

# Kamaelia: P2P Whiteboarding & More

Kamaelia is a general purpose open source framework for building software. Yes, I know, another one. It's different though. It's designed to feel very similar to lego or k'nex, and even includes a nascent tool for creating these systems visually. It originated from BBC Research targeted at networked delivery of fun and useful content, but is already useful today.

It includes components for working with Freeview, as well as tools for dealing with multimedia. This allows you to use Pygame, OpenGL, networking (custom servers, clients, protocols including HTTP & Bit Torrent), dirac, vorbis, speex, and a variety of other tools simply and easily in the same system.

This tutorial focusses on our "Whiteboard" application that was written to solve a problem our team faced. Our team is split across multiple sites, which led to a need for real collaboration including audio, and sketching that just works (ie precisely the same problem as many open source projects). It has a spartan interface, which allows you to forget about the application and just use it.

Features:

- You have very basic drawing functions
- It supports multiple pages, which you just go back and forth between
- Each whiteboard can be a client, server or client and server. If you connect your whiteboard to mine, and I change pages, you see the page change. Obviously anything drawn on one whiteboard is shown on all the others connected. Given any whiteboard can be a server and client, this means that this is, at its most basic, a peer to peer whiteboard. Anyone who connects to any server in the network will receive everything.
- It also supports audio using the speex audio codec - which is optimised for people speaking, and to encode to very low bitrates. This means you can chat to anyone connected.
- Sessions can be recorded and played back. Whilst a recording is being played back, you can continue to use the whiteboard as you would normally - as can anyone connected to your network of whiteboards.
- You can also inject MP3 audio into the whiteboard - which is particularly useful for setting the scene for a brainstorming session.
- A command line console
- Experimental capability to cause whiteboards to page through locally stored pages in a semi-synchronised fashion.

## Installation

Kamaelia is written in python, and it is currently developed under SuSE linux, using python 2.4. Any linux distribution should work, however we are unlikely to have tested your setup - feedback to the development team is welcome. You should be able to use Kamaelia with any version of python since 2.2.2, but python 2.4 is recommended.

Installation of Kamaelia comprises of: installing the core of

the system called "Axon", installing the library of components and tools "Kamaelia". The next step is to install any further dependencies for the functionality you wish to use. For convenience below I'm going to assume you are installing as root.

Also, we provide a number of packages regarding installation, and the largest of these - KamaeliaMegaBundle - includes all the major dependencies. Inside this bundle you will find the following required files:

- Axon-1.5.1.tar.gz, Kamaelia-0.5.0.tar.gz

The following recommended files:

- pygame-1.7.1release.tar.gz
- Pyrex-0.9.3.1.tar.gz

The following additional dependencies for audio for the whiteboard:

- Speex capture and playback requires speex-1.0.5.tar.gz, pySpeex-0.2.tar.gz, pymedia-cvs-patched.tar.gz to be installed. pySpeex requires Pyrex (above) for installation
- Loading and saving of images efficiently requires the using of the python imaging library, hence why Imaging-1.1.5.tar.gz is needed.

The other files are useful as follows:

- Bit Torrent support integrates with the standard bit torrent distribution so BitTorrent-4.20.8.tar.gz needs to be installed for that.
- Open GL support requires access to the python Open GL bindings, so PyOpenGL-2.0.2.01.tar.gz needs to be installed for that.
- Dirac support requires dirac-0.5.4.tar.gz, Dirac-0.0.1.tar.gz to be installed in that order, in addition to pyrex.
- Vorbis support requires libao-0.8.6.tar.gz, libogg-1.1.3.tar.gz, libvorbis-1.1.2.tar.gz, pyao-0.82.tar.gz, vorbissimple-0.0.2.tar.gz to be installed in that order, in addition to pyrex.
- Freeview - DVB-T - support requires a recent linux kernel and python-dvb3-0.0.4.tar.gz to be installed (again, in addition to pyrex!)

This set of dependencies is pretty large, but its worth bearing in mind that they are only required to support these external features. If you don't need Dirac, don't install those dependencies! In a way you can think of Axon as being akin to your linux kernel, Kamaelia as being the base installation, and all the other files as optional extras you may want installed.

