THE UNIVERSAL PRIMER: AN OPEN SOURCE SOLUTION FOR ARCHIVING, ORGANIZING AND STREAMING LIVE LECTURES

Marc Juul Christoffersen, Maciej Krajowski-Kukiel, Christian Panton, David Christian Askirk Fotel, Henrik Madsen, Lasse Engbo Christiansen, Povl Ole Haarlev Olsen, Halfdan Mouritzen and Janne Kofod Lassen

Technical University of Denmark

ABSTRACT

Many disparate projects providing open access to educational videos are currently available or under development. These projects lack a unifying interface for accessing content, employ differing content licenses, and provide little or no infrastructure for user-contribution or live teaching. The goal of the Universal Primer is to address these problems, and allow anyone, anywhere, to teach or learn anything that can be reasonably taught or learned through a computer. The Universal Primer is 1: A fully open source solution for streaming live lectures. And 2: A Wikipedia-like website for uploading and organizing open-licensed community-contributed educational material.

KEYWORDS

online, video, streaming, community, open-source.

PROJECT SCOPE AND REQUIREMENTS

The Universal Primer attempts to solve a very large problem, in essence, creating a system for free access to computer-based education to the fullest extent possible. Fully meeting this challenge will take a lot of time and resources, yet building even a small part of the whole system could present a valuable contribution to free and open education. This describes the result of a phase one, or version 0.1 of an ongoing project.

The project was split into two main sub-projects, with the goal of future integration:

1. The live-lecture project: Facilitating live interactive teaching.
2. The wiki-site project: Giving access to prerecorded material.

The scope of the first phase of the project was limited to focus mainly on video and tightly coupled materials such as lecture slides, as this was seen as the most challenging immediate priority. It is the future goal of the project to include support for formats such as Wikipedia-like web-pages, typeset documents and interactive components.

The primary goal of the first phase was specified in the form of a use-case: Use the Universal Primer to teach a traditional university classroom lecture in a statistics course. This goal was then expanded upon to list the minimal features it would take to satisfy the use-case:

1. One live audio and video stream from lecture.
2. Ability to show slides simultaneously with video.
3. Ability to annotate and/or point to features on lecture slides during the lecture.
4. Interactive communication with remote students in the form of text-based chat.

Requirements were added concerning the required software and hardware. The first requirement is that the students attending the virtual lecture should not be required to have anything more than an internet-connected computer with a modern standards-compliant web browser, and the second: That everything the lecturer needs should be affordable, easily available in any part of the world and require only minimal technical understanding to set up and use.

Finally, an important requirement was that the system be built with the knowledge that it should eventually be able to scale to a size such as has been accomplished with Wikipedia. The bandwidth required for video, coupled with the interactive elements of live teaching, make this a unique challenge. It would not be feasible to build such a system using a traditional, centrally administrated architecture as is used to support video sites like YouTube, since the cost of bandwidth alone would likely overwhelm the project beyond the ability for donations or even ads to support it. Rather, the system should be built with highly decentralized technologies where possible, in order to spread the cost of bandwidth, processing power and storage, across a large network of participating computers.

The second goal of the first phase was to develop a wiki-like site for uploading and viewing prerecorded video lectures. Other than a simple video upload and viewing site, of which there are many, the site should serve to organize large amounts of educational material by subject, using a system whereby users can ask to learn about a specific subject, and the site will inform the user of a suggested sequence of other topics or materials with which the user should be familiar before attempting to learn the subject at hand. If implemented on a large enough scale, this would enable the site to answer questions such as "I know subject X and I want to learn subject Y; How do I get from where I am to where I want to be?". A requirement for this system was that suggestions given to the user regarding which additional subjects may depend on the subject currently viewed, must be specified in a collaborative manner by the users, as opposed to being centrally specified by a group of administrators. Inspiration for this system comes in part from Tutor-web [1], an existing web-based educational platform that implements a dependency system.

OPEN STANDARDS AND OPEN LICENSING

A prime motivator for this project was the lack of good open source, open standards solutions for live teaching, so a requirement for the project was that all software must be free software (as specified by the free software definition [2]). The target license for the software is the GPLv3 or AGPLv3 Copyleft license, though some components that build on existing free software projects will have to keep to the licenses employed by those projects. As for open standards, the goal is to build something that does not employ any proprietary standards or technologies, and to use open standards where technically justifiable. During the course of development, it was deemed necessary to temporarily allow the use of proprietary technologies in certain cases where the open equivalents did not yet provide the necessary functionality.

The required license for all contributed content is Creative Commons Attribution Share-Alike 3.0. This is the same license used by Wikipedia, and fits well with the spirit of both the Open Source [3] and Free Software definitions. This choice excludes some existing educational videos, such as the MIT OpenCourseWare material [4] with more restrictive licenses such as those Creative Commons licenses containing the "non-commercial" clause. Such exclusion will limit the amount of available material for the project, but the minimal restrictions will hopefully encourage more people to use the material, even in for-profit educational settings, and contribute their improvements and additions back to the project.

DESIGN – LIVE LECTURE PROJECT

Overview

The live lecture project can be divided into three high-level units:

1. The presentation software: For broadcasting a live lecture.
2. The server software: For facilitating communication between the presenter and students.
3. The student software: For participating in a live lecture.

These high-level units are further sub-divided as seen in the following diagram and explained in subsequent sections:

