of the canvas as well as the next and previous page are requested and cached locally to allow smooth scrolling.

`VideoTimeLine` and `VideoImageFrame` are the only two classes that communicate with the video-server. The former is used to request video-session information (frame-rate, duration, aspect-ratio), the latter handles (raw RGB) image-data. The actual `curl_http_get()` function is defined in the file `video_image_frame.cc` which also includes a threaded handler for async replies of image-data.

The GET request-parameters to query information about the video-file are:
SERVER-URL`/info?file=`*fileURI*`&format=plain`
The GET request-parameters for image data are as follows:
SERVER-URL`/?frame=`*frame-number*
`&w=`*width*`&h=`*height*
`&file=`*fileURI*`&format=rgb`

The width and height are calculated by using the video's aspect-ratio and the height of the timeline-ruler. Current options are defined in `editor_rulers.cc` and include:

- "Small": $3H_t$
- "Normal": $4H_t$
- "Large": $6H_t$

with H_t = `Editor::timebar_height = 15.0;` pixels defined in `editor.cc`.

4.1.2 system-exec

Starting an external application with bi-directional communication over standard-IO, requires dedicated code.
`gtk2_ardour/system_exec.cc` implements a cross-platform compatible (POSIX, windows) API to launch, terminate and pipe data to/from child processes.

[3] `http://gareus.org/gitweb/?p=ardour3.git;a=commitdiff_plain;hp=master;h=videotl`

[4] No existing code is removed or changed, however the build-script and some of the documentation is modified.

It is used to communicate with Xjadeo [5] as well as to launch ffmpeg and the video-server (on localhost).

4.1.3 transcoding

`gtk2_ardour/transcode_ffmpeg.cc` includes low-level interaction with ffmpeg. It is concerned with locating and starting `ffmpeg` and `ffprobe` executables as well as parsing output and progress information. The command-parameters (presets, options) as well as progress-bar display is part of the `transcode_video_dialog.cc`.

4.1.4 Dialogs

The vast majority of the code contributed to Ardour consists of dialogs to interact with the user.

`gtk2_ardour/video_server_dialog.cc` asks to start the video-server on localhost. It is shown on video-import if the configured video-server can not be reached.

`gtk2_ardour/add_video_dialog.cc` Called from Session-menu → Import → Video, asks which file to import. The file-selection dialog assumes that the video-server runs on localhost or the server's document-root is shared via NFS[6]. The dialog offers options to transcode the file on import, or copy/hardlink it to the session folder. Furthermore there are options to launch the video-monitor and set the Ardour session's timecode-rate to match the video-file's FPS (see figure 5). `gtk2_ardour/video_copy_dialog.cc` is a potential follow-up dialog with progress-bar and overwrite confirmation.

If the video-server is running on localhost, the dialog also checks if the (imported) file is under the configured document-root of the video-server.

`gtk2_ardour/transcode_video_dialog.cc` allows to transcode video-files on import. It is recommended to do so in order to decrease the CPU load and optimize I/O resource usage of the video-decoder. The dialog also allows to select an audio-track to be extracted from the video-file and imported as audio-track into Ardour.