## Installing Axon

Axon is need to provide the basic communications framework for components. As a result we need to install this first. Unpack, change into the directory, run the installer:

- # tar zxf Axon-1.5.1.tar.gz
- # cd Axon-1.5.1
- # python setup.py install

## Installing Kamaelia

Then we need to install the repository of components.

Unpack, change into the directory, run the installer:

- # tar xzf Kamaelia-0.5.0.tar.gz
- # cd Kamaelia-0.5.0
- # python setup.py install

### Installing Pygame

The Whiteboard uses pygame for display and input. You may already have this installed or available in your distribution's version of pygame. If you haven't, or it's not a recent version installing pygame is essentially the same: unpack, change into the directory, run the installer:

- # tar xzf pygame-1.7.1release.tar.gz
  - # cd pygame-1.7.1release
  - # python setup.py install
- NB You will be asked a number of questions based on your local installation, which is why it may be preferable to use packages from your distribution!

### Installing Speex Audio support

The whiteboard uses the speex audio codec for efficient transmission of audio. First install speex, then pySpeex, and then pymedia. (Speex for raw encoding, pyspeex to make the speex components active, and pymedia for input and output)

- Installing speex is pretty similar - you unpack speex-1.0.5.tar.gz change into the directory. You then type ./configure followed by make; make install
- You then need to install speex support for python by unpacking pySpeex-0.2.tar.gz changing into there and doing python setup.py install .
- Finally to install pymedia, unpack the provided patched version of pymedia,

### Installing the Python Imaging Library

This is the same to install as Axon & Kamaelia, unpack change into the directory Imaging-1.1.5.tar.gz, and do: python setup.py install

### Installing other dependencies

For this tutorial I've only covered the dependencies needed for the whiteboard. For more detailed instructions for all the dependencies either look at the individual packages or look at <http://kamaelia.sourceforge.net/GettingStarted.html> . Freeview/DVB-T support is useful for the PVR related tools for capturing TV for timeshifting in conjunction with certain linux compatible USB DVB-T sticks.

If all has been successful, you will now be left with Kamaelia installed on your system. If you encounter problems, check to see if your package manager has the relevant versions.

### Using the Whiteboard

#### Standalone mode

The whiteboard is currently designed to be run out of the installation directory. However you can move the whiteboard's directory to anywhere convenient on your system. You run the application as follows:

- cd Kamaelia-0.5.0-rc1/Tools/Whiteboard/

- ./Whiteboard.py

Assuming all has gone well, you will be greeted by a blank screen and some colours to choose to draw in, along with an eraser. The system will already be capturing audio, but since we have only run the whiteboard in standalone mode, this won't be going anyway. First, scribble on the first page. To create new pages, click "new page". To page back and forth between pages, click "<<" or ">>".

If you write on a page, and then change pages using "<<" or ">>", then the pages are automatically saved when you change pages. This makes use of the whiteboard, especially on a tablet laptop or using an external tablet (eg the kind you can pick up at a supermarket!) extremely intuitive and familiar for taking notes.

#### Running as a server

To run as a server, you add a port number to serve on.

- ./Whiteboard.py --serveport=1500

As many clients as your hardware can handle can then connect to that server.

#### Running as a client

Suppose the whiteboard above is running on a machine with IP address 192.168.2.5, you would connect a whiteboard to it as follows:

- ./Whiteboard.py --connectto=192.168.2.5:1500

If you're testing on the local machine (ie localhost - aka 127.0.0.1), it is wise to take a copy of the Whiteboard directory before running the second copy:

- cd Kamaelia-0.5.0/Tools/
- cp -R Whiteboard ClientWhiteboard
- cd ClientWhiteboard
- ./Whiteboard.py --connectto=127.0.0.1:1500

(Clearly you could put the copy of the directory anywhere you like)

The reason for taking a copy is because moving between pages you've edited saves the changes, and having two whiteboards overwriting each other would be rather irritating!

Now that you have two whiteboards, you can see that anything you draw on one whiteboard is duplicated onto the other. If anyone locally creates new pages and turns pages, their pages are theirs - each person gets their own set of local pages. The reason for not having synchronised collections of pages is because we found it to be more useful that way - it means you don't need to worry about making sure you're starting from the same set of pages before connecting two whiteboards - you connect and start collaborating.

Also, you can now talk to your friend who's connected. The audio quality really depends heavily on your machines' audio hardware and the microphones you use, so it's worth investing in an external microphone or a headset. A USB microphone can be particularly good here. If you do use an external microphone set it as the default capture source in your systems' audio mixer!