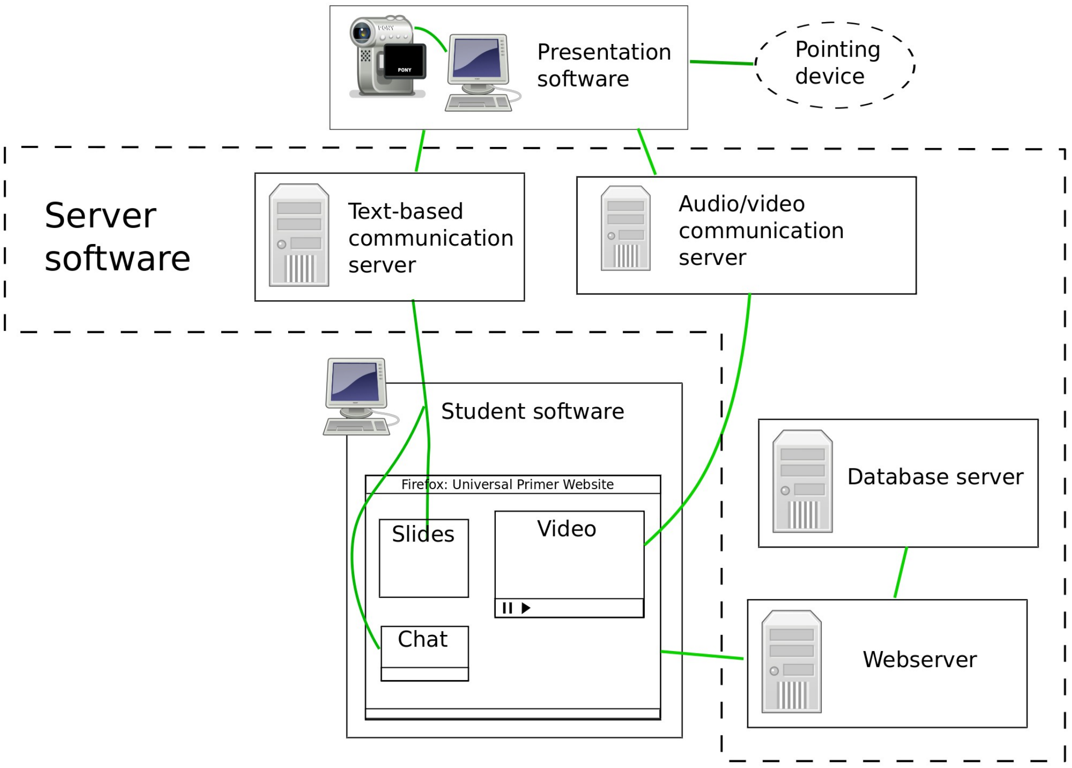


Illustration 1: Design of the live-lecture project. Node: Slide-changes and annotations are sent as text-based communication. The actual slides are uploaded as a pdf via the web-app.

Presentation Software

The presentation software receives live audio and video from a video camera, encodes it to minimize the required bandwidth, and sends it to an audio/video server that forwards the streams to students. The presentation software allows the presenter to switch between a set of slides and annotate the slides using a pointing device. These slide change and annotion events are communicated to the students through a text-based communication server. The presentation allows two-way communication with the connected students in the form of a text-based chat through the same server.

The presentation software is a standalone GUI application. The optimal solution would be to implement it as a web-based application, but the limitations inherent in current web browsers would make some of the current and future goals difficult to implement, including input from DV and HDMI video cameras. To keep things simple for the first phase of development, a stand-alone application that runs on Ubuntu Linux was deemed sufficient. The application was developed to be easily portable to the Windows and OS X operating systems; a goal for future versions.

For slide annotation functionality, the presentation software accepts input from an input device, e.g. a mouse.

Server Software

The server software receives all of the data sent by the presenter software, video streams and text-based data, and sends this data to all connected students. The software also receives chat messages from students and sends it to the presenter software and to all other students. The server software also hosts the website for the student software including the database for the website.

The server software consists of at least four separate applications.

1. The text-based communication server.
2. The audio/video communication serer.
3. The web server.
4. The database server.

The webserver and database server are fairly straightforward, together hosting the web-based student software. The text-based-communication server handles all text-based messages sent between presenter and/or students. These messages include chat messages, slide change messages, and slide annotation messages The text-based communication server also implements the concept of channels. The channel functions in the same way a chat channel does in most chat software. Each presentation has one associated channel, and the student and presenter software requests to join the channel when the user connects to a presentation. The text-based communication server forwards all messages sent with a specific channel as the recipient to every client that has joined the channel. The audio/video communication server allows the presenter to send a stream to the server that is then forwarded to the connected clients (student software) that request it.

Student Software

The student software is web-based. It receives and displays the data sent by the presentation software from the server: Video stream, slide-changes, slide annotations and chat messages. In addition, it allows the students to send chat messages to the server software, which is forwarded to the presenter and all connected students.

Modules

The live lecture system is designed to be highly modular in terms of functionality. Video, slide and chat functionality is implemented as three separate modules. Modularity in this context is independence of functionality that allows any one module to be removed or added, without affecting the functionality of any other module. Many modules will require code to be written for both student and presenter software, and in some cases even for the server software as well, so it is not required to be modular in the sense that it is a single piece of code. A future goal for the project is to encourage users and/or developers to write their own modules that facilitate the teaching of particular subjects and enables a variety of teaching styles. An example of such a module that is currently under development is the Piano module. The first version of the Piano module will accept midi input on the presenter side and displays the music notation for the played notes for the students as the teacher plays. Ideas for future modules include a screencasting module and a module for interfacing to smartpens such as the livescribe [5] allowing teachers to share hand written notes with students in real time. The modules are implemented such that the student software chooses which modules to enable, and the server software only sends and receives data for the modules enabled by the student software.

On Decentralization

It may seem odd to employ such a centralized server-based design, given the focus on a scalable peer-to-peer solution. This choice was made in part due to a lack of browser-based peer-to-peer support, but also as a way of limiting the scope of the project. Though peer-to-peer in-browser streaming video solutions do exist, these technologies are not yet adapted to live streams where large delays are unwanted, and require third-party plug-ins that are either proprietary or not yet widely deployed [6]. It is also reasonable to assume that, even with full peer-to-peer support in a future version of the software, a fall-back option in the form of a traditional server-based approach will be needed to ensure a dependable and backwards compatible service for quite some time to come. At the same time, Universal Primer developers have begun developing and testing a Java applet based peer-to-peer in-browser solution for the Universal Primer, which may in the future be able to provide peer-to-peer streaming video support.

