Flexible Presenter Tools for e-Learning

Student: Christian Panton
Supervisor: Henrik Madsen

October 4th, 2010

Contents

All HTTP links retrieved October 4th, 2010.

1 Introduction and Context

The wealth of knowledge held by educational institutions is often kept secret or hidden from public view. Although most hide it intentionally, for institutions who wants to be open, the tools available to the teachers tends to keep knowledge bound to non-digital or propitiatory formats. Some open alternatives have developed, such as MIT OpenCourseWare, but all the components of a full open ecosystem is hard to come by[1]. The project tentatively named The Universal Primer Project (TUPP), seeks to create such an ecosystem, partly though tools for teaching and partly though tools for knowledge storage and dependency of knowledge.

This project aims to create one of parts needed in the tools for teaching. Software that can stream live video and broadcast slides of lectures for distance learners is not available as an open source ecosystem (clients and servers), and the teachers client seems to be one of the first places to start. Ideally this would all run in a internet browser, but even the most current revision of the HTML standard, would not enable us to interface with hardware such as pointing devices and cameras and provide the performance needed to stream live video. Although 3rd party plug-in technology might exist, such as Adobe Flash, these are propitiatory formats and requires the user to install software.

2 Development of the Presenter Tool - *emcee*

Due to the limited functionality of the current open standards based browser technologies, a stand-alone Python application was developed (See Appendix C for the source code). Its purpose is to support the teacher in presenting slides, setting up the live video and audio feeds and provide communication with the distance learners. The application is named after a Master of Ceremonies, the *Em-Cee*.

Python[2] is a dynamically typed, object orientated, interpreted programming language. Thus it 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 Graphical User Interface, *GUI*, the Qt4[3] library was chosen. Like Python it is licensed under the GNU General Public License and freely available and it is implemented on all major platforms, ensuring cross-platform

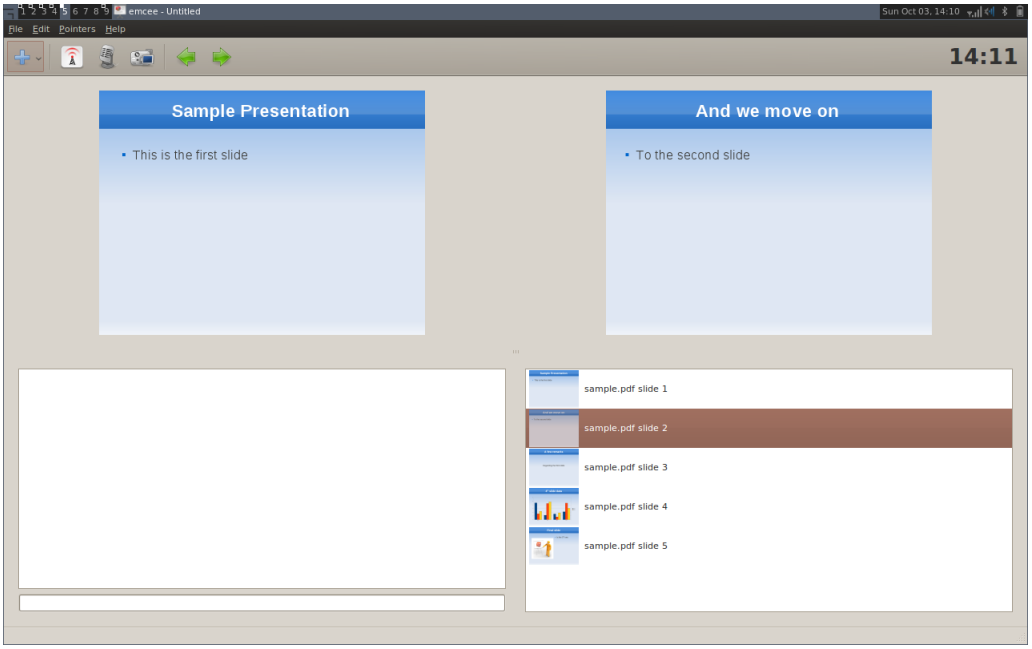
[1] http://primer.grafiki.org/index.php/Related_projects
[2] http://www.python.org/
[3] http://qt.nokia.com/

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[4] bindings. Another set of bindings, *PySide*, is currently being developed 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 API's, 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.

3 Functionality Overview

The main screen is depicted in figure 1. It shows the teacher 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 teacher to sort the slides while giving a presentation. By clicking on a slide this also enables the teacher to set the upcoming slide, ie. to skip a series of slides, go back in the presentation during an Q&A session, etc.



[4]http://wiki.python.org/moin/PyQt4

Figure 1: 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 teacher to write text based messages to the distance learners and makes it possible for the distance learners to ask questions to the teacher. 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 (going forward, etc.) and the types of media available on the slides, is 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. See Appendix B for the plug-in structure.

3.1 Pointing Devices

Pointing devices is a broad category of keyboards, mice and remote controls for controlling the flow of 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 no-name 2.4GHz wireless remote control[5] which enumerates as a USB HID keyboard on the host computer.

The second devices 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 figure 2.

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

Figure 2: The Wiimote (right) and the Nunchuck (left)

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 left abandoned. The CWIID[6] 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[7]. 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 it has the downside that 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.

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

[7]http://wiibrew.org/wiki/Wiimote#Bluetooth_Communication

The Wiimote is used for presentation flow control using the left and right arrow keys. The vibration motor is used to get attention from the teacher, such as getting 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 teacher 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[8]. 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 teacher.

3.2 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 is two types of plug-ins. Like with the pointer plug-ins above, a simple and a more advanced plug-in is avaliable.

The simple plug-in is a blank slide. Its MIME-type is the non-standard `application/x-blank-slide` and can be used to blank the projector between multiple presentation or in a Q&A session.

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

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` with a set of Python/Qt4 bindings called `pypoppler-qt4`.

We had to write a patch for the bindings, as they did not expose the Table of Contents of the PDF file, often used to provide a slide name. The slide name often set by software creating presentations, such as LaTeX-`beamer`.

3.3 Broadcasting Features

One of the main features of this application is its broadcasting features. They cover slide switching, distance learner chat/question and audio/video broadcast of the lecture.

The audio/video broadcast features has not yet been implemented, as the infrastructure was not available. The foundation for adding this is build into the application and it is implemented as a shell-command.

The slide control (forward, backward, annotation, etc.) and chat protocol named `CHIMP` has been defined (Appendix A) and implemented in the application.

For each lecture a unique stream-name is given. This acts as an identifier for the media streams, the web page created for the distance learner and in the CHIMP protocol.

The CHIMP protocol is based on the Ajax Push Engine[9] (APE) server. APE is made as a JavaScript framework for real-time push of data to websites. It has different methods of communicating, but they rely on standard HTTP requests. A custom Python APE client implementation was written to support this communication.

The APE client connects to a communication channel, in this case the stream-name. Once in the channel it can send and receive messages from and broadcast to all clients in the channel. The messages are plain strings and for the CHIMP protocol the strings are URL-escaped JSON[10]. The object created from the unescaped JSON is then used as a remote procedure call in the client. The procedure is determined by the `type` field. Any other parameters is depended on the procedure, see Appendix A for a listing of different procedures.

[9] http://www.ape-project.org/

[10] JavaScript Object Notation, a simple serialization of JavaScript objects, commonly used in web services

4 Concluding Summary

As no suiting presenting application was found for the e-Learning project. A custom application was developed.

The application has the basic functionality needed for synchronized slide-casting with chat and annotation features. Video and audio broadcast can be added in a few lines of code once the infrastructure is ready.

A wealth of different libraries has been used including Wiimote interaction, HTTP asynchronous connection pools (for APE), Qt4 graphic user interface and PDF rending.

A The CHIMP Protocol

The CHIMP Protocol is using APE chat messages as a transport layer. The CHIMP message is URL escaped JSON. Once unescaped, the message type is detected by the `type` string. From the type, it is possible to determine other fields in the message. The types and fields are described below.

A.1 CHIMP v1 DRAFT messages

`slides/change`
: `(identifier,index)` Changes the current slide to the slide content identified by `identifier` (if a file, a md5 checksum of the file) and zero-indexed page identified by `index`.

`chat/public-msg`
: `(nickname,msg)` Sends a chat message with content `msg` and author name `nickname`.

`draw/lines`
: `(lines)` Draws lines on the slide. `lines` is a double array, such that `[[[`$x_0^{line_0}$`,`$y_0^{line_0}$`],[`$x_1^{line_0}$`,`$y_1^{line_0}$`],...],[[`$x_0^{line_1}$`,`$y_0^{line_1}$`],[`$x_1^{line_1}$`,`$y_1^{line_1}$`],...]]`

`draw/lines-clear`
: Clears any lines drawn on the slide

B Plug-in Structure

The plug-in is a Python module. In order to provide a distinction between different plug-in types the string `plugintype` must be set. Currently supported types are `pointer` and `content`.

B.1 `pointer`

In addition to `plugintype` a few other variables, classes and methods must be defined.

`name`
: A short descriptive name.

`description`
: A longer description of the plug-in.

`capabilities`
: An array of capabilities of the plug-in. The currently supported are: `['control','draw','attention','battery']`. Control is flow control, draw is drawing on slides, attention is the capability of notifying the teacher of some event through the device and battery is that the battery can be monitored though an interface provided by the plug-in.

`enabledefault`
: A boolean describing if the plug-in should be loaded at application start.

`pointer`
: A class holding the current pointer.

`pointer.enable()`
: Enable the pointer.

`pointer.disable()`
: Disable the pointer.

`pointer.battery()`
: (Capabilities dependent) Returns the percentage of the battery charge.

`pointer.xy()`
: (Capabilities dependent) Returns a tuple of normalized coordinates (x,y) or None if the pointer is not pointing on the screen.

`pointer.attention()`
: (Capabilities dependent) Try to get some attention.

B.2 `content`

In addition to `plugintype` a few other variables and methods must be defined.