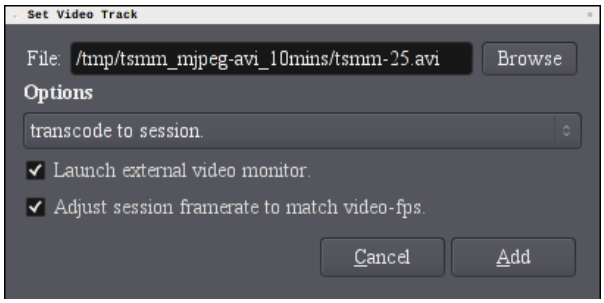

Figure 5: A3: the video-open dialog

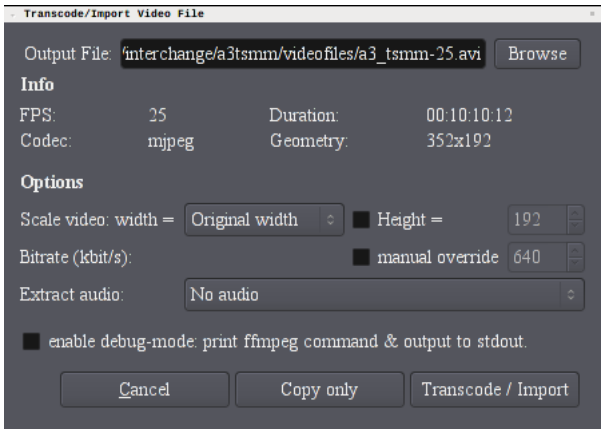

Figure 6: A3: the import-video, transcode dialog

`gtk2_ardour/open_video_monitor_dialog.cc` optional settings for Xjadeo. This dialog can be bypassed (configuration option) and allows to customize settings of Xjadeo that should be retained between sessions: e.g. window-size, position and state, On-Screen-Display settings, etc.

`gtk2_ardour/export_video_dialog.cc` is the most complex dialog. It includes various (hardcoded) presets for video-encoding and ffmpeg options. It also includes a small wrapper around Ardour's audio-export function.

4.1.5 video-monitor / Xjadeo

`gtk2_ardour/video_monitor.cc` implements interaction with Xjadeo. In particular sending video-seek messages if Ardour is using internal

[5] actually xj-remote which in turn communicates with Xjadeo, platform dependent either via message-queues or RPC calls

[6] A prefix can be configured to be removed from the selected path for making requests to the video-server.

transport and time-offset parameters if the video is re-aligned.

Switching Ardour's sync mechanism between JACK and internal-transport automatically toggles the way Ardour controls Xjadeo. Also Xjadeo's window state (on-top), on-screen-display mode (SMPTE, frame-number display) as well as the window position and size are remembered across sessions. Ardour sets override-flags in Xjadeo's GUI that disable some of the direct user-interaction with Xjadeo; in particular: you can not close the monitor-window nor load different files by drag-drop on the Xjadeo window or neither modify the time-offset using keyboard shortcuts. These interactions need to be done through Ardour.

4.1.6 misc

There are various small patches - mostly glue - to Ardour's `editor_*.cc`, `ardour_ui*.cc` as well as preference (video-server URL) and session-option (sync, pull up/down) dialogs. Dedicated video functions have been collected in `gtk2_ardour/editor_videotimeline.cc` and `gtk2_ardour/utils_videotl.cc`

4.1.7 libardour

While the video-timeline patch itself only concerns the Ardour GUI, a few helper functions semantically belong in *libardour*. They are almost exclusively related to saving preferences or resolving directories.

4.2 prototype server

The source-code includes a simple PHP script `tools/videotimeline/vseq.php` that emulates the behaviour of the video-server (no frame-caching) and implements the minimal interface protocol specifications. It is built on top of the command-line applications: ffmpeg, ffprobe and ImageMagick's convert.

4.3 video-server – Sodankylä – icsd

The video-server was born in Sodankylä June 2008. It's a motley collection of tools for handling video files. The 'image compositor socket daemon' (`icsd` - the names goes back to Ardour-1) implements a video-frame-caching HTTP server according to specs defined in section 3.1. `icsd2` in the same source-tree is a (work-in-progress) version adding the video-session and NLE capabilities.

By default `icsd` listens on all network interfaces' TCP port 1554 and runs in foreground, serving files below a given *document-root*.

```
usage: icsd [OPTION] <document-root>
```

The number of cached video-frames can be specified with the `-C <num>` option. It defaults to 128. The IP address to listen on can be set with `-P <IPv4 address>` (default: 0.0.0.0). There are various options regarding daemonizing, chroot, set-uid, set-gid and logging that allow icsd be started as system-service. They are documented in the manual-page.

Note that, icsd does not perform any access control. As with a web-server, all documents below the document-root are publicly readable.