IMPLEMENTATION – LIVE-LECTURE PROJECT

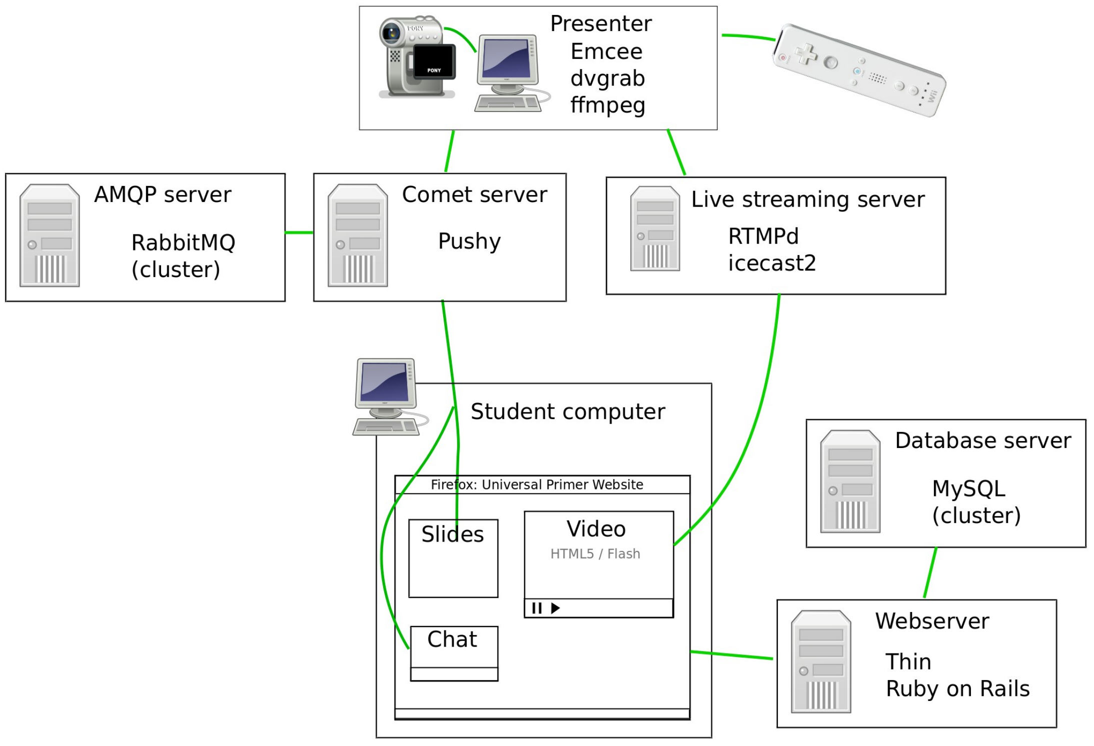


Illustration 2: Implementation of the live lecture design

Server Software

The server software facilitates communication between the student software and the presenter software. The server software consist of four applications:

1. Icecast2 / RTMPd: The streaming video server.

2. Pushy: The Comet server.
3. RabbitMQ: AMQP Server.
4. Thin: The Ruby on Rails webserver.
5. MySQL: Database for Ruby on Rails.

The Thin [7] and MySQL [8] servers are required to host the student software web app, which is discussed in further detail in the next section.

Streaming Video Server

Icecast2 [9] is an open source streaming media server capable of live video streams over HTTP suitable for the HTML5 video support built into modern browsers. As will be explained in the next section, the HTML5 video support proved problematic. For now, the Universal Primer uses Flash video. RTMPd is an open source streaming media server with support for the Real Time Messaging Protocol (RTMP) developed by Macromedia (now Adobe) and used for live streaming for Flash clients [10].

The RTMPd server is configured to receive a video stream from the presenter software over a TCP connection and send out the video over RTMP to all connected clients.

The video stream from the presenter software is in the FLash Video (FLV) [11] container format. The video format used is H.264 [12] and the audio format is MP3 [13]. The stream is sent as-is to clients, except for the change of protocol to RTMP.

Pushy and AMQP

Pushy is a Comet server: A specialized HTTP server that facilitates two-way communication with JavaScript running in a web browser. Whereas a normal HTTP server uses a request-response model, with all requests originating from the client, in a Comet server a connection is kept open and either server or client is free to initiate communication. Pushy is written in Ruby using the Rack framework [14]. It is based on the code from the "Pusher" Comet server developed by Marc-André Cournoyer [15].

The purpose of Pushy is to facilitate communication for the modules that rely on text-based data, including the chat module and the slide module. To accomplish this, it must solve two main problems. One problem is communicating with the student software and presenter software, and the other is implementing the concept of "channels" that ensures that data received from a student or presenter in a specific presentation is only forwarded to the other students and presenters connected to that presentation, similar to normal chat channels, e.g. Internet Relay Chat (IRC).

To facilitate communication with a wide range of web browsers, Pushy implements a set of different techniques, referred to as transports [16]. These transports are:

1. XHR Streaming: For Firefox, Safari and Chrome support.
2. HTMLFile: For Internet Explorer support.
3. Server-sent Events: Part of the HTML5 specification. Adds Opera support.
4. Server-sent Events draft: Needed to support older Opera browsers.
5. Long polling: For older browsers in general.

Pushy communication is based on simple JSON (JavaScript Object Notation) formatted messages.

To implement the concept of channels, Pushy interfaces to a RabbitMQ process [17]. RabbitMQ is an open source application that supports the Advanced Message Queuing

Protocol (AMQP) [18]. AMQP can be used to implement complex scenarios of message routing and queuing, which will likely be needed to meet demands for advanced features such as private messages and privilege levels, in future versions of the Universal Primer software. For now, the implementation is fairly straightforward. It is outside the scope of this report to explain the specifics of AMQP. Generally: Channels are implemented by using one named fanout exchange per channel, with each connected student or presenter creating a nameless queue and binding it to the fanout exchange. Sending messages to the channel is accomplished by publishing to the exchange, receiving messages by reading messages from the queue.

On Scalability

The technologies used, though based on the traditional client-server model, rather than the more scalable peer-to-peer mode, were chosen with some care to allow the system to scale. The MySQL server can be scaled using the MySQL cluster support [19], though it may be beneficial to move to a more scalable database solution such as the NoSQL database Apache CouchDB. RabbitMQ has good built-in clustering support, which should allow it to scale well, and RTMPd can be set up to forward video streams to other RTMPd nodes. The rest of the server software does clustering support in order to scale, as the relevant data is kept in sync between multiple different processes by the common database cluster (MySQL) and AMQP cluster (RabbitMQ).

Student Software

The student software used to view and participate in the presentation is fully web-based and consists of a server-side web application and a client-side JavaScript application.

The server-side web app was built using the Model-View-Control framework Ruby on Rails 3.0 [20]. The web app uses a database back-end that can be a variety of SQL servers, such as the open source PostgreSQL server [21]. The web app has four core pieces of functionality:

1. User login and new user sign-up.
2. Upload of PDF slides and conversion into sets of images.
3. Serving of the client-side HTML, CSS, JavaScript etc. that handles the actual presentation.
4. Exposing a web service to fetch the URL for specific slide images.

The implementation details of this web app are mostly trivial and will not be covered in this document, except for the pdf to image conversion and handling. The conversion is handled by the GraphicsMagick application [22]. This conversion is necessary because web browsers do not have PDF viewing capability built in. The images are generated once and saved to the server when the files are uploaded. The base path name for the images is saved to the database, allowing the web service to return the URL for the image of a specific slide on demand.

The client-side application is mainly written in a combination of JavaScript, HTML and CSS. It is implemented as a single HTML web page, with the associated JavaScript, HTML and CSS residing in external files and with a minimal amount of in-line Ruby code to allow the server-side application to check if the user is logged in.

As explained in the design section, there are three implemented modules: The video module, the slide module and the chat module. The implementation of these modules is explained in the following section.