`mimetype`
: A string of the MIME-type supported by this plug-in.

`name`
: A short descriptive name.

`source`
: The source type, currently supported: `file` or `internal`. If the type is file, a file dialog will be provided by the system for choosing the file.

`filetype`
: (Source dependent) If the source is a file, then this describes the extention of the file, needed for the file dialog.

`widget(container,index=0)`
: Returns a QWidget of the slide in container with index.

`icon(container,index=0)`
: Returns a QIcon of the slide in container with index.

`container`
: A class holding the source for the slides.

`container(reference)`
: Constructor for the container class. Reference is the file path for file-sources.

`container.numSlides()`
: Returns the number of slides in the container

`container.getName(index)`
: Returns a descriptive name for the slides at index, ie. from file metadata.

`container.getIdentifier()`
: Returns a unique identifier for this container. If the container contains the same content, the identifier should be the same across instances. If the source is a file, use a md5 checksum as identifier.

C Source code

Source code is listed below and available at:
`http://github.com/UniversalPrimer/emcee-gui-client`.

The revision for this report is `954beaf775bd3349c1fd`.

C.1 README

../emcee–gui–client/README

```
1 # Todo
2 - Server interaction features
```

```
- Setup broadcast of video and audio
- Evaluate Douglas-Peucker vs Vertex reduction

# What is working

Content plugins:
- PDF
- Blank slides

Pointer plugins:
- Standard keyboard and remotes
- Bluetooth wiimote

# Installation

Packages needed:

python-qt4
python-qt4-phonon
python-qt4-dev
python-cwiid
libpoppler-qt4-dev
python-sip4-dev
sip4


Get pypoppler from:
http://pyqt4-extrawidgets.googlecode.com/files/
   pypoppler-qt4-20962-fixed.tar.gz

copy pypoppler-qt4.patch to the directory where you
   extracted the source

Install using:
patch -R -p1 -i pypoppler-qt4.patch
python configure.py
make
sudo make install
```

C.2 emcee

../emcee–gui–client/emcee

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# core library
import sys

# addon libraries
from PyQt4.QtCore import *
from PyQt4.QtGui import *

# local imports
import controller

__version__ = "0a"

app = QApplication(sys.argv)
app.setApplicationVersion(__version__)
app.setOrganizationName("tupp")
app.setApplicationName("emcee")
control = controller.Controller()
control.start()

sys.exit(app.exec_())
# if it segfaults, blame the bindings, fixed in
   4.7.4 i hope
```

C.3 ape.py

../emcee–gui–client/ape.py

```
import asyncore, asynchat
import socket
import json
import string
import random
import threading
import time
import select
import urllib

class APEConnection(asynchat.async_chat):
```

```
12
13      def __init__(self, apeclient):
14          asynchat.async_chat.__init__(self)
15
16          self.apeclient = apeclient
17          self.callback_func = apeclient.callback
18          self.set_terminator("\n\n")
19          self.data = ""
20          self.gotheaders = False
21          self.age = time.time()
22          self.timeout = 15
23
24          if self.apeclient.state == 0:
25              cmd = 'connect'
26              params = {'name': self.apeclient.name}
27
28          elif self.apeclient.state == 1:
29              cmd = 'join'
30              params =  {'channels': self.apeclient.
                    channel}
31
32          elif self.apeclient.state == 2:
33              if len(self.apeclient.msgqueue) > 0:
34                  msg = self.apeclient.msgqueue.pop(0)
35                  cmd = 'send'
36                  params =  {"msg": self.escape(msg),
                        "pipe": self.apeclient.pipeid}
37              else:
38                  cmd = 'check'
39                  params =  {}
40
41          self.create_socket(socket.AF_INET, socket.
                SOCK_STREAM)
42          self.connect((self.apeclient.host, self.
                apeclient.port))
43
44          data = [{'cmd': cmd.upper(),'chl': self.
                apeclient.challenge, 'params': params}]
45
46          if self.apeclient.sessid:
```

```
47              data[0]['sessid'] = self.apeclient.
                    sessid
48
49          data_json = json.dumps(data, False, True,
                True, True, json.JSONEncoder, None, (',',
                ':'))
50
51          s  = "POST /%d/? HTTP/1.1\r\n" % self.
                apeclient.protocol
52          s += "Host: %s:%d\r\n" % (self.apeclient.
                host,self.apeclient.port)
53          s += "Accept-Encoding: identity\r\n"
54          s += "Content-type: application/x-www-form-
                urlencoded; charset=utf-8\r\n"
55          s += "Keep-alive: 60\r\n"
56          s += "Connection: Keep-alive\r\n"
57          s += "Content-Length: %d\r\n" % len(
                data_json)
58          s += "\r\n"
59          s += data_json
60
61          self.push(s)
62          self.apeclient.challenge += 1
63
64          if len(self.apeclient.msgqueue) > 0:
65              self.close()
66
67      def handle_expt(self):
68          self.close()
69
70      def collect_incoming_data(self, data):
71          self.data = self.data + data
72
73      def found_terminator(self):
74
75          data = self.data
76          if data.endswith("\r"):
77              data = data[:-1]
78          self.data = ""
79
80          if not self.gotheaders:
```

```
81              (headers, data) = data.split("\r\n\r\n"
                    ,1)
82              self.gotheaders = True
83
84          reply = json.loads(data.strip())
85          self.handle_reply(reply)
86
87
88      def handle_reply(self,reply):
89          self.age = time.time()
90          for item in reply:
91              key = item["raw"]
92              value = item["data"]
93              if key == u"LOGIN":
94                  self.apeclient.sessid = value["
                        sessid"]
95                  self.apeclient.state = 1
96                  self.close()
97              elif key == u"CHANNEL":
98                  self.apeclient.pipeid = value["pipe"
                        ]["pubid"]
99                  self.apeclient.state = 2
100             elif key == u"CLOSE":
101                 self.close()
102             elif key == u"ERR":
103                 self.apeclient.terminate = True
104                 print value
105             elif key == u"DATA":
106                 self.callback(value["msg"])
107             elif key == u"IDENT":
108                 pass
109             elif key == u"LEFT":
110                 pass
111             elif key == u"JOIN":
112                 pass
113             else:
114                 print "Unhandled: " + key
115                 print value
116
117     def readable(self):
118         if time.time() - self.age > self.timeout:
```

```
119             self.close()
120         return 1
121 
122     def callback(self,msg):
123         if self.callback_func:
124             self.callback_func(self.apeclient,self.
                   descape(msg))
125 
126     def escape(self,msg):
127         return urllib.quote(msg)
128 
129     def descape(self,msg):
130         return urllib.unquote(msg)
131 
132 class APEClient:
133 
134     def __init__(self, host, port, channel,
           frequency=0, callback=None):
135         self.frequency = frequency
136         self.channel = channel
137         self.host = str(frequency) + "."  + host
138         self.port = port
139         self.name = ''.join(random.sample(list(
               string.letters + string.digits),16))
140         self.challenge = 1
141         self.state = 0
142         self.protocol = 1 # XHR
143         self.sessid = None
144         self.pipeid = None
145         self.callback = callback
146         self.terminate = False
147         self.connection = None
148         self.thread = threading.Thread(target=self.
               run)
149         self.msgqueue = []
150 
151     def run(self):
152         while not self.terminate:
153             try:
154                 self.connection = APEConnection(self
                       )
```

```
155                 asyncore.loop(1)
156             except select.error:
157                 pass
158 
159 
160     def connect(self):
161         self.thread.start()
162 
163     def close(self):
164         self.terminate = True
165         self.thread.join(0)
166         self.connection.close()
167 
168     def send(self,msg):
169         self.msgqueue.append(msg)
170         if self.connection:
171             self.connection.close()
```

C.4 controller.py

../emcee–gui–client/controller.py

```
1  from PyQt4.QtCore import *
2  from PyQt4.QtGui import *
3  
4  import gui
5  import os
6  import presentation
7  import plugins
8  import util
9  import ape
10 import json
11 
12 class Controller(QObject):
13 
14     def __init__(self):
15         QObject.__init__(self)
16         self.loadDefaultSettings()
17         self.screens = ScreenController()
18         self.mainview = gui.MainWindow(self)
19         self.beamview = gui.BeamWindow(self)
20         self.prefview = gui.PreferencesWindow(self)
```

```
21          self.bcconnection = None
22          self.presentation = None
23          self.currentslide = None
24
25          # setup the main window
26          self.mainview.setCentralWidget(gui.
               LoadPresentationWidget(self))
27
28          # enable the default pointers
29          self.pointers = {}
30          self.drawingpointers = []
31
32          for pointerplugin in plugins.pointer:
33              pointer = pointerplugin.pointer(self)
34              self.pointers[pointerplugin] = pointer
35              if pointerplugin.enabledefault:
36                  self.enablePointer(pointerplugin)
37
38          self.drawing = DrawingController(self.
               drawingpointers,self.beamview,self)
39
40
41      # application
42      def start(self):
43          if self.screens.countScreens() > 1:
44              self.screens.moveToScreen(1,self.
                   beamview)
45              self.showFullScreen()
46          self.beamview.show()
47          self.mainview.show()
48
49      def quitApplication(self):
50          self.beamview.close()
51          self.mainview.close()
52          for (pointermodule, pointer) in self.
               pointers.items():
53              pointer.disable()
54
55      def mainClosed(self):
56          question = QMessageBox.question(self.
               mainview, self.tr("Application Close"),
```