The internal video frame-cache uses LRU (least-recently requested video-frame) cache-line expiry strategy. icsd is multi-threaded, it keeps a pool of decoders and tries to map consecutive frames from one keyframe to the same-decoder; however a new decoder is only spawned if the current one is busy in another thread.

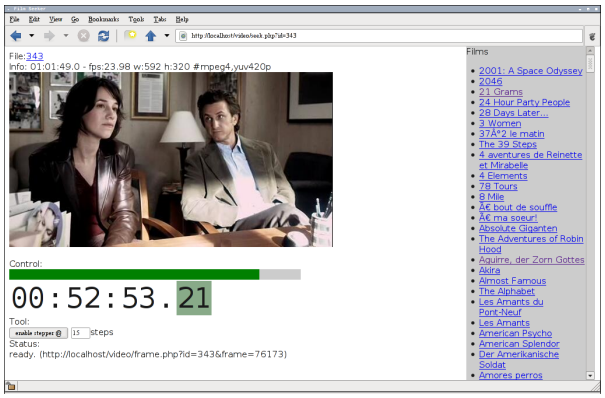

Figure 7: icsd2: JavaScript+XHTML video-monitor

`icsd2` extends the simple API towards video-session and non-linear-editing.

To allow interoperability with various client-side tools, a REST-API is being defined. XLST is used to export information in XML, as well as generate XHTML pages. For prototyping and debugging a simple web-player and JavaScript EDL editor came to be (screenshot in figure 7).

5 QA and Performance Tests

The most important part is to assure fitness for the purpose. Usability and integration is useless if fundamental quality is not sufficient. Criteria that require verification include accuracy, time-code mapping, as well as latency and system-load.

Depending on the zoom-level video-frames in the audio-timeline must be properly aligned. During playback the video-monitor must not lag behind. Audio latency must be compensated so that the video-frame on screen corresponds to the audio-signal currently reaching the speakers. The system load must remain within reasonable bounds and the applications should run smoothly without spiking.

5.1 Quality Assurance and Testing

Audio/Video frame alignment has been tested using the "time-stamp-movie-maker" [Gareus and Garrido, 2006 2012a]. It was used to create movies with time-code and frame-number rendered on the image at most common frame-rates (23.976, 24, 24.976, 25, 29.97df, 30, 59.94 and 60 fps) using common codecs + format combinations (mjpeg/avi, h264/avi, h264/mov, mpeg4/mov, theora/ogg). All 40 videos have been loaded into Ardour and tested to align correctly.

Figure 8: testing time-code mapping using a time-stamped movie

Latency compensation is verified by playing a movie of a unique series of black and white frames full-screen synchronously to an audio-track that sends an impulse on every white frame. The audio-impulse signal must occur aligned to the (luminance) signal of the video within the bounds of display frequency. Since there are 3 different clocks (display-refresh frequency, video frame-rate, audio sample-rate) involved the problem is not trivial. However audio sample-rate is an order of magnitude smaller than the usual video frame-rates and can be neglected. Systematic latency-compensation tests have successfully been performed at 25fps file of alternating black and white frames on a 75Hz display frequency. Other rates have been verified but not analyzed in depth.

5.2 Performance tests

Performance is evaluated by benchmarking unit-tests of the individual components as well as monitoring overall system performance.

The former includes measuring request latency to query video-frames which is further broken up into distributing workload on multiple video-decoders. An average session will spawn 16-20 video-decoders on a dual-core. The memory footprint of a decoder is negligible compared to the cached image data, yet it greatly improves response time when seeking in files to use a decoder positioned close to the key-frame.

Request and decode latencies have been measured with `ab` - The Apache HTTP server benchmarking tool as well as by timing the requests with Ardour. Cache hits are dominated by transfer time which is on the order of a few (1-3) milliseconds depending on image-size and network. Decoder latency is highly dependent on the geometry of the movie-file, codec, scaling and CPU or I/O. On slower CPUs (< 1.6 GHz Intel) a full HD video can be decoded and scaled at 25 fps using mjpeg codec width only intra-frames at the cost of high I/O. Faster CPUs can shift the load towards the CPU. Parallelizing requests increases latency to up to a few hundred milliseconds. This is intended behavior: image for a whole view-point page will arrive simultaneously.