GUI

The current GUI is exceedingly simple, and does not allow for presenter or students to resize or move the video or slide displays.

A new GUI is under development that will both give a more professional consistent look and feel, and allow more flexibility in resizing and moving the different modules. A screenshot of the work in progress:

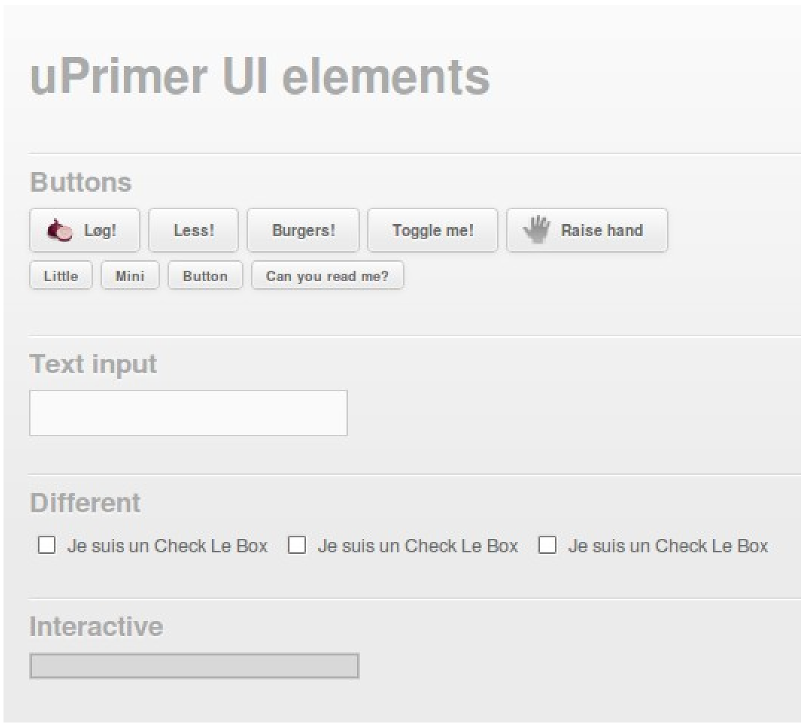


Illustration 3: Elements of the new GUI

This new GUI will be very task oriented. The design is as follows:

A minimalist interface displays video (one or more streams), slides, chat, interaction with lecture and other interface elements suitable for the subject.

The interface will be minimal and to some degree configurable - if there for example is a video stream and a slide viewer, one of these can be made central and big in the interface, while the other is smaller. It will be easy to switch these. It will also be possible to have several streams show up "big", if the screen real estate permits this.

The Video Module

It was attempted to implement the client-side of the video module using an HTML5 video tag, but after many tests with different configurations it was concluded that HTML5 video, though it works acceptably with prerecorded video files, is not ready to be used for live video streams. Observed problems included frequent and sometimes undetectable stalling, which is not acceptable in a live presentation situation. A HTML5 video solution will be re-evaluated when the technology is more mature.

Instead, an Adobe Flash application was used for the video module. Though Flash is a proprietary technology, this was deemed a necessary temporary compromise. The open source FlowPlayer application [23] was used in this first version. FlowPlayer was deemed overly complicated for the purposes of this project, and a team-member began to develop a simple Adobe Flex based video player, that would be controllable by a javascript API equal to the HTML5 Video component, readying the application for future drop-in replacement of the Flash solution with an HTML5 solution.

The Flash player connects to the server side RTMPd server using RTMP, and starts video playback as soon as the presenter starts sending video.

The Slide Module

The slide module was implemented as a simple HTML image tag showing the current slide. This has the drawback of disabling such features as copy-paste and zoom (an improved solution is discussed in the "Future Work" section). Changing to a different slide from the presentation is controlled by the presenter. The slide module receives a message, sent by the presenter software, telling it to change to a different slide, given by a page number, asks the server what the URL for the image associated with the slide is, and then changes the image source of the HTML image tag to the new image path, updating the displayed image. The communication with the presenter software is explained in one of the later sections.

Annotations are received as a series of messages, each commanding the slide module to draw a line segment between to points given by two coordinates. An SVG component is overlaid on the image using the cross-browser vector-drawing library Raphael [24], and the specified line is drawn across the image for each received message. When a message is received to clear all lines, the equivalent Raphael API function is called and the SVG overlay is cleared. When the current slide is changed to a different page, the SVG overlay is also cleared. Annotations are not remembered when switching back to previous slides.

The Chat Module

The chat module GUI is implemented as an HTML unordered list, a textbox, and a button. The user first fills in the desired nickname on the chat and clicks the button; all subsequent clicks of the button send messages to the other chat participants for the current presentation. As chat messages are received, the chat module Javascript appends the messages to the list.

JavaScript Comet Library

To communicate the slide change, chat and slide annotation messages, a JavaScript library was developed, based on the Pusher code, to allow for robust cross-browser communication with the Pushy Comet server. This Comet JavaScript library includes support for the five different transports implemented for the Pushy server.

The Comet library includes an API for connecting/disconnecting from/to a channel on the Pushy server, including a parameter for, if, and how many times the library should attempt to auto-reconnect when a disconnect is detected. The following callbacks are implemented in the API:

1. on_connect: Called when connection succeeds.
2. on_connect_retry: When an auto-reconnect attempt is begun.
3. on_reconnect: When an auto-reconnect attempt succeeds.
4. on_disconnect: When an unexpected disconnect is detected.
5. on_receive: When a message is received.

6. on_ping: When a keepalive ping is received from the server.
7. on_error: When an unexpected error is detected.
8. on_unload: When the library detects that the current web-page is being unloaded.

Finally, the API includes a function for sending messages to the channel.

On top of this API, a simple protocol was implemented for handling chat module and slide module communication. The protocol was named CHIMP, an homage to the APE Comet server that was used in the early days of the Universal Primer. CHIMP defines the following messages:

1. slides/change: Changes the current slide to a slide specified by page number.
2. chat/public-msg: Sends a chat message to the channel.
3. draw/lines: Draw a line as annotation on the current slide.
4. draw/lines-clear: Clear currently drawn lines.

Since Pushy uses a JSON-based protocol, these messages are also specified as JSON. Messages are sent to their corresponding module based on the part of the message name before the "/".