```
            self.tr("Are you sure to quit?"),
            QMessageBox.No, QMessageBox.Yes)
57          if question == QMessageBox.Yes:
58              self.beamview.close()
59              return True
60          else:
61              return False
62
63      def aboutApp(self):
64          QMessageBox.about(self.mainview, self.tr("
                emcee"),self.tr("emcee\n\nQt version: %s\
                nPyQt version: %s\nApplication version: %
                s" % (QT_VERSION_STR, PYQT_VERSION_STR,
                QApplication.applicationVersion())))
65
66      def setStatus(self,message):
67          self.mainview.status.showMessage(message)
68
69      def loadDefaultSettings(self):
70          settings = QSettings()
71          defaults = {"pointercolor": QColor(255, 0,
                0),
72                      "pointersize": 20,
73                      "linecolor": QColor(0,0,0),
74                      "linewidth": 10,
75                      }
76          for (k,v) in defaults.items():
77              settings.setValue(k,settings.value(k,v))
78
79      # presentations
80      def newPresentation(self):
81          self.mainview.setCentralWidget(gui.
                NewPresentationWidget(self))
82
83      def closePresentation(self):
84          if self.presentation and self.presentation.
                saveneeded:
85              print "Todo: Ask for save"
86          self.mainview.showToolbar(False)
87          self.mainview.setCentralWidget(gui.
                LoadPresentationWidget(self))
```

```
88
89      def openPresentation(self):
90          filename = QFileDialog.getOpenFileName(self.
               mainview, self.tr("Open Presentation"),
               os.getcwd(), self.tr("emcee Presentation
               (*.paj)"))
91          if filename:
92              pres = presentation.Presentation(str(
                   filename))
93              self.setPresentation(pres)
94              self.presentation.emit(SIGNAL("changed()
                   "))
95
96      def openRemotePresentation(self):
97          print "NYI: Remote Open"
98
99      def savePresentation(self):
100         if self.presentation:
101             if self.presentation.filename:
102                 self.presentation.save()
103             else:
104                 self.saveAsPresentation()
105
106     def saveAsPresentation(self):
107         if self.presentation:
108             filename = QFileDialog.getSaveFileName(
                   self.mainview, self.tr("Save
                   Presentation"), os.getcwd() + "/" +
                   self.presentation.suggestedFileName()
                    + ".paj", self.tr("emcee
                   Presentation (*.paj)"))
109             if filename:
110                 self.presentation.save(str(filename)
                       )
111
112     def setPresentation(self,presentation):
113         self.presentation = presentation
114         self.currentslide = presentation.
               defaultSlide
115         self.nextslide = presentation.defaultSlide
116         self.mainview.setTitle(presentation.title)
```

```
117         self.mainview.setCentralWidget(gui.
               PresentationWidget(self))
118         self.mainview.showToolbar(True)
119         self.emit(SIGNAL("updateSlides()"))
120 
121     def createPresentation(self,title,name,email,
           forclass,organization):
122         p = presentation.Presentation()
123         p.title = str(title) if len(title) else str(
               p.title)
124         p.name = str(name)
125         p.email = str(email)
126         p.forclass = str(forclass)
127         p.organization = str(organization)
128         self.setPresentation(p)
129 
130     # slide control
131     def addSlide(self, mimetype):
132         plugin = plugins.mimehandlers[mimetype]
133         if plugin.source == "file":
134             filename = QFileDialog.getOpenFileName(
                   None, self.tr("Select file"), os.
                   getcwd(), "%s (*.%s)" % (plugin.name,
                    plugin.filetype))
135             if filename:
136                 source = presentation.Source(str(
                       filename),mimetype)
137                 self.presentation.addSource(source)
138         elif plugin.source == "internal":
139              source = presentation.Source(None,
                    mimetype)
140              self.presentation.addSource(source)
141 
142     def nextSlide(self):
143         if self.presentation:
144             self.setCurrentSlide(self.presentation.
                   nextindex)
145             self.setNextSlide(self.presentation.
                   nextindex+1)
146             self.emit(SIGNAL("setNextSlide(int)"),
                   self.presentation.nextindex)
```

```
147
148     def previousSlide(self):
149         if self.presentation:
150             self.setNextSlide(self.presentation.
                    currentindex)
151             self.setCurrentSlide(self.presentation.
                    currentindex-1)
152             self.emit(SIGNAL("setNextSlide(int)"),
                    self.presentation.nextindex)
153
154     def setCurrentSlide(self, index):
155         if index >= 0 and index < len(self.
                presentation.slides):
156             self.presentation.currentindex = index
157             self.currentslide = self.presentation.
                    getSlide(index)
158         else:
159             self.currentslide = self.presentation.
                    defaultSlide
160         self.emit(SIGNAL("updateSlides()"))
161         self.drawing.drawClear()
162
163         if self.bcconnection:
164             jvar = json.dumps({"type": "slides/
                    change", "identifier": self.
                    currentslide.source.identifier(), "
                    index":  self.presentation.
                    currentindex})
165             self.bcconnection.send(jvar)
166
167     def setNextSlide(self, index):
168         if index >= 0 and index < len(self.
                presentation.slides):
169             self.presentation.nextindex = index
170             self.nextslide = self.presentation.
                    getSlide(index)
171         else:
172             self.nextslide = self.presentation.
                    defaultSlide
173         self.emit(SIGNAL("updateSlides()"))
174
```

```
175     # chat
176     def chatSend(self,chat):
177         if len(self.presentation.name):
178             who = self.presentation.name
179         else:
180             who = "Teacher"
181 
182         if self.bcconnection:
183             self.bcconnection.send(json.dumps({"type
                    ": "chat/public-msg", "msg": unicode(
                    chat), "nickname": unicode(who)}))
184             self.emit(SIGNAL("chatRecieved(QString)"
                    ),"<b>%s</b>: %s" % (who,chat))
185         else:
186             self.emit(SIGNAL("chatRecieved(QString)"
                    ),"<i>Not connected</i>" )
187 
188     # ape
189     def apeRecieved(self,obj,var):
190         jvar = json.loads(var)
191         if jvar["type"] == "chat/public-msg":
192              self.emit(SIGNAL("chatRecieved(QString)
                     "),"<b>%s</b>: %s" % (jvar["nickname
                     "],jvar["msg"]))
193 
194 
195     # broadcast
196     def startBroadcast(self):
197        settings = QSettings()
198        self.bcconnection = ape.APEClient(str(
              settings.value("server").toString()),
              6969, self.presentation.suggestedFileName
              (), callback=self.apeRecieved)
199        self.bcconnection.connect()
200 
201     def endBroadcast(self):
202         if self.bcconnection:
203             self.bcconnection.close()
204             self.bcconnection = None
205 
206     # pointers
```

```
207     def enablePointer(self,pointerplugin):
208         if self.pointers[pointerplugin].enable() and
                pointerplugin.capabilities.count("draw")
            :
209             self.drawingpointers.append(self.
                   pointers[pointerplugin])
210
211     def disablePointer(self,pointer):
212         self.pointers[pointer].disable()
213
214
215 class ScreenController():
216
217     def __init__(self):
218         self.desktop = QDesktopWidget()
219         self.geometry = []
220         primary = self.desktop.screenGeometry(self.
               desktop.primaryScreen()).topLeft()
221
222         for i in range(0,self.desktop.numScreens()):
223             self.geometry.append(self.desktop.
                   screenGeometry(i).topLeft()-primary)
224
225     def moveToScreen(self,screen,widget):
226         widget.move(self.geometry[screen])
227
228     def countScreens(self):
229         return self.desktop.numScreens()
230
231 class DrawingController(QThread):
232
233     def __init__(self,pointers,beamview,controller):
234         QThread.__init__(self)
235         self.pointers = pointers
236         self.beamview = beamview
237         self.controller = controller
238         self.drawing = False
239         self.points = []
240         self.current = -1
241         self.start()
242
```

```
243     def drawBegin(self):
244         self.current += 1
245         self.points.append([])
246         self.points[self.current] = []
247         self.drawing = True
248 
249     def drawEnd(self):
250         if self.drawing:
251             self.drawing = False
252             if self.controller.bcconnection:
253                 self.controller.bcconnection.send(
                       json.dumps({"type": "draw/lines",
                        "lines": self.points}))
254 
255     def drawClear(self):
256         self.points = []
257         self.current = -1
258         self.beamview.update()
259         if self.controller.bcconnection:
260             self.controller.bcconnection.send(json.
                   dumps({"type": "draw/clear"}))
261 
262     def run(self):
263         while True:
264             if len(self.pointers):
265                 pointerset = False
266                 for pointer in self.pointers:
267                     xy = pointer.xy()
268                     if xy:
269                         if self.drawing:
270                             self.points[self.current
                                ].append((xy[0],xy
                                [1]))
271 
272                         self.beamview.setPointer(xy
                            [0],xy[1])
273 
274                         pointerset = True
275                         break
276 
277                 if not pointerset:
```

```
278                     self.beamview.clearPointer()
279
280                 self.beamview.setPaths(self.points)
281
282
283                 self.msleep(33)
284             else:
285                 self.sleep(10)
```

C.5 gui.py

../emcee–gui–client/gui.py

```
1  from PyQt4.QtCore import *
2  from PyQt4.QtGui import *
3
4  import plugins
5  import time
6  import util
7  import webbrowser
8
9  ####################################################
10 # The Main Window
11 ####################################################
12
13 class MainWindow(QMainWindow):
14
15     def __init__(self, controller):
16         QMainWindow.__init__(self)
17         self.controller = controller
18
19         self.setWindowIcon(QIcon("icons/presentation
               .svg"))
20
21         # setup the window
22         self.createStatus()
23         self.createMenus()
24         self.createToolbar()
25
26         # start centered and set minimum size
27         screen = QDesktopWidget().screenGeometry()
28         size =  self.geometry()
```