## Running as a Client and Server

This is just a combination of the options:

- `./Whiteboard.py --serveport=1500 --connectto=192.168.2.5:1500`

This sets up the whiteboard to be a true peer in terms of a peer to peer setup. However there is no concept of mesh or tree setup. If you wanted to automate connection to a whiteboard network, that would be an interesting addition.

### Where are the pictures?

One obvious question then is, given the pictures are stored automatically, where are they? They're stored inside the "Scribbles" subdirectory. They're stored as standard PNG files. PNG is used due to being lossless, and therefore working better with the kinds of drawing the Whiteboard is good for.

On my system, they're here:

```
# cd Kamaelia-0.5.0/Tools/Whiteboard/  
# ls Scribbles  
slide.1.png slide.2.png slide.3.png
```

### Advanced Whiteboard Foo

If you've followed the previous steps, you should now be able to use the whiteboard to collaborate with a friend or colleague - even if its something as whimsical as playing noughts and crosses, "boxes", or brainstorming your plan to take over the world. Recording noughts and crosses or boxes might not be that thrilling. However, capturing not just the pages from your plan to world domination (through delivery of mind control devices at christmas perhaps whilst posing as Santa) but the entire session that led up to that would probably be useful.

OK, more practically let's look at how we record a session; how we play a session back; how we can load and save pages to/from arbitrary place on disk; and finally how to feed an MP3 into the running session (eg for transcription).

### Recording a session

First of all, start a server by typing `./Whiteboard.py --serveport=1500`. Then you can start the recorder up by asking it to connect to this server. Assume this server is running on the localhost we type:

- `./WhiteboardRecorder.py whiteboard_session.rec 127.0.0.1 1500`

If the whiteboard server is remote - eg running on 192.168.2.5, we would type:

- `./WhiteboardRecorder.py whiteboard_session.rec 192.168.2.5 1500`

And so on. To stop recording, simply press control-c.

### Playing back a session recording

Again, start a server by typing `./Whiteboard.py --serveport=1500`. Then, again assuming this is running on localhost, just start up the tool to play back in much the same way:

- `./WhiteboardPlayer.py whiteboard_session.rec 127.0.0.1 1500`

If it's running on a remote machine:

- `./WhiteboardPlayer.py whiteboard_session.rec 192.168.2.5 1500`

The nice thing you'll find here is that you can also talk over the session, and scribble over the whiteboard as this plays back. The reason this works is because as far as the system is concerned, the player is just another person connecting to the whiteboard. Similarly that's how the recorder gets the data - it just acts as another client - just storing the data.

### Loading & Saving Manually

As well as the page metaphor, the system can also load and save pages manually. If you flip back to the console you started the whiteboard from, you'll find a small, quiet command prompt. Suppose I'm happy with our plan for how we're loading the sledge and I decide I want to grab a copy, I could do that by typing either of the following:

```
>>> SAVE "/home/michaels/plans.png  
>>> SAVE "/home/michaels/plans.jpg
```

Likewise, if I've already saved a picture, I can load one back by replacing "SAVE" with "LOAD":

```
>>> LOAD "/home/michaels/plans.png  
>>> LOAD "/home/michaels/plans.jpg
```

### Playing back MP3

Playing back MP3s through the session is something that can be extremely annoying, but depending on the content can be extremely useful - such as a podcast you want to make notes on. If you want to do this, you need a whiteboard server running (again, say 192.168.2.5 port 5). Then you'd type:

```
./MP3Player some_podcast.mp3 192.168.2.5 1500
```

Depending on how fast the person speaking speaks, having multiple people transcribing this way can be very effective. As before this is essentially a specialised client, and people can talk over what's being said.

### The Science Bit

Kamaelia works on the principle of taking unix pipelines to the next logical step. The differences are as follows:

- Instead of just pipelines, you can create arbitrary shapes we call graphlines.
- You can send any python object though links in the graphline, rather than being limited to just file-like data flows
- Our components use a python trick to allow the system to be single threaded. Multiple threads can be used if desired, but we are not required to use heavyweight processes unlike unix pipelines.
- This naturally encourages small focussed components and reusability as a result.

To many unix people this approach should be second nature - small focussed pieces of code loosely joined.