Presenter Software

Due to the limited functionality of the current open standards based browser technologies, a stand-alone Python application was developed. Its purpose is to support the teacher in presenting and using video, slides and chat. The application is named Emcee after a Master of Ceremonies: The MC. Python [25] is a very high-level language and was chosen for its rapid development features. Among those are its large standard library, excellent documentation and broad usage. For the GUI, the Qt4 [26] library was chosen. Like Python it is licensed under the GNU General Public License, is freely available and is implemented on all major platforms, ensuring cross-platform compatibility. Although the application is being developed with a target platform of Ubuntu Linux 10.04, it will lower the effort required to port the application to another platform. The Qt4 C++ library is accessed though the PyQt4 bindings. Another set of bindings, PySide, is currently being developed by Nokia and is largely API compatible with PyQt4, but was disregarded due to the lack of maturity of the project. The application is incorporating a large number of different APIs, hardware and network communication and a lot of glue-code, to keep everything working together. The focus has been on developing the glue code and using existing libraries when available. While the core Python library is very well documented, a lot of the third party libraries are sparsely documented – often only using code examples.

Functionality overview

The main screen is depicted in figure 1. It shows the presenter the current slide and the upcoming slide, side by side. Additionally a list of all the slides is kept in a sortable list below the upcoming slide. This enables the presenter to sort the slides while giving a presentation. By clicking on a slide this also enables the presenter to set the upcoming slide, i.e. to skip a series of slides, go back in the presentation during an Q&A session, etc.

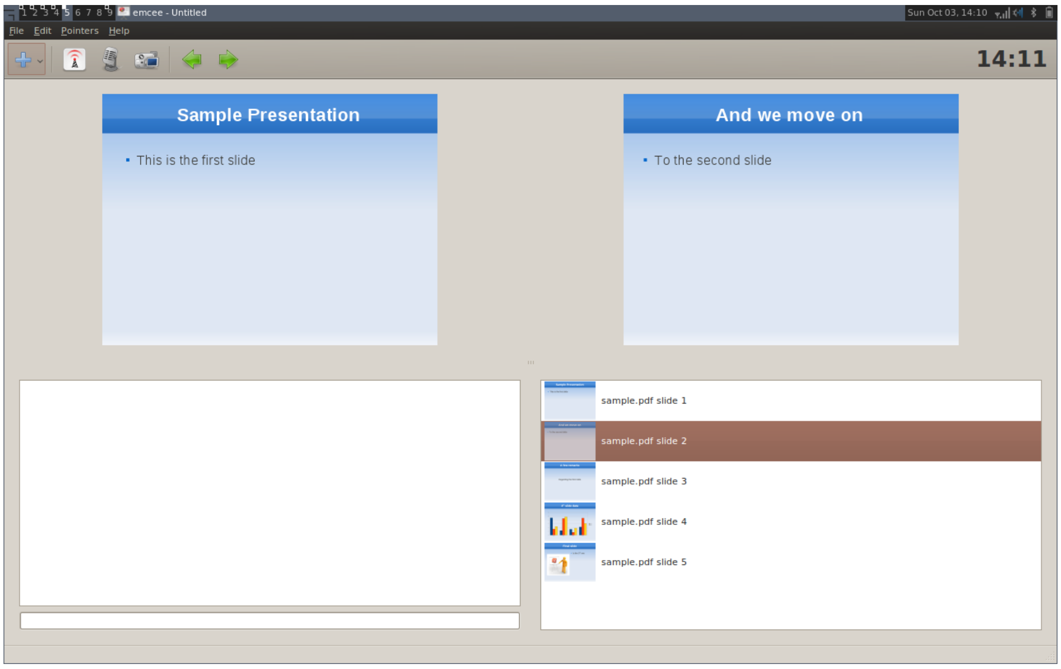


Illustration 4: The main window of Emcee. Upper left quadrant: Current slide view (also mirrored on a secondary screen). Upper right quadrant: Upcoming slide. Lower left quadrant: Chat and questions window. Lower right quadrant: Next slide chooser, slide sorter and list.

On the main screen there is also a chat window, which makes it possible for the presenter to write text based messages to the distance learners and makes it possible for the distance learners to ask questions to the presenter. A toolbar is also available for adding additional slides, toggling of video, audio and slide broadcasting and to provide information, such as the current time. The control of the presentation slides (going forward, etc.) and the types of media available, are controlled though a simple plug-in system. Python scripts residing in the plugins/ folder are automatically loaded into to program. The currently supported plug-in types are pointing devices and slide content.

Pointing Devices

Pointing devices is a broad category of keyboards, mice and remote controls for controlling the slides and annotating the presentation. As they are plug-in based, two pointing devices were implemented. The first one is a simple keyboard based flow control. It only implements a forward and backwards button, either as the arrow keys or the page-up and down keys. The main reason for also implementing page-up and down keys, is that a large number of wireless remote controls use these keys. It was tested using a cheap off-brand 2.4GHz wireless remote control [27] which enumerates as a USB HID keyboard on the host computer. The second device is much more sophisticated. Originally developed for the Nintendo Wii gaming console system, the Wii Remote or Wiimote is ideal as a pointing device. The device can be seen in illustration 5.

Illustration 5: The Wiimote (right) and the Nunchuck (left). The Nunchuck is currently not used by the Universal Primer software.

The Wiimote is using the wireless Bluetooth communication protocol, which means that it can communicate with most modern laptops. It features a large number of buttons, 3-axis accelerometers, an I^2C communications port (for the Nunchuck, and other peripherals), vibration motor, LEDs and a 4 point-tracking infra-red camera. A number of Python libraries emerged when developers discovered how the communication protocol worked, but most of them were abandoned. The CWIID [28] library was chosen for its maturity, as it is available as a binary package for most Linux distributions. One thing all the libraries have in common is the lack of proper Bluetooth pairing [29]. The Wiimote can be placed in two modes. Host (PC) initiated communication and Wiimote initiated communication. Only the host-initiated is implemented, as it is much simpler to use, although the Wiimote has to be put into discoverable mode every time you want to connect to it. This is done by pressing the 1 and 2 buttons on the remote simultaneously. Although this is a cumbersome procedure, this skips the step of host-remote pairing and makes sure that Wiimotes can be used interchangeably between computers.

The Wiimote is used for controlling the flow of the presentation, using the left and right arrow keys. The vibration motor is used to get the attention of the presenter, when, for example, there is a question from a distance learner. But the primary reason to use a Wiimote over a much simpler remote, is the infra-red point-tracking camera. By placing two clusters of IR-diodes above or below the projector screen, the Wiimote can be used as a digitized laser pointer. The IR diodes will show up as two points on the IR-camera and its build-in computer vision algorithm sends the positions of the diodes to the host computer. A simple algorithm was written based on the midpoint of the two points and it includes a small correction for rotation of the Wiimote. Pressing a button on the remote enables the presenter to draw on the screen. A vertex reduction algorithm is used to limit the amount of points created by the freehand drawing. The Wiimote was intended to be used like this as a mouse, but another configuration is also very popular: Although not implemented in this software, if the Wiimote is placed on a stand, using a pen with an IR-diode in the end, the projector screen can be used as blackboard [30]. This alternative configuration is worth considering in a future revision of the program, as the accuracy of the drawing is much higher. Using the plug-in system, it is a trivial task to add such functionality. The digitization of the pointing device is important, as it delivers the same teaching experience to the distance learners as it does to the people in the classroom. The distance learners will be able to see the gestures drawn on the slides by the presenter.