```
29          self.move((screen.width()-size.width())/2, (
               screen.height()-size.height())/2)
30          self.setMinimumSize(QSize(640,480))
31
32      # menubars, status and toolbars
33      def createStatus(self):
34          self.status = self.statusBar()
35          self.status.setSizeGripEnabled(True)
36          self.status.showMessage("Ready", 5000)
37
38      def createMenus(self):
39
40          self.menu = self.menuBar()
41          menufile = self.menu.addMenu(self.tr("&File"
               ))
42          menuedit = self.menu.addMenu(self.tr("&Edit"
               ))
43          menupointers = self.menu.addMenu(self.tr("&
               Pointers"))
44          menuhelp = self.menu.addMenu(self.tr("&Help"
               ))
45
46          filenew = self.createAction(self.tr("&New"),
                self.controller.newPresentation,
               QKeySequence.New, None, self.tr("New
               presentation"))
47          fileopen = self.createAction(self.tr("&Open
               ..."), self.controller.openPresentation,
               QKeySequence.Open, None, self.tr("Open
               presentation"))
48          fileopenremote = self.createAction(self.tr("
               Open &Remote..."), self.controller.
               openRemotePresentation, "Ctrl+R", None,
               self.tr("Open presentation from a server"
               ))
49          filesave = self.createAction(self.tr("&Save"
               ), self.controller.savePresentation,
               QKeySequence.Save, None, self.tr("Save
               the presentation"))
50          filesaveas = self.createAction(self.tr("Save
                &As..."), self.controller.
```

```
            saveAsPresentation, QKeySequence.SaveAs,
            None, self.tr("Save the presentation to a
             new file"))
51          fileclose = self.createAction(self.tr("&
            Close"), self.controller.
            closePresentation, QKeySequence.Close,
            None, self.tr("Close the presentation"))
52          filequit = self.createAction(self.tr("&Quit"
            ), self.controller.quitApplication, "Ctrl
            +Q", None, self.tr("Close the program"))
53
54          editpreferences = self.createAction(self.tr(
            "&Preferences..."), lambda: self.
            controller.prefview.show())
55
56          pointers = []
57          for pointermodule in plugins.pointer:
58              act = self.createAction(pointermodule.
                name, None, None, None, pointermodule
                .description, True)
59              if pointermodule.enabledefault:
60                  act.setChecked(True)
61              self.connect(act, SIGNAL("triggered()"),
                  lambda p=pointermodule: self.
                controller.enablePointer(p) if act.
                isChecked() else self.controller.
                disablePointer(p))
62              pointers.append(act)
63
64          helpwww = self.createAction(self.tr("&
            Website..."), lambda: webbrowser.open_new
            ("http://github.com/UniversalPrimer"))
65          helpabout = self.createAction(self.tr("&
            About..."), self.controller.aboutApp)
66
67          self.addActions(menufile,(filenew,fileopen,
            fileopenremote,None,filesave,filesaveas,
            None,fileclose,filequit))
68          self.addActions(menuedit,(editpreferences,))
69          self.addActions(menupointers,pointers)
```

```
70          self.addActions(menuhelp,(helpwww,None,
                helpabout))
71
72      def createToolbar(self):
73          self.toolbar = QToolBar()
74          self.toolbar.setIconSize(QSize(32,32))
75          self.toolbar.setVisible(False)
76          self.toolbar.setMovable(False)
77
78          addsource = QToolButton()
79          addsource.setIcon(QIcon("icons/list-add.svg"
                ))
80          addsource.setPopupMode(QToolButton.
                InstantPopup)
81
82          addsourcemenu = QMenu(self)
83
84          for mimetype in plugins.mimehandlers.
                iterkeys():
85              plugin = plugins.mimehandlers[mimetype]
86              action = QAction(self)
87              action.setText(plugin.name)
88              self.connect(action,SIGNAL("triggered()"
                    ),lambda x=mimetype: self.controller.
                    addSlide(x))
89              addsourcemenu.addAction(action)
90
91          addsource.setMenu(addsourcemenu)
92
93          forwardbtn = QToolButton()
94          forwardbtn.setIcon(QIcon("icons/go-next.svg"
                ))
95          self.connect(forwardbtn,SIGNAL("clicked()"),
                self.controller.nextSlide)
96
97          backbtn = QToolButton()
98          backbtn.setIcon(QIcon("icons/go-previous.svg
                "))
99          self.connect(backbtn,SIGNAL("clicked()"),
                self.controller.previousSlide)
100
```

```
101         bcast = QToolButton()
102         bcast.setIcon(QIcon("icons/broadcast.svg"))
103         bcast.setCheckable(True)
104         self.connect(bcast,SIGNAL("clicked()"),
               lambda: self.controller.startBroadcast()
               if bcast.isChecked() else self.controller
               .endBroadcast())
105
106         mic = QToolButton()
107         mic.setIcon(QIcon("icons/microphone.svg"))
108         mic.setCheckable(True)
109
110         cam = QToolButton()
111         cam.setIcon(QIcon("icons/camera-video.svg"))
112         cam.setCheckable(True)
113
114         labeltimer = QLabel()
115         labeltimer.setFont(QFont('Sans', 20, QFont.
               Bold))
116         self.timer = QTimer()
117         self.connect(self.timer,SIGNAL("timeout()"),
               lambda x=labeltimer: x.setText(time.
               strftime("%H:%M ")))
118         self.timer.start(1000)
119
120         stretch = QWidget()
121         stretch.setSizePolicy(QSizePolicy(
               QSizePolicy.Expanding,QSizePolicy.
               Expanding))
122
123         self.toolbar.addWidget(addsource)
124         self.toolbar.addSeparator()
125         self.toolbar.addWidget(bcast)
126         self.toolbar.addWidget(mic)
127         self.toolbar.addWidget(cam)
128         self.toolbar.addSeparator()
129         self.toolbar.addWidget(backbtn)
130         self.toolbar.addWidget(forwardbtn)
131
132         self.toolbar.addWidget(stretch)
133         self.toolbar.addWidget(labeltimer)
```

```
134         self.addToolBar(self.toolbar)
135
136     # helpers
137     def createAction(self, text, slot=None, shortcut
            =None, icon=None, tip=None, checkable=False,
            signal="triggered()"):
138         action = QAction(self)
139         action.setText(text) #pyside bug
140         if icon is not None:
141             action.setIcon(QIcon(icon))
142         if shortcut is not None:
143             action.setShortcut(shortcut)
144         if tip is not None:
145             action.setToolTip(tip)
146             action.setStatusTip(tip)
147         if slot is not None:
148             self.connect(action, SIGNAL(signal),
                    slot)
149         if checkable:
150             action.setCheckable(True)
151         return action
152
153     def addActions(self, target, actions):
154         for action in actions:
155             if action is None:
156                 target.addSeparator()
157             else:
158                 target.addAction(action)
159
160     def setTitle(self, name):
161         self.setWindowTitle(QApplication.
                applicationName() + " - " + name)
162
163     def showToolbar(self, state=True):
164         self.toolbar.setVisible(state)
165
166     def closeEvent(self, event):
167         if self.controller.mainClosed():
168             event.accept()
169         else:
170             event.ignore()
```

```
171
172 ######################################################
173 # Window: Beam View, the presentation for a
      projector
174 ######################################################
175
176 class BeamWindow(QWidget):
177
178     def __init__(self, controller):
179         QWidget.__init__(self)
180         self.controller = controller
181         self.overlay = Overlay()
182         self.slide = SlideWidget(self.overlay)
183         layout = QGridLayout()
184         layout.setMargin(0)
185         self.slide.layout.setMargin(0)
186         self.setMinimumSize(QSize(400,300))
187         layout.addWidget(self.slide,0,0)
188         self.setLayout(layout)
189         self.connect(self.controller,SIGNAL("
              updateSlides()"),self.updateSlideView)
190         self.testcard = True
191         self.setWindowTitle(self.tr("Projector
              Display"))
192
193     def setPointer(self,x,y):
194         size = self.overlay.size()
195         x = int(size.width() * (1-x))
196         y = int(size.height() * (1-y))
197         self.overlay.x = x
198         self.overlay.y = y
199         self.overlay.update()
200
201     def clearPointer(self):
202         if not (self.overlay.x == 0 and self.overlay
              .y == 0):
203             self.overlay.x = 0
204             self.overlay.y = 0
205             self.overlay.update()
206
207     def setPaths(self,paths):
```

```
208         size = self.overlay.size()
209         newpaths = []
210         for path in paths:
211             if len(path):
212                 newpath = []
213                 path = util.vertexreduce(path
                        ,1/100.)
214                 for point in path:
215                     x = int(size.width() * (1-point
                            [0]))
216                     y = int(size.height() * (1-point
                            [1]))
217                     newpath.append(QPoint(x,y))
218                 newpaths.append(newpath)
219         self.overlay.paths = newpaths
220 
221     def updateSlideView(self):
222         self.slide.setSlide(self.controller.
                currentslide)
223         self.testcard = False
224 
225     def paintEvent(self,arg):
226         painter = QPainter(self)
227 
228         if self.testcard :
229             w = self.size().width()/8
230             h = self.size().height()/2
231             painter.fillRect(0*w, 0, w, h, QColor("
                    white"))
232             painter.fillRect(1*w, 0, w, h, QColor("
                    yellow"))
233             painter.fillRect(2*w, 0, w, h, QColor("
                    cyan"))
234             painter.fillRect(3*w, 0, w, h, QColor("
                    lime"))
235             painter.fillRect(4*w, 0, w, h, QColor("
                    magenta"))
236             painter.fillRect(5*w, 0, w, h, QColor("
                    red"))
237             painter.fillRect(6*w, 0, w, h, QColor("
                    blue"))
```

```
238             painter.fillRect(7*w, 0, w, h, QColor("
                    black"))
239             painter.fillRect(7*w, h, w, 2*h, QColor(
                    "white"))
240             painter.fillRect(6*w, h, w, 2*h, QColor(
                    "yellow"))
241             painter.fillRect(5*w, h, w, 2*h, QColor(
                    "cyan"))
242             painter.fillRect(4*w, h, w, 2*h, QColor(
                    "lime"))
243             painter.fillRect(3*w, h, w, 2*h, QColor(
                    "magenta"))
244             painter.fillRect(2*w, h, w, 2*h, QColor(
                    "red"))
245             painter.fillRect(1*w, h, w, 2*h, QColor(
                    "blue"))
246             painter.fillRect(0*w, h, w, 2*h, QColor(
                    "black"))
247             firstpen = QPen(QColor("grey"), 40, Qt.
                    DotLine, Qt.SquareCap, Qt.BevelJoin)
248             painter.setPen(firstpen)
249             painter.drawRect(0,0,w*8,h*2)
250         else:
251             painter.fillRect(0, 0, self.size().width
                    (), self.size().height(), QColor("
                    black"))
252
253 ##################################################
254 # Window: Preferences
255 ##################################################
256
257 class PreferencesWindow(QDialog):
258
259     def __init__(self, controller):
260         QDialog.__init__(self)
261         self.setModal(True)
262
263
264         settings = QSettings()
265         self.serverfield = QLineEdit()
```