6 User Manual

6.1 Setup

- get Ardour3 with video-timeline patch.
- install icsd (≥alpha-11) and optionally ffmpeg, ffprobe and Xjadeo (≥ 0.4.12) to $PATH.
- make sure that there is no firewall on localhost TCP-port 1554 (required for communication between Ardour and the video-server).

If you run the video-server on the same host as Ardour, no further configuration is required.

When opening the first video, Ardour3 will ask for the path of the video-server binary (unless

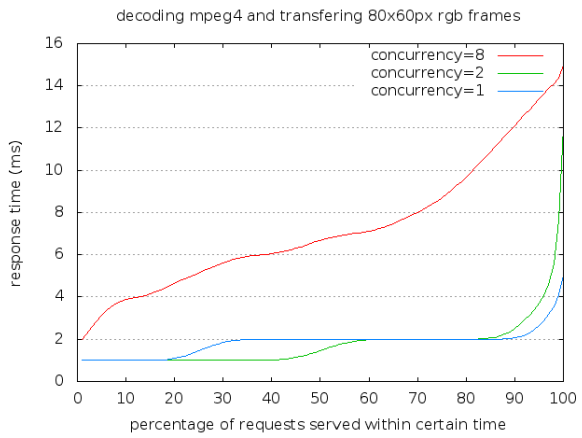

Figure 9: latency for decoding thumbnail frames for 1,2 and 8 parallel requests

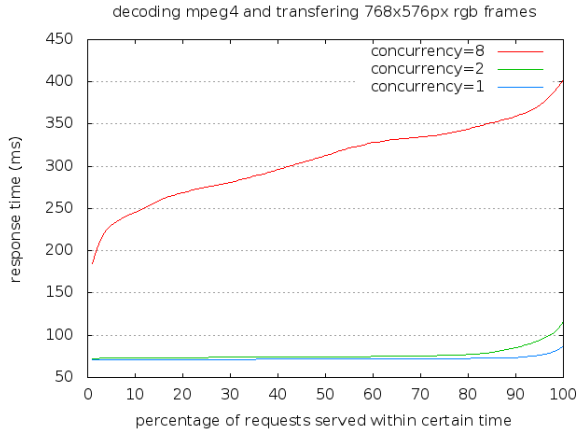

Figure 10: decoder and request latency for PAL video.

it is found in $PATH) and also allows you to change the TCP port number as well as the set cache size. Transcoding is only available if ffmpeg and ffprobe are found in $PATH (fallback on OSX: `/usr/local/bin/ffmpeg_sodankyla` - provided with icsd.pkg).

The video-monitor likewise requires `xjremote` to be present in $PATH or on OSX under `/Applications/Jadeo.app/Contents/MacOS/` and windows `$PROGRAMFILES\xjadeo\`. Xjremote comes with Xjadeo and Jadeo respectively.

6.2 Getting Started

It is pretty much self-explanatory and intended to be intuitive. All functions are available from the Ardour Menu. Select actions are accessible from the context-menu on the video-timeline.

- Menu → Session → Open Video
- Menu → Session → Export → Video
- Menu → View → Rulers → Video (or right-click the ruler/marker bar)
- Menu → View → Video Monitor (Xjadeo)
- Menu → Edit → Preferences → Video
- Menu → Session → Video maintenance → . . . (manual video server interaction)

A quick walk-through goes something like this: Launch Ardour3, Menu → Open Video. If the video-server is not running, Ardour asks to start it. You choose a video-file and optionally extract the original sound of the video to an Ardour track. By default Ardour also sets the session FPS and opens a video-monitor window.

From then on, everything else is plain Ardour. Add audio-tracks, import samples, mix and master,. . .