Slide Media Support

In order to support a large amount of different content on the slides, such as PDF files, YouTube videos, etc., the slide content is defined as a plug-in system as well. All content plug-ins are defined by the MIME-type they support. As a matching server-side plug-in is needed, the MIME-type serves as a plug-in identifier. Implemented in the application are two types of plug-ins. Like with the pointer plug-ins above, a simple and a more advanced plug-in is available. The simple plug-in is a blank slide. Its MIME-type is the non-standard application/x-blank-slide and can be used to clear the projector between multiple presentations or in a Q&A session.

The more advanced plug-in is the main content plug-in. It is based on PDF files and has the standard MIME-type application/pdf. Rendering of the PDF files is done by the libpoppler librar with a set of Python/Qt4 bindings called pypoppler-qt4. A patch was written for the bindings, as they did not expose the Table of Contents of the PDF file, which is often used to provide a slide name, usually set by presentation-creating software.

Broadcasting

The main features of this application are its broadcasting and interactive features. They cover slide switching, slide annotation, chat and audio/video broadcast of the presentation to the server software. For the text-based communication (slide switching, slide annotation and chat), a client for the CHIMP protocol was implemented in Python. This was implemented over HTTP using the XHR streaming transport. Future versions should rely on either raw Sockets or WebSocket support [31] as there is no reason for implementing, outside of a web browser, what is essentially a web browser-specific hack.

For video broadcasting, the application supports a connected DV [32] camera over an IEEE-1394 [33] (also known as FireWire) connection. To handle the video input, the dvgrab program [34] was modified and compiled as a library. This library, written in C, was called from Python using the CTypes library in order to read video from the camera. The video is then forwarded to the ffmpeg program [35], called as a sub-process, which transcodes the video from DV to H.264 video with MP3 audio inside of an FLV container. The ffmpeg also handles sending of the FLV video to the RTMPd process running on the server. The current version of the application does not yet support high definition video or input sources other than DV cameras, though support for webcams and raw HDMI input is planned for upcoming releases.

In order for more than one presentation to take place simultaneously on a single server, it is necessary to name them. All broadcast information is associated with the presentation name specified by the presenter before it is sent. For the video stream, this is implemented using a patched version of ffmpeg that allows setting of the "stream name" feature of FLV streams. The stream name is simply set to the presentation name. For the CHIMP protocol, the channel name, as discussed in previous sections, is simply set to the presentation name, such that each presentation has exactly one channel.

DESIGN – WIKI-SITE PROJECT

The main challenges in developing the wiki site were identified as:

1. Developing a system for collaborative organization of available material.
2. Handling video contributions in a multitude of formats.

Other challenges, such as handling collaborative editing of text and images, has been solved by existing open source solutions such as MediaWiki [36] and was not included in this first phase of development.

Organization

The organization of available material was built around the idea of a dependency system. As an example: The user chooses to learn how to multiply, but the website says that in order to learn how to multiply, the user should learn how to add numbers first. But, to learn this, the user should have knowledge of numbers. In other words, a dependency system is a recommendation system, which suggests to the user the order in which to read/watch the materials that are available in the system. As an example, Tutor-web [1] is an existing dependency system, which today containsorganized information in relation to mathematics and statistics.

Specifying a list of dependencies for each piece of contributed material, and displaying these dependencies, is technically fairly straight forward. The challenge of a dependency system in the context of the Universal Primer is to build an interface that makes it easy for a large group of contributors to collaboratively define the dependencies.

There are many possible approaches to this problem. This first phase of development has explored a tag-based approach.

The tag-based approach idea is fairly simple: Material on the site is organized into named pages, and users are given the option of adding tags to the pages. Dependencies can be added to pages either by manually searching or by choosing from a list of suggestions that are generated by the site based on page tags.

Prerecorded Video

Goals

One of the goals of the Universal Primers is to enable as many people as possible to watch and contribute prerecorded video content. This would be simple if all video was in the same format, but in reality is complicated by two factors:

1. No single video format is supported for playback on all browsers and on all platforms.
2. User-contributed video will be uploaded in a variety of open and proprietary formats.

This situation can be dealt with either by restricting the formats supported by the system, or developing a system for automatic conversion between formats. The Universal Primer went with the latter approach. User-contributed video must be converted from the existing format, to one or more formats that allow the video to be displayed in the most commonly used browsers.

For user-contributed videos, the optimal list of supported formats is simple: All formats. This may not be attainable in practice, but the limiting factor will simply be the availability of free/open software that is able to decode the video.

The problem of deciding on an optimal list of video formats needed to support the most used browsers required some testing. A set of video formats were tested on several different browsers, browser plug-ins and operating systems. A minimal list of video codecs was compiled based on a need for support across the most commonly used browsers and browser plug-ins. The actual list used by the project when it launches for wide public use at a later date, will have to take into account any applicable license fees associated with the use of proprietary codecs, and whether it is even desirable for such a project to support non-open formats.

A system was designed to include functionality for upload, conversion and viewing of video formats, using a web-based interface (shown in Illustration 6).