```
266         self.serverfield.setText(settings.value("
               server").toString())
267
268         self.pointercolor = ColorButton()
269         self.pointercolor.setColor(QColor(settings.
               value("pointercolor")))
270
271         self.linecolor = ColorButton()
272         self.linecolor.setColor(QColor(settings.
               value("linecolor")))
273
274         self.pointersize = QSpinBox()
275         self.pointersize.setValue(settings.value("
               pointersize").toInt()[0])
276
277         self.linewidth = QSpinBox()
278         self.linewidth.setValue(settings.value("
               linewidth").toInt()[0])
279
280         cancelbtn = QPushButton(self.tr("C&ancel"),
               self)
281         savebtn = QPushButton(self.tr("C&reate"),
               self)
282         savebtn.setDefault(True)
283
284         formLayout = QFormLayout()
285         formLayout.addRow(self.tr("&Server:"), self.
               serverfield)
286         formLayout.addRow(self.tr("Pointer &color:")
               , self.pointercolor)
287         formLayout.addRow(self.tr("Pointer s&ize:"),
                self.pointersize)
288         formLayout.addRow(self.tr("Line c&olor:"),
               self.linecolor)
289         formLayout.addRow(self.tr("Line &width:"),
               self.linewidth)
290
291         buttonLayout = QHBoxLayout()
292         buttonLayout.addStretch(1)
293         buttonLayout.addWidget(cancelbtn)
294         buttonLayout.addWidget(savebtn)
```

```
295
296         vbox = QVBoxLayout()
297         vbox.addWidget(QLabel("<b>" + self.tr("
               Preferences") +  "</b>"))
298         vbox.addLayout(formLayout)
299         vbox.addStretch(1)
300         vbox.addLayout(buttonLayout)
301
302         self.setLayout(vbox)
303         self.connect(savebtn, SIGNAL("clicked()"),
               self.save)
304         self.connect(cancelbtn, SIGNAL("clicked()"),
                self.close)
305
306         self.resize(400,300)
307         screen = QDesktopWidget().screenGeometry()
308         size =  self.geometry()
309         self.move((screen.width()-size.width())/2, (
               screen.height()-size.height())/2)
310
311
312     def save(self):
313         settings = QSettings()
314         settings.setValue("server",self.serverfield.
               text())
315         settings.setValue("pointercolor",self.
               pointercolor.color)
316         settings.setValue("linecolor",self.linecolor
               .color)
317         settings.setValue("pointersize",self.
               pointersize.value())
318         settings.setValue("linewidth",self.linewidth
               .value())
319         self.close()
320
321
322 #
    ############################################################

323 # Widget: Overlay widget, showing cursor and drawing
```

```
324 #
    ##########################################################

325
326 class Overlay(QWidget):
327
328     def __init__(self, parent = None):
329         QWidget.__init__(self, parent)
330         palette = QPalette(self.palette())
331         palette.setColor(palette.Background, Qt.
              transparent)
332         self.setPalette(palette)
333         self.x = 0
334         self.y = 0
335
336
337         settings = QSettings()
338         # Set options another place
339
340         self.pointercolor = QColor(settings.value("
              pointercolor"))
341         self.pointersize = settings.value("
              pointersize").toInt()[0]
342         self.linecolor = QColor(settings.value("
              linecolor"))
343         self.linewidth = settings.value("linesize").
              toInt()[0]
344
345         self.paths = None
346
347     def paintEvent(self, event):
348
349         painter = QPainter()
350         painter.begin(self)
351         painter.setRenderHint(QPainter.Antialiasing)
352         painter.setPen(QPen(self.linecolor, self.
              linewidth))
353         painter.setBrush(QBrush(self.pointercolor))
354
355         if self.x and self.y:
```

```
356             painter.drawEllipse(QRectF(self.x-self.
                   pointersize , self.y-self.pointersize
                    , 2*self.pointersize , 2*self.
                   pointersize ))
357
358         if self.paths:
359             for path in self.paths:
360                 pg = QPolygon(path)
361                 painter.drawPolyline(pg)
362
363         painter.end()
364
365
366
367
368 ######################################################
369 # Widget: Loading a presentation (for application
       startup
370 #         and if the current presentation was closed
       )
371 ######################################################
372
373 class LoadPresentationWidget(QWidget):
374
375     def __init__(self, controller):
376         QWidget.__init__(self)
377         self.controller = controller
378
379         loadlocalbtn = QPushButton()
380         loadlocalbtn.setStatusTip("Open file")
381         loadlocalbtn.setIcon(QIcon("icons/document-
               open.svg"))
382         loadlocalbtn.setIconSize(QSize(64,64))
383         loadlocalbtn.setFlat(True)
384
385         loadserverbtn = QPushButton()
386         loadserverbtn.setStatusTip("Open from server
               ")
387         loadserverbtn.setIcon(QIcon("icons/document-
               remote-open.svg"))
388         loadserverbtn.setIconSize(QSize(64,64))
```

```
389         loadserverbtn.setFlat(True)
390
391         loadnewbtn = QPushButton()
392         loadnewbtn.setStatusTip("New file")
393         loadnewbtn.setIcon(QIcon("icons/document-new
               .svg"))
394         loadnewbtn.setIconSize(QSize(64,64))
395         loadnewbtn.setFlat(True)
396
397         hbox = QHBoxLayout()
398         hbox.addStretch(1)
399         hbox.addWidget(loadlocalbtn)
400         hbox.addWidget(loadserverbtn)
401         hbox.addWidget(loadnewbtn)
402         hbox.addStretch(1)
403
404         vbox = QVBoxLayout()
405         vbox.addStretch(1)
406         vbox.addLayout(hbox)
407         vbox.addStretch(1)
408         self.setLayout(vbox)
409
410         self.connect(loadlocalbtn, SIGNAL("clicked()
               "), self.controller.openPresentation)
411         self.connect(loadserverbtn, SIGNAL("clicked
               ()"), self.controller.
               openRemotePresentation)
412         self.connect(loadnewbtn, SIGNAL("clicked()")
               , self.controller.newPresentation)
413
414
415 ##################################################
416 # Widget: Metadata for a new presentation
417 ##################################################
418
419 class NewPresentationWidget(QWidget):
420
421     def __init__(self, controller):
422         QWidget.__init__(self)
423         self.controller = controller
424
```

```
425         self.titlefield = QLineEdit()
426         self.namefield = QLineEdit()
427         self.emailfield = QLineEdit()
428         self.classfield = QComboBox()
429         self.classfield.setEditable(True)
430         self.orgfield = QComboBox()
431         self.orgfield.setEditable(True)
432 
433         cancelbtn = QPushButton(self.tr("C&ancel"),
              self)
434         createbtn = QPushButton(self.tr("C&reate"),
              self)
435         createbtn.setDefault(True)
436 
437         formLayout = QFormLayout()
438         formLayout.addRow(self.tr("&Title:"), self.
              titlefield)
439         formLayout.addRow(self.tr("Your &Name:"),
              self.namefield)
440         formLayout.addRow(self.tr("Your &Email:"),
              self.emailfield)
441         formLayout.addRow(self.tr("&Class:"), self.
              classfield)
442         formLayout.addRow(self.tr("&Organisation:"),
               self.orgfield)
443 
444         buttonLayout = QHBoxLayout()
445         buttonLayout.addStretch(1)
446         buttonLayout.addWidget(cancelbtn)
447         buttonLayout.addWidget(createbtn)
448 
449         vbox = QVBoxLayout()
450         vbox.addWidget(QLabel("<b>" + self.tr("New
              Presentation") +  "</b>"))
451         vbox.addLayout(formLayout)
452         vbox.addStretch(1)
453         vbox.addLayout(buttonLayout)
454 
455         self.setLayout(vbox)
456         self.connect(createbtn, SIGNAL("clicked()"),
               self.create)
```

```
457         self.connect(cancelbtn, SIGNAL("clicked()"),
                self.controller.closePresentation)
458
459     def create(self):
460         self.controller.createPresentation(self.
               titlefield.text(),self.namefield.text(),
               self.emailfield.text(),self.classfield.
               currentText(),self.orgfield.currentText()
               )
461
462 ######################################################
463 # Widget: The presenter display
464 ######################################################
465
466 class PresentationWidget(QWidget):
467
468     def __init__(self, controller):
469         QWidget.__init__(self)
470         self.controller = controller
471
472         # Main Layout
473         splitter = QSplitter(Qt.Vertical)
474         lowerlayout = QHBoxLayout()
475         upperlayout = QHBoxLayout()
476         upperpart = QWidget()
477         upperpart.setLayout(upperlayout)
478         lowerpart = QWidget()
479         lowerpart.setLayout(lowerlayout)
480         splitter.addWidget(upperpart)
481         splitter.addWidget(lowerpart)
482         splitter.setSizes([1,1])
483
484         # Chat Box
485         chatwidget = QWidget()
486         chatlayout = QVBoxLayout()
487         chatfield = QLineEdit()
488         chatview = QTextEdit()
489         chatview.setReadOnly(True)
490         chatlayout.addWidget(chatview)
491         chatlayout.addWidget(chatfield)
492         chatwidget.setLayout(chatlayout)
```