Eventually you'll want to export your work. Session → Export → Video covers the common cases from file-creation for online-services over producing a snapshot for the client at the end of the day to optimized high-quality two-pass MPEG encoding. However it is no substitute for dedicated software (such as dvd-creator) or a video engineer with a black-belt in ffmpeg or expertise in similar transcoding software, especially when it comes to interlacing and scan-ordering..

The export dialog defaults to use the imported video as source, note however that using the original file usually produces better results; every transcoding step potentially decreases quality.

6.3 Advanced Setups

The video-server can be run on a remote-machine with Ardour connecting to it. In this case you need to launch icsd on the server-machine and configure Ardour to access it. The server's URL and docroot configuration can be accessed via Ardour's menu → Edit → Preferences.

You should have a fast network connection (≥100Mbit/s, low latency (a switch performs better than router) and preferably a dedicated network) between the machine running Ardour and the video-server.

While not required for operation, it is handy to share the video-server's document-root file-system with the machine running Ardour. It is, however, required to

- browse to a video-file to open[7]
- display local video-monitor window
- import/transcode a video-file to the Ardour session folder
- extract audio from a video-file [7]
- export to video-file[7]

File-system sharing can be done using any network-file-system (video-files reside on the server, not the Ardour workstation) or using NAS. Alternatively a remote replica of the Ardour-project-tree that only contains the video-files is an option. The local project folder only needs a copy of the video-file for displaying a video-monitor.

The document-root configured in Ardour is removed from the local absolute-path to the selected file when making a request to the video-server.

Xjadeo itself includes a remote-control API that allows to control the video-monitor over a network connection. Please refer to the Xjadeo manual for details.

6.4 Known Issues

Video-monitor settings (window state, position, size) are sent to Ardour when the video-monitor is terminated. Ardour saves and restores these settings. The video-monitor is closed when a session is closed. If any of the video-settings have changed since the last save, session-close will ask again to save the session, even if it was saved just before closing. Every session-save should query and save the current state of the video-monitor to prevent this.

Operating Ardour with very close-up zoom (only one or two video-frames visible in the timeline), caching images only for the previous and next viewpoint is not sufficient. On playback video-frames are missing (black-cross images) in the timeline. The Ardour internal frame-cache should require a minimum of frames regardless of canvas pages to compensate for the request and redraw latency.

[7] later versions of the video-server and Ardour may not require this anymore.

7 Acknowledgements

Thanks go to Paul Davis, Luis Garrido, Rui Nuno Capela, Dave Phillips and Natanael Olaiz.

This project would not have been possible without various free-software projects, most notably ffmpeg/libav. Kudos go to the Ardour and JACK development-teams and to linux-audio-users for feedback.

Last but not least to Carolina Feix for motivating this project in the first place.

References

Rui Nuno Capela. 2005–2011. Qtractor. `http://qtractor.sourceforge.net/`.

Paul Davis et al. 1999–2012. Ardour. `http://ardour.org`.

Paul Davis et al. 2001–2012. Jack transport. `http://jackaudio.org/files/docs/html/transport-design.html`.

Gabriel Finch. 2004–2012. Lives. `http://lives.sourceforge.net/`.

Blender Foundation. 1999–2012. Blender. `http://blender.org`.

Robin Gareus and Luis Garrido. 2006–2012a. Time-stamp-movie-maker. `http://xjadeo.sf.net`.

Robin Gareus and Luis Garrido. 2006–2012b. Xjadeo - the x-jack-video-monitor. `http://xjadeo.sf.net`.

Robin Gareus. 2008–2012. Sodankylä, icsd. `http://gareus.org/wiki/a3vtl`.

Heroine Virtual Ltd. 2002–2011. Cinelerra. `http://cinelerra.org/`.

Piero Scaruffi. 2007. *A brief history of Popular Music before Rock Music*. Omniware.

Werner Schweer et al. 1999–2012. Musescore: Music score editor. .

Wikipedia. 2011. Film-score. `http://en.wikipedia.org/wiki/Film_score`.