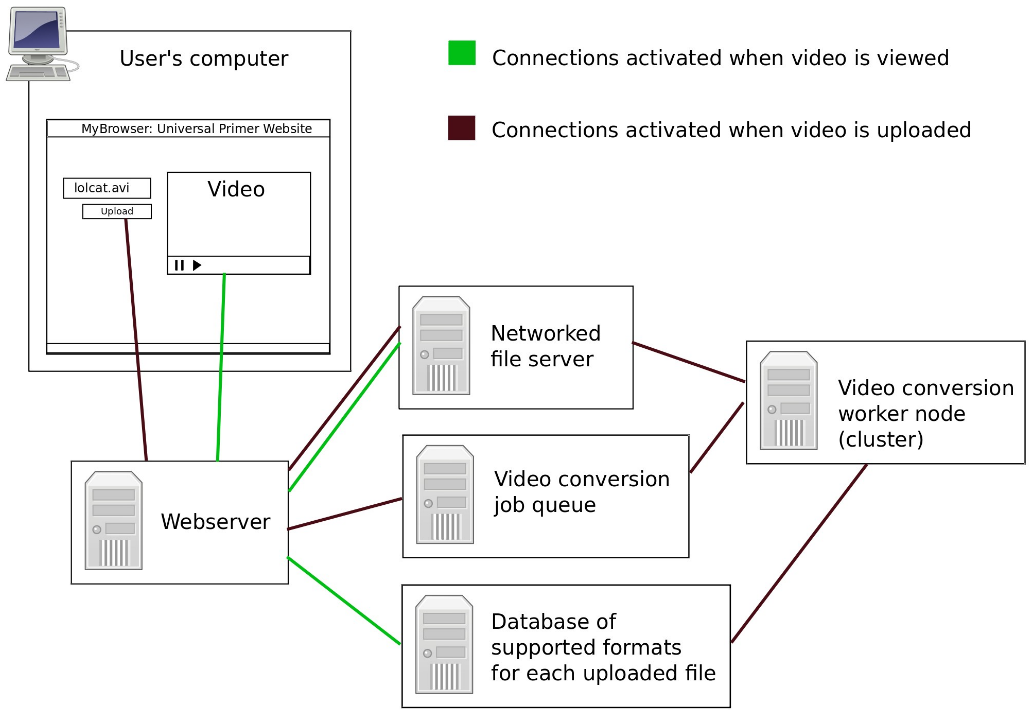

User's computer
MyBrowser: Universal Primer Website
lolcat.avi
Upload
Video
Connections activated when video is viewed
Connections activated when video is uploaded
Networked file server
Video conversion worker node (cluster)
Video conversion job queue
Webserver
Database of supported formats for each uploaded file

Illustration 6: System design for handling prerecorded video. Both use-cases for viewing video (green lines) and uploading video (dark lines) are shown.

Videos are uploaded via a standard web form, and are saved to a central networked file system. When a video is uploaded it must be converted to the list of supported formats. A database is kept updated with a list of all uploaded videos, and for each video, which formats the video is available in. When the video is uploaded, a job is added to a networked queue for each supported video format. A small collection of "workers", programs running on one or more servers, check the job queue at regular intervals. If a worker finds a new conversion job, the job is marked as belonging to that particular worker and the worker starts converting the video. When the worker is finished, the job is marked as completed and the list of available formats for that video gets updated to include the newly created file.

Users view videos by browsing a list of uploaded videos and clicking the video of interest. When a user selects a video to watch, a small script is started. This script firsts downloads a list of available formats for that particular video from the database. The script then instructs the browser to attempt to display the video in each of the available formats, beginning with the most open/free format. If the first format does not work in the browser, the next format is tried and so on until all the available formats have been tried. If none of the formats work, the user is given the option of instead downloading one of the files (a link to each of the available formats is shown). Hopefully the user will then be able to play the downloaded video in a standalone player, outside of the browser.

IMPLEMENTATION – WIKI-SITE PROJECT

A prototype of the dependency system was implemented as a web application using the Ruby on Rails framework and a MySQL database. The system implements the following features:

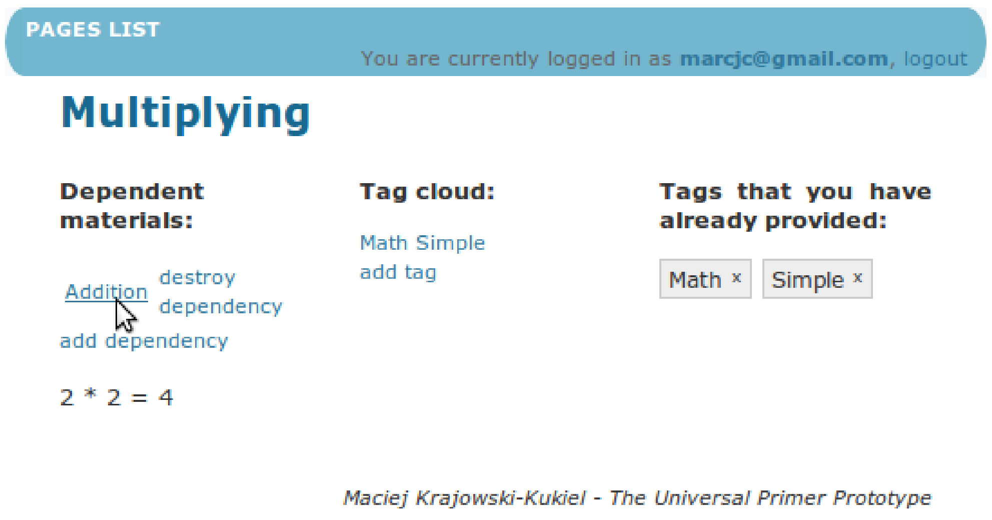
1. User sign-up and login/logout.
2. Creating, deleting and updating wiki-pages.
3. Adding tags to a page.
4. Adding dependencies to a page.
5. Tag-based searching.
6. Intelligent algorithm for suggesting new dependencies for a page, based on tags.

The most interesting feature in this system is the tagging-based dependency system. The other features may be seen as supporting the development of this feature rather than being of specific interest to the project and have not be documented in detail in this article. A detailed report on this prototype is available upon request [37].

Tags are added to pages by typing comma-separated text strings into a text-box. Dependencies can be added to pages by the users, and a recommendation algorithm is used to suggest pages that may be candidates for becoming dependencies of the current page.

The recommendation algorithm works as follows: Assume that the user has chosen a page called A and wishes to set dependencies for it. First the system collects all tags that belong to A. Then it compares them with the tags of all other pages known to the system. Referring to one of the pages being compared to A as B: The more matching tags are found between A and B, the more points B gets. In addition, the algorithm takes into consideration how popular the matching tags are on the site in general: The more pages that have the tag associated, the more points B gets. After analysis, all of the compared pages are sorted by points. The system displays up to ten pages sorted by highest number of points. If more than ten pages are found, then they are paginated.

An option to manually search for the page is also given, for cases where the recommendation algorithm fails to find the desired page.