```
493
494         # Slide Overview
495         slideoverview = QWidget()
496         slideoverviewlist = DragDropListWidget(self.
               controller)
497         slideoverviewlist.setIconSize(QSize(64,64))
498         slideoverviewlist.setSelectionMode(
               QAbstractItemView.SingleSelection)
499         slideoverviewlayout = QGridLayout()
500         slideoverviewlayout.addWidget(
               slideoverviewlist,0,0)
501         slideoverview.setLayout(slideoverviewlayout)
502
503         # Left and Right Slide
504         self.leftslide = SlideWidget()
505         self.rightslide = SlideWidget()
506
507         upperlayout.addWidget(self.leftslide)
508         upperlayout.addWidget(self.rightslide)
509         lowerlayout.addWidget(chatwidget)
510         lowerlayout.addWidget(slideoverview)
511
512         layout = QGridLayout()
513         layout.addWidget(splitter,0,0)
514         layout.setMargin(0)
515         self.setLayout(layout)
516
517         self.connect(self.controller.presentation,
               SIGNAL("changed()"),slideoverviewlist.
               update)
518         self.connect(self.controller,SIGNAL("
               updateSlides()"),self.updateSlideView)
519         self.connect(chatfield,SIGNAL("returnPressed
               ()"),lambda: self.chatSend(chatfield))
520         self.connect(self.controller,SIGNAL("
               chatRecieved(QString)"),chatview.append)
521
522     def updateSlideView(self):
523         self.leftslide.setSlide(self.controller.
               currentslide)
```

```
524         self.rightslide.setSlide(self.controller.
              nextslide)
525
526     def chatSend(self,field):
527         self.controller.chatSend(field.text())
528         field.setText("")
529
530
531
532 ##################################################
533 # Widget: A QListWidget that you can drag and
534 #         drop items in
535 ##################################################
536
537 class DragDropListWidget(QListWidget):
538     def __init__(self, controller):
539         QListWidget.__init__(self)
540         self.controller = controller
541         self.presentation = controller.presentation
542         self.setDragDropMode(self.InternalMove)
543         self.installEventFilter(self)
544         self.connect(self,SIGNAL("
              itemSelectionChanged()"),self.
              updateSelection)
545         self.connect(self.controller,SIGNAL("
              setNextSlide(int)"),self.setrow)
546
547     def eventFilter(self, sender, event):
548         if (event.type() == QEvent.ChildRemoved):
549             self.reorderModel()
550         return False
551
552     def contextMenuEvent(self, s):
553         item = self.itemAt(s.pos())
554         if item:
555             i = item.data(Qt.UserRole).toPyObject()
556             menu = QMenu(i.getTitle(),self)
557             action = QAction(self)
558             action.setText("Remove slide")
559             action.setIcon(QIcon("icons/list-remove.
                  svg"))
```

```
560             self.connect(action,SIGNAL("triggered()"
                    ),lambda: self.presentation.
                    removeSlide(self.row(item)))
561             menu.addAction(action)
562             menu.exec_(s.globalPos())
563
564     def update(self):
565         index = self.currentRow()
566         self.clear()
567         for item in self.presentation.slides:
568             i = QListWidgetItem(item.asIcon(),item.
                    getTitle())
569             i.setData(Qt.UserRole,item)
570             self.addItem(i)
571         if self.count() > 0 and index == -1:
572             index = 0
573         self.setCurrentRow(index)
574
575     def updateSelection(self):
576         self.controller.setNextSlide(self.currentRow
                ())
577
578     def reorderModel(self):
579         slides = []
580         for i in range(0,self.count()):
581             slide = self.item(i).data(Qt.UserRole).
                    toPyObject()
582             slides.append(slide)
583         self.presentation.slides = slides
584         self.presentation.setSaveNeeded()
585
586     def setrow(self,i):
587         self.setCurrentRow(i)
588
589 ######################################################
590 # Widget: Slide View Widget
591 ######################################################
592
593 class SlideWidget(QWidget):
594
595     def __init__(self,overlay=None):
```

```
596         QWidget.__init__(self)
597         self.overlay = overlay
598         self.slide = QWidget()
599         self.layout = QHBoxLayout()
600         self.layout.addWidget(self.slide)
601         self.setLayout(self.layout)
602         if self.overlay:
603             wl = QHBoxLayout()
604             wl.addWidget(self.overlay)
605             self.slide.setLayout(wl)
606 
607     def setSlide(self, slide):
608         self.slide.hide()
609         self.layout.removeWidget(self.slide)
610         self.slide = slide.asWidget()
611         self.layout.addWidget(self.slide)
612         if self.overlay:
613             wl = QHBoxLayout()
614             wl.addWidget(self.overlay)
615             self.slide.setLayout(wl)
616 
617 ##################################################
618 # Widget: Color Chooser Button
619 ##################################################
620 
621 class ColorButton(QPushButton):
622 
623     def __init__(self):
624         QPushButton.__init__(self)
625         self.setColor(QColor("black"))
626         self.connect(self,SIGNAL("clicked()"),self.
               choose)
627 
628     def setColor(self, color):
629         self.color = color
630         self.update()
631 
632     def paintEvent(self, event):
633         super(ColorButton,self).paintEvent(event)
634         painter = QPainter()
635         painter.begin(self)
```

```
636         painter.setBrush(QBrush(self.color))
637         painter.drawRect(7,7,self.size().width()-14,
               self.size().height()-14)
638         painter.end()
639
640     def choose(self):
641         color = QColorDialog.getColor(self.color,
               self, self.tr("Pick a color"))
642         print color
643         self.setColor(color)
```

C.6 plugins.py

../emcee–gui–client/plugins.py

```
1  import os
2  import sys
3
4  content = []
5  pointer = []
6  mimehandlers = {}
7
8  sys.path.insert(0,"plugins/")
9
10 for f in os.listdir("plugins/"):
11     if f.endswith(".py"):
12         module = __import__(f[:-3])
13
14         if module.plugintype == "content":
15             content.append(module)
16             mimehandlers[module.mimetype] = module
17         if module.plugintype == "pointer":
18             pointer.append(module)
```

C.7 plugin: blank.py

../emcee–gui–client/plugins/blank.py

```
1  from PyQt4.QtCore import *
2  from PyQt4.QtGui import *
3
4  class widget(QWidget):
5
```

```
6       def __init__(self,container=None,index=0):
7           super(widget,self).__init__()
8           self.index = index
9           self.aspect = 4/float(3)
10
11      def paintEvent(self,arg):
12          owidth = width = self.size().width()
13          oheight = height = self.size().height()
14
15          if float(width)/float(height) > self.aspect:
16              width = int(height*self.aspect)
17          else:
18              height = int(width/self.aspect)
19
20          painter = QPainter(self)
21          painter.fillRect(int((owidth-width)/2), int
               ((oheight-height)/2), width, height,
               QColor("black"))
22
23
24      def sizeHint(self):
25          width = self.size().width()
26          height = self.size().height()
27
28          a = 0
29          if height > 0:
30              float(width)/float(height)
31
32          if  a > self.aspect:
33              width = int(height*self.aspect)
34          else:
35              height = int(width/self.aspect)
36
37          return QSize(width,height)
38
39  class icon(QIcon):
40
41        def __init__(self,container,index=0,width=128,
             height=96):
42          image = QPixmap(width,height)
43          image.fill(QColor("black"))
```

```
44          QIcon.__init__(self,image)
45
46
47  class container:
48
49      def __init__(self,reference=None):
50          pass
51
52      def numSlides(self):
53          return 1
54
55      def getName(self,index=0):
56          return "Blank"
57
58      def getIdentifier(self):
59          return "*BLANKSLIDE"
60
61
62  plugintype = 'content'
63  mimetype   = 'application/x-blank-slide'
64  name       = 'Blank Slide'
65  source     = 'internal'
```

C.8 plugin: keyboard.py

../emcee–gui–client/plugins/keyboard.py

```
1  from PyQt4.QtCore import *
2  from PyQt4.QtGui import *
3
4  class pointer(QObject):
5
6      def __init__(self, controller):
7          self.controller = controller
8          self.next1 = QShortcut(QKeySequence("PgDown"
               ),self.controller.mainview)
9          self.prev1 = QShortcut(QKeySequence("PgUp"),
               self.controller.mainview)
10         self.next2 = QShortcut(QKeySequence("Right")
               ,self.controller.mainview)
11         self.prev2 = QShortcut(QKeySequence("Left"),
               self.controller.mainview)
```

```
12
13      def enable(self):
14          self.connect(self.prev1,SIGNAL("activated()"
                ),self.controller.previousSlide)
15          self.connect(self.next1,SIGNAL("activated()"
                ),self.controller.nextSlide)
16          self.connect(self.prev2,SIGNAL("activated()"
                ),self.controller.previousSlide)
17          self.connect(self.next2,SIGNAL("activated()"
                ),self.controller.nextSlide)
18          return True
19
20      def disable(self):
21          self.disconnect(self.prev1,SIGNAL("activated
                ()"),self.controller.previousSlide)
22          self.disconnect(self.next1,SIGNAL("activated
                ()"),self.controller.nextSlide)
23          self.disconnect(self.prev2,SIGNAL("activated
                ()"),self.controller.previousSlide)
24          self.disconnect(self.next2,SIGNAL("activated
                ()"),self.controller.nextSlide)
25          return True
26
27
28
29  plugintype      = 'pointer'
30  name            = 'Keyboard'
31  description     = 'Keyboard and keyboard-like remotes
        '
32  capabilities    = ['control']
33  enabledefault   = True
```

C.9 plugin: pdf.py

../emcee–gui–client/plugins/pdf.py

```
1  from PyQt4.QtCore import *
2  from PyQt4.QtGui import *
3
4  import QtPoppler
5  import os.path
6  import hashlib
```