The trick we use in python is called a "generator". This is effectively a small, simplified co-routine like object, which can also be thought of as a resumable function. This is best explained by showing you an example.

```

>>> def fib():
...   a,b = 1,1
...   while 1:
...     yield a
...     a,b = b, a+b
...
>>> G = fib()
>>> G
<generator object at 0xb7b59bec>
>>> G.next(), G.next(), G.next(), G.next(), G.next()
(1, 1, 2, 3, 5)

```

As you can see here, this function when called returns a generator object. Python does this because there is a *yield* keyword in the body of the function. We can then call the "next" method of the generator object repeatedly. This effectively gives it some CPU time, and get it to do some work.

We then put this inside a class, calling it "main". This allows us to add on some communications and manage state better, add some metadata about the component, and as a result support things like visual composition of systems using our graphical builder Compose. A simple component for echoing content to the display for example could look like this:

```

from Axon.Component import component
class ConsoleEchoer(component):
    def main(self):
        while 1:
            while self.dataReady("inbox"):
                data = self.recv("inbox")
                print data
            yield 1

```

This takes data from an inbox (much like reading from stdin) and prints it. Conversely, I mentioned we do have threaded components. Suppose we wanted to write a component for reading from the console. That would look like this:

```

from Axon.ThreadedComponent
import threadedcomponent
class ConsoleReader(threadedcomponent):
    def main(self):
        while 1:
            data = raw_input(">>>")
            self.send(data, "outbox")

```

This is essentially the way the console reader works inside the whiteboard - a small focussed component for getting user input. Precisely what it says on the can.

To compose these two into a running system, you create a pipeline:

```

from Kamaelia.Chassis.Pipeline
import Pipeline
Pipeline( ConsoleReader(),
          ConsoleEchoer(), ).run()

```

Similarly, a button in pygame works in much the same focussed way - it renders the button, and when clicked sends out a message.

As final example for now, if we skip how we get the list of files, and the imports, a simple presentation tool can be

created this using a graphline:

```

Graphline(
    CHOOSEER = Chooser(items = files),
    IMAGE = Image(size=(800,600),
                  position=(8,48)),
    NEXT = Button(caption="Next",
                  msg="NEXT", position=(72,8)),
    PREV = Button(caption="Previous",
                  msg="PREV", position=(8,8)),
    FIRST = Button(caption="First",
                  msg="FIRST", position=(256,8)),
    LAST = Button(caption="Last",
                  msg="LAST", position=(320,8)),
    linkages = {
        ("NEXT", "outbox") : ("CHOOSEER", "inbox"),
        ("PREV", "outbox") : ("CHOOSEER", "inbox"),
        ("FIRST", "outbox") : ("CHOOSEER", "inbox"),
        ("LAST", "outbox") : ("CHOOSEER", "inbox"),
        ("CHOOSEER", "outbox") : ("IMAGE", "inbox"),
    }
).run()

```

This says create four buttons "NEXT", "PREVIOUS", "FIRST", "LAST", which send messages to a chooser. This chooses which filename to send out to the image component. The image component then loads and displays the image.

### Closing words

The whiteboard is a nice example of a Kamaelia system. It was faster to write using Kamaelia - the first version in a matter of days. It has been more extensible as a result. It built on top of the existing networking tools, allowing work to solely focus on the application. Adding in audio done by first testing independently and then merging cleanly. The natural separation of concerns, that Kamaelia encourages, simplified this process. Sure there were problems, but those problems would have existed anyway. This approach made them more explicit and easier to resolve.

However the key punchline is this: Kamaelia is designed to make maintenance of concurrent software simple (that's the real research area!). Old Unix hacks will recognise that the graphlines, like pipelines are trivially capable of being made fully concurrent - it's just waiting for the time when hardware systems are largely multi-core. When they are Kamaelia systems will be ready to switch to multicore almost transparently and take full advantage of multicore. Until then we're finding it a simpler more practical way of building software.

We're using it to research better ways of delivering BBC content, and enabling the BBC to work smarter. What would you do where you can join Open GL to your PVR to an IRC component to a webserver, all with relative transparency?

### Author

Michael Sparks is a Senior Research Engineer at BBC Research and Project Lead on the open source Kamaelia project. This article reflects his personal opinions as lead of the Kamaelia project, but does not represent BBC opinion on any subject.