PAGES LIST

You are currently logged in as **marcjc@gmail.com**, logout

Multiplying

Dependent materials:

Addition destroy dependency

add dependency

2 * 2 = 4

Tag cloud:

Math Simple

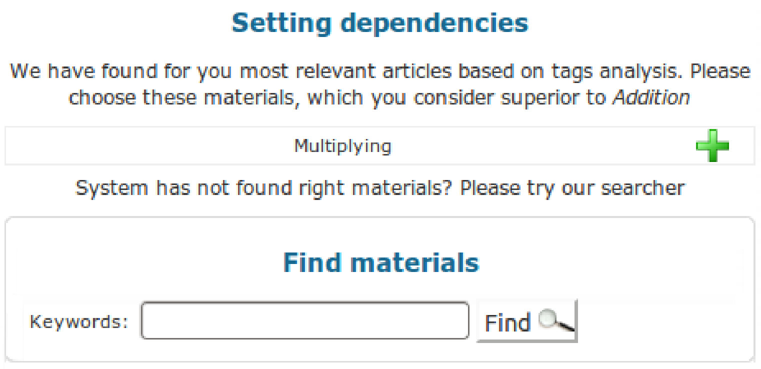
add tag

Tags that you have already provided:

Math × Simple ×

Maciej Krajowski-Kukiel - The Universal Primer Prototype

Illustration 7: Screenshot of a page named "Multiplying" with one dependency on the "Addition" page. Two tags, "Math" and "Simple" are associated with the page.


Setting dependencies

We have found for you most relevant articles based on tags analysis. Please choose these materials, which you consider superior to *Addition*

Multiplying

System has not found right materials? Please try our searcher

Find materials

Keywords: Find

Illustration 8: The dialog for setting dependencies. The system is recommending that the "Multiplying" article may be a relevant dependency for the current article.

Prerecorded Video

The implementation of the system for handling prerecorded video uses a combination of existing open source applications and a set of Perl scripts [38]. The web server is a standard Apache 2 [39] server, and Perl CGI [40] scripts are used for dynamic content.

CONCLUSION

A suite of software for live education has been developed with features suitable for a university classroom setting. A system for handling the online streaming of prerecorded video in a variety of formats was developed and tested. A system for collaborative organization of educational material using a combination of tagging and dependencies was designed and implemented. All software is covered by open source licenses. These systems have not yet been integrated into a unified whole. The software is useful as is, but requires some amount of technical expertise for setup. Better documentation, integration, packaging and real world testing are required before it can be considered for general public release.

FUTURE WORK

The work presented in this article, while functional, needs to be integrated into a single system. A serious testing, documentation and integration effort is needed before the Universal Primer is ready for wide public deployment. Major scalability improvements are planned: Moving to a highly scalable NoSQL database back-end has already been discussed, and would not represent a great challenge since the database structures used are relatively simple. A more pressing issue is large scale transcoding and hosting of video. A distributed solution for video storage must be found if the system is to scale properly. This could be either in the form of a distributed storage engine such as Ceph [41] or Tahoe-LAFS [42], or peer to peer video streaming software [6].

New modules will be written to supplement the video, slides and chat. Planned modules include a screen-casting module, a two-way video module to allow students to ask questions using video+voice and a peer-to-peer module for streaming video implemented as a Java applet. Minor planned improvements include the previously mentioned new GUI and a switch to SVG format instead of raster images for web-based display of the slides, as the vector-based SVG allows students to copy-paste text and adjust zoom.

Support is planned for mobile platforms such as Google Android and iOS, in order to support smartphones and tablet devices.

Ideas for more collaborative and flexible dependency-like organization solutions are being considered. A prominent idea is to allow users to specify a favoured “path” through the material on the website, that they can annotate with personal notes, and then use an algorithm to average over all user-specified paths when displaying what might be a dependency for the current subject, and how best to move forward learning new subjects.

An important addition to the project before deploying it in a public setting is increased security and filtering, specifically for live lectures, where the system is currently lacking a good access control implementation.

It is hoped that the list of supported codecs can be replaced by a single format: WebM video [43]. WebM uses the VP8 video format and Ogg Vorbis audio format. These codecs are both open, free and unencumbered by any enforced patents. It is hoped that this format will see wide cross-platform and cross-browser adoption in the near future.

When it becomes possible to switch to HTML5 video instead of Flash for live video support, the plan is to use the free WebM video format. Right now, HTML5 Video requires different codecs in different browsers, which means that more processing power has to be expended on transcoding. It is hoped that future browsers will all support the WebM format as a common standard.

ACKNOWLEDGEMENTS

This project was developed with support from:

The Technical University of Denmark Department of Informatics and Mathematical Modelling.

Interreg IV project “Vind i Øresund”.

Special thanks to:
  Carly L. Nacmanie

REFERENCES

[1] G. Stefansson; The Tutor-web; an educational system for classroom presentation, evaluation and self-study. Journal Computers & Education archive Volume 43 Issue 4, December 2004 Elsevier Science Ltd. Oxford, UK.

[2] http://www.gnu.org/philosophy/free-sw.html

[3] http://www.opensource.org/osd.html

[4] http://ocw.mit.edu/index.htm

[5] http://www.livescribe.com/en-us/

[6] http://torrentfreak.com/bittorrent-p2p-live-streaming-110119/

[7] http://code.macournoyer.com/thin/

[8] http://www.mysql.com/

[9] http://www.icecast.org/

[10] http://www.adobe.com/devnet/rtmp.html

[11] http://www.adobe.com/devnet/f4v.html

[12] http://www.itu.int/rec/T-REC-H.264

[13] http://mpeg.chiariglione.org/

[14] http://rack.rubyforge.org/

[15] https://github.com/macournoyer/pusher

[16] http://ww.telent.net/diary/Streaming_XHR_with_Ruby_and_Mongrel
http://cometdaily.com/2007/11/18/ie-activexhtmlfile-transport-part-ii/
http://www.w3.org/TR/eventsource/ http://cometdaily.com/2007/11/15/the-long-polling-technique/

[17] http://www.rabbitmq.com

[18] http://www.amqp.org

[19] http://www.mysql.com/products/cluster/

[20] http://rubyonrails.org/

[21] http://www.postgresql.org/

[22] http://www.graphicsmagick.org/

[23] http://flowplayer.org/

[24] http://raphaeljs.com/

[25] http://python.org

[26] http://qt.nokia.com/

[27] http://www.dealextreme.com/details.dx/sku.3071

[28] http://abstrakraft.org/cwiid/

[29] http://wiibrew.org/wiki/Wiimote#Bluetooth Communication

[30] http://johnnylee.net/projects/wii/

[31] http://dev.w3.org/html5/websockets/

[32] http://dvswitch.alioth.debian.org/wiki/DV_format/

[33] http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4659231

[34] http://www.kinodv.org/

[35] http://ffmpeg.org/

[36] http://www.mediawiki.org/wiki/MediaWiki

[37] Contact Marc Juul Christoffersen: mjchristoffersen@lbl.gov

[38] http://www.perl.org/

[39] http://httpd.apache.org/docs/2.0/

[40] http://www.ietf.org/rfc/rfc3875.txt

[41] http://ceph.newdream.net/

[42] http://tahoe-lafs.org/trac/tahoe-lafs

[43] http://www.webmproject.org/