```
7
8 MAGICDPI = 72
9
10 class widget(QWidget):
11
12     def __init__(self,container,index=0):
13         super(widget,self).__init__()
14         self.page = container.getSlide(index)
15         self.pagesize = self.page.pageSize()
16         self.aspect = self.pagesize.width()/float(
               self.pagesize.height())
17         self.index = index
18         self.cache = container.cache
19
20     def paintEvent(self,arg):
21
22         owidth = width = self.size().width()
23         oheight = height = self.size().height()
24         if float(width)/float(height) > self.aspect:
25             width = int(height*self.aspect)
26             scale = width / float(self.pagesize.
                   width())
27         else:
28             height = int(width/self.aspect)
29             scale = height / float(self.pagesize.
                   height())
30
31         if self.cache.has_key(self.cacheindex(width,
               height)):
32             image = self.cache[self.cacheindex(width
                   ,height)]
33         else:
34             image = self.page.renderToImage(scale*
                   MAGICDPI,scale*MAGICDPI)
35             self.cache[self.cacheindex(width,height)
                   ] = image
36
37         painter = QPainter(self)
38         painter.drawImage(int((owidth-width)/2),int
               ((oheight-height)/2),image)
39
```

```
40      def cacheindex(self,width,height):
41          return hashlib.sha1(str(width) + "x" + str(
               height) + "*" + str(self.index)).
               hexdigest()
42
43      def sizeHint(self):
44          width = self.size().width()
45          height = self.size().height()
46
47          a = 0
48          if height > 0:
49              float(width)/float(height)
50
51          if  a > self.aspect:
52              width = int(height*self.aspect)
53          else:
54              height = int(width/self.aspect)
55
56          return QSize(width,height)
57
58  class icon(QIcon):
59
60        def __init__(self,container,index=0,width=128,
             height=96):
61          page = container.getSlide(index)
62          image = QPixmap.fromImage(page.renderToImage
               ())
63          QIcon.__init__(self,image)
64
65
66  class container:
67
68      def __init__(self,pdffile):
69
70          self.filename = os.path.basename(pdffile)
71
72          f=open(pdffile)
73          d=f.read()
74          f.close()
75
76          self.identifier = hashlib.md5(d).hexdigest()
```

```
77
78          self.document = QtPoppler.Poppler.Document.
               load(pdffile)
79          self.document.setRenderHint(QtPoppler.
               Poppler.Document.Antialiasing and
               QtPoppler.Poppler.Document.
               TextAntialiasing)
80          self.numslides = self.document.numPages()
81          size = self.document.page(0).pageSize()
82          self.cache = {}
83
84          self.namemap = self.parseToc(self.document.
               toc().firstChild())
85
86      def numSlides(self):
87          return self.numslides
88
89      def getSlide(self,index):
90          if index < self.numslides and index >= 0:
91              return self.document.page(index)
92          else:
93              raise IndexError()
94
95      def getName(self,index=0):
96          if self.namemap.has_key(index+1):
97              return self.namemap[index+1]
98          else:
99              return self.filename + " slide " + str(
                   index+1)
100
101     def getIdentifier(self):
102         return self.identifier
103
104     def parseToc(self, toc, hashmap={}):
105         if not toc.isNull():
106             name = str(toc.nodeName())
107
108             dest = toc.attributes().namedItem("
                   DestinationName")
109             if not dest.isNull():
```

```
110                 page = self.document.linkDestination
                       (dest.nodeValue()).pageNumber()
111                 hashmap[page] = name
112
113             dest = toc.attributes().namedItem("
                   Destination")
114             if not dest.isNull():
115                 pages = dest.nodeValue().split(";")
116                 page = int(pages[1])
117                 hashmap[page] = name
118
119             self.parseToc(toc.firstChild(),hashmap)
120             self.parseToc(toc.nextSibling(),hashmap)
121             return hashmap
122
123
124
125
126 plugintype = 'content'
127 mimetype   = 'application/pdf'
128 name       = 'PDF File'
129 source     = 'file'
130 filetype   = 'pdf'
```

C.10 plugin: wiimote.py

../emcee–gui–client/plugins/wiimote.py

```
 1 import cwiid
 2 import time
 3 import math
 4 #import Queue
 5 from threading import Thread
 6
 7 from PyQt4.QtCore import *
 8 from PyQt4.QtGui import *
 9
10 class pointer(QThread):
11
12     def __init__(self, controller):
13         QThread.__init__(self)
14         self.controller = controller
```

```
15
16      def __del__(self):
17          self.disable()
18
19      def enable(self):
20
21          self.controller.setStatus("Press 1 & 2 on
               the Wiimote simultaneously to find it")
22          self.connect(self,SIGNAL("setStatus(QString)
               "), self.controller.setStatus)
23
24          self.w = None
25          self.valid = False
26          self.coordinatepool = []
27          self.x = 0
28          self.y = 0
29          self.battery = 0
30          self.alive = False
31          self.start()
32
33          return True
34
35      def disable(self):
36          self.finished = True
37          return True
38
39      def battery(self):
40          return (self.battery/cwiid.BATTERY_MAX)*100
41
42      def xy(self):
43          if self.valid:
44              return (self.x,self.y)
45          else:
46              return None
47
48      def attention(self):
49          self.w.rumble = 1
50          time.sleep(0.5)
51          self.w.rumble = 0
52
53      ####
```

```
54
55      def run(self):
56              self.alive = True
57              try:
58                  self.w = cwiid.Wiimote()
59              except Exception as e:
60                  self.emit(SIGNAL("setStatus(QString)
                        "),QString(str(e)))
61                  self.alive = False
62
63              if self.alive:
64                  self.connect(self,SIGNAL("left()"),
                        self.controller.previousSlide)
65                  self.connect(self,SIGNAL("right()"),
                        self.controller.nextSlide)
66                  self.connect(self,SIGNAL("draw()"),
                        lambda: self.controller.drawing.
                        drawBegin())
67                  self.connect(self,SIGNAL("keyup()"),
                        lambda: self.controller.drawing.
                        drawEnd())
68                  self.connect(self,SIGNAL("clear()"),
                        lambda: self.controller.drawing.
                        drawClear())
69
70
71                  # Set Wii Parameters to get
72                  self.w.enable(cwiid.FLAG_MESG_IFC)
73                  self.w.rpt_mode = cwiid.RPT_IR |
                        cwiid.RPT_BTN
74                  self.finished = False
75                  self.emit(SIGNAL("setStatus(QString)
                        "),QString("Wiimote connected"))
76                  self.attention()
77
78                  while not self.finished:
79                      self.usleep(100)
80                      try:
81                          messages = self.w.get_mesg()
82                          for mesg in messages:
```

```
83                              if mesg[0] == cwiid.
                                   MESG_IR:
84                                  self.handle_ir(mesg
                                       [1])
85                              elif mesg[0] == cwiid.
                                   MESG_BTN:
86                                  self.handle_key(mesg
                                       [1])
87
88                          self.battery = self.w.state[
                                "battery"]
89                      except Exception:
90                          self.finished = True
91                          pass
92                  if self.w:
93                      self.w.close()
94
95
96      def handle_ir(self,mesg):
97          points = [None,None]
98
99          # add the two largest points to points
100         for s in mesg:
101             if s:
102                 for i,p in enumerate(points):
103                     if p:
104                         if p['size'] < s['size']:
105                             points[i] = s
106                             break
107                     else:
108                         points[i] = s
109                         break
110
111         if points[0] and points[1]:
112             x1 = float(points[0]['pos'][0])/cwiid.
                    IR_X_MAX
113             y1 = float(points[0]['pos'][1])/cwiid.
                    IR_Y_MAX
114             x2 = float(points[1]['pos'][0])/cwiid.
                    IR_X_MAX
```

```
115             y2 = float(points[1]['pos'][1])/cwiid.
                    IR_Y_MAX
116
117             midx = 1 - ((x1+x2) / 2)
118             midy = (y1+y2) / 2
119             ang = math.atan2((y1-y2)*0.75,x1-x2)
120             dist = math.sqrt(math.pow((y1-y2)
                    *0.75,2)+math.pow(x1-x2,2))
121             self.valid = True
122         else:
123             self.valid = False
124
125         if self.valid:
126             # rotate the points
127             cx = 0.5 - midx
128             cy = 0.5 - midy
129             if ang > math.pi / 2:
130                 ang -= math.pi
131             elif ang < -math.pi / 2:
132                 ang += math.pi
133
134             x = cx * math.cos(ang) - cy * math.sin(
                    ang)
135             y = cx * math.sin(ang) + cy * math.cos(
                    ang)
136             midx = 0.5 + x
137             midy = 0.5 + y
138
139             self.coordinatepool.append((midx,midy))
140
141             while len(self.coordinatepool) > 10:
142                 self.coordinatepool.pop(0)
143
144             x = 0
145             y = 0
146
147             for i in self.coordinatepool:
148                 x += i[0]
149                 y += i[1]
150
```

```
151             self.x = x/float(len(self.coordinatepool
                    ))
152             self.y = y/float(len(self.coordinatepool
                    ))
153         else:
154             self.coordinatepool = []
155 
156     def handle_key(self,p):
157         if p == cwiid.BTN_RIGHT:
158             self.emit(SIGNAL("right()"))
159         elif p == cwiid.BTN_LEFT:
160             self.emit(SIGNAL("left()"))
161         elif p == cwiid.BTN_A:
162             self.emit(SIGNAL("draw()"))
163         elif p == cwiid.BTN_B:
164             self.emit(SIGNAL("clear()"))
165         elif p == 0:
166             self.emit(SIGNAL("keyup()"))
167         else:
168             print "Unhandled keypress: %x" % p
169 
170 #       BTN_2=0x0001
171 #     BTN_1=0x0002
172 #     BTN_B=0x0004
173 #     BTN_MINUS=0x0010
174 #     BTN_HOME=0x0080
175 #     BTN_DOWN=0x0400
176 #     BTN_UP=0x0800
177 #     BTN_PLUS=0x1000
178 
179 
180 
181 
182 plugintype     = 'pointer'
183 name           = 'Wiimote'
184 description    = 'Bluetooth Wiimote'
185 capabilities   = ['control','draw','attention','
        battery']
186 enabledefault  = False
```

C.11 presentation.py

../emcee–gui–client/presentation.py

```
1  import os.path
2  import mimetypes
3  import unicodedata
4  import re
5  import json
6  import hashlib
7
8  from PyQt4.QtCore import *
9  from PyQt4.QtGui import *
10
11 import plugins
12
13 class Presentation(QObject):
14
15     def __init__(self, filename=None):
16         QObject.__init__(self)
17
18         self.slides = []
19         self.defaultSlide = Slide(Source(None,"
               application/x-blank-slide"),0)
20         self.saveneeded = True
21         self.currentindex = -1
22         self.nextindex = -1
23
24         if filename:
25             self.filename = filename
26             self.saveneeded = False
27
28             openfile = open(self.filename,"r")
29             openfilejson = json.load(openfile)
30             openfile.close()
31
32             self.title = openfilejson["metadata"]["
                   title"]
33             self.name = openfilejson["metadata"]["
                   author"]
```

```
34              self.email = openfilejson["metadata"]["
                   email"]
35              self.forclass = openfilejson["metadata"
                   ]["class"]
36              self.organization = openfilejson["
                   metadata"]["organization"]
37
38              tmpsrc = {}
39              for (ident,source) in openfilejson["
                   sources"].items():
40                  tmpsrc[ident] = Source(source["
                       reference"], source["mimetype"])
41
42              for slide in openfilejson["slides"]:
43                  self.slides.append(Slide(tmpsrc[
                       slide["source"]],slide["index"]))
44
45          else:
46              self.filename = None
47              # metadata
48              self.title = self.tr("Untitled
                   Presentation")
49              self.name = ""
50              self.email = ""
51              self.forclass = ""
52              self.organization = ""
53
54      def save(self,filename=None):
55          if filename:
56              self.filename = filename
57
58          if self.filename:
59              savefile = open(self.filename,"w")
60              savefile.write(self.asJSON())
61              savefile.close()
62              self.saveneeded = False
63
64      def addSource(self, source):
65          for i in range(0,source.numSlides()):
66              self.addSlide(source.getSlide(i))
67
```

```
68      def addSlide(self, slide):
69          self.slides.append(slide)
70          self.emit(SIGNAL("changed()"))
71          self.setSaveNeeded()
72
73      def getSlide(self, index):
74          return self.slides[index]
75
76      def removeSlide(self, index):
77          if len(self.slides) > index:
78              self.slides.pop(index)
79              self.emit(SIGNAL("changed()"))
80              self.setSaveNeeded()
81
82      def setSaveNeeded(self):
83          self.saveneeded = True
84
85      def suggestedFileName(self):
86          if len(self.title)  > 0:
87              x = unicodedata.normalize('NFKD',
                    unicode(self.title)).encode('ascii',
                    'ignore')
88              x = unicode(re.sub('[^\w\s-]', '', x).
                    strip().lower())
89              x = re.sub('[-\s]+', '-', x)
90              return str(x)
91          else:
92              return self.tr("untitled")
93
94      def asJSON(self):
95          metadata = {"title": self.title,
96                      "author": self.name,
97                      "email": self.email,
98                      "class": self.forclass,
99                      "organization": self.
                          organization}
100         slides = []
101         sources = {}
102
103         for slide in self.slides:
```

```
104             sources[slide.source.identifier()] = {"
                   reference": slide.source.reference, "
                   mimetype": slide.source.mimetype}
105             slides.append({"index": slide.index, "
                   name": slide.getTitle(), "source":
                   slide.source.identifier()})
106
107         return json.dumps({"metadata": metadata, "
               slides": slides, "sources": sources})
108
109 class Source:
110
111     def __init__(self, reference, mimetype=None):
112         if mimetype == None:
113             if os.path.exists(reference):
114                 mimetype = mimetypes.guess_type(
                       reference)[0]
115             else:
116                 raise RuntimeError("No mimetype was
                       found for reference: %s" %
                       reference)
117
118         self.mimetype = mimetype
119         self.reference = reference
120         self.handler = plugins.mimehandlers[mimetype
               ]
121         self.container = self.handler.container(self
               .reference)
122
123     def numSlides(self):
124         return self.container.numSlides()
125
126     def getSlide(self,index=0):
127         return Slide(self,index)
128
129     def identifier(self):
130         return self.container.getIdentifier()
131
132
133 class Slide:
134
```

```
135     def __init__(self, source, index):
136         self.source = source
137         self.handler = self.source.handler
138         self.index = index
139         self.title = self.source.container.getName(
              self.index)
140         self.icon = self.handler.icon(self.source.
              container,self.index)
141 
142     def asWidget(self):
143         return self.handler.widget(self.source.
              container,self.index)
144 
145     def asIcon(self):
146         return self.icon
147 
148     def getTitle(self):
149         return self.title
```

C.12 util.py

../emcee–gui–client/util.py

```
1  import math
2  
3  # quick and dirty vertex reduction
4  def vertexreduce(points, tolerance):
5      newpoints = []
6      prevpoint = points[0]
7      newpoints.append(points[0])
8      for point in points[1:]:
9          d = math.sqrt((prevpoint[0] - point[0]) ** 2
               + (prevpoint[1] - point[1]) ** 2)
10         if d > tolerance:
11             newpoints.append(point)
12             prevpoint = point
13     return newpoints
14 
15 # FROM: http://mappinghacks.com/code/dp.py.txt
16 # pure-Python Douglas-Peucker line simplification/
     generalization
```

```
17 # this code was written by Schuyler Erle <
       schuyler@nocat.net> and is
18 #   made available in the public domain.
19 def douglaspeucker(pts, tolerance):
20     anchor  = 0
21     floater = len(pts) - 1
22     stack   = []
23     keep    = set()
24 
25     stack.append((anchor, floater))
26     while stack:
27         anchor, floater = stack.pop()
28 
29         # initialize line segment
30         if pts[floater] != pts[anchor]:
31             anchorX = float(pts[floater][0] - pts[
                   anchor][0])
32             anchorY = float(pts[floater][1] - pts[
                   anchor][1])
33             seg_len = math.sqrt(anchorX ** 2 +
                   anchorY ** 2)
34             # get the unit vector
35             anchorX /= seg_len
36             anchorY /= seg_len
37         else:
38             anchorX = anchorY = seg_len = 0.0
39 
40         # inner loop:
41         max_dist = 0.0
42         farthest = anchor + 1
43         for i in range(anchor + 1, floater):
44             dist_to_seg = 0.0
45             # compare to anchor
46             vecX = float(pts[i][0] - pts[anchor][0])
47             vecY = float(pts[i][1] - pts[anchor][1])
48             seg_len = math.sqrt( vecX ** 2 + vecY **
                   2 )
49             # dot product:
50             proj = vecX * anchorX + vecY * anchorY
51             if proj < 0.0:
52                 dist_to_seg = seg_len
```

```
53              else:
54                  # compare to floater
55                  vecX = float(pts[i][0] - pts[floater
                        ][0])
56                  vecY = float(pts[i][1] - pts[floater
                        ][1])
57                  seg_len = math.sqrt( vecX ** 2 +
                        vecY ** 2 )
58                  # dot product:
59                  proj = vecX * (-anchorX) + vecY * (-
                        anchorY)
60                  if proj < 0.0:
61                      dist_to_seg = seg_len
62                  else:  # calculate perpendicular
                        distance to line (pythagorean
                        theorem):
63                      dist_to_seg = math.sqrt(abs(
                            seg_len ** 2 - proj ** 2))
64                  if max_dist < dist_to_seg:
65                      max_dist = dist_to_seg
66                      farthest = i
67
68          if max_dist <= tolerance: # use line segment
69              keep.add(anchor)
70              keep.add(floater)
71          else:
72              stack.append((anchor, farthest))
73              stack.append((farthest, floater))
74
75      keep = list(keep)
76      keep.sort()
77      return [pts[i] for i in keep]
```

C.13 pypoppler-qt4.patch

../emcee–gui–client/pypoppler–qt4.patch

```
1 --- pypoppler-qt4-hacked/poppler-qt4.sip  2010-10-03
      19:25:09.633023511 +0200
2 +++ pypoppler-qt4/poppler-qt4.sip 2008-11-04
     10:21:35.000000000 +0100
3 @@ -18,33 +18,8 @@
```

```
4  */
5  namespace Poppler {
6
7 -class LinkDestination {
8 -%TypeHeaderCode
9 -#define UNSTABLE_POPPLER_QT4 1
10-#include <qt4/poppler-qt4.h>
11-#include <qt4/poppler-link.h>
12-%End
13
14-public:
15-         LinkDestination(const QString &description)
       ;
16-         enum Kind
17-         {
18-             destXYZ = 1,
19-             destFit = 2,
20-             destFitH = 3,
21-             destFitV = 4,
22-             destFitR = 5,
23-             destFitB = 6,
24-             destFitBH = 7,
25-             destFitBV = 8
26-         };
27-         Kind kind() const;
28-         int pageNumber() const;
29-         double left() const;
30-         double bottom() const;
31-         double right() const;
32-         double top() const;
33-};
34+
35
36 class TextBox {
37
38@@ -139,8 +114,6 @@
39      bool search(const QString &text, QRectF &rect,
           SearchDirection direction, SearchMode
           caseSensitive) const;
40
41      QList<Poppler::TextBox*> textList() const;
```

```
-
-    QString label() const;
 
     QSizeF pageSizeF() const;
 
@@ -172,8 +145,6 @@
     Page( const Poppler::Page & );
 };
 
-
-
 class Document {
 %TypeHeaderCode
 #define UNSTABLE_POPPLER_QT4 1
@@ -268,7 +239,7 @@
 
     QDomDocument *toc() const;
 
-     Poppler::LinkDestination *linkDestination( const QString &name );
+//     LinkDestination *linkDestination( const QString &name );
 
 //     bool print(const QString &fileName, const QList<int> pageList, double hDPI, double vDPI, int rotate);
```