Kamaelia - Networking Using Generators

Michael Sparks
BBC Research & Development

ACCU Python 2005, Oxford
Kamaelia

- Project to explore long term systems for large scale media delivery
  - Forms a concurrency toolkit, focussed mainly on experimenting with network protocols.

- 2 key portions:
  - Axon - Core Component infrastructure, based on communicating generators
  - Kamaelia - Collection of components that use Axon.

- Aim: Scalable, easy & safe concurrent systems
Kamaelia Status

- Released as open source:
  - http://kamaelia.sourceforge.net/

- Axon is at version 1.0.3, and considered feature stable.
  - Runs on Linux, Windows (variety), Mac OS X
  - Specialised distribution for Nokia Series 60 mobiles

- Kamaelia is at version 0.1.2, and growing
  - Ability to write TCP & Multicast clients and servers
  - Variety of simple servers, clients and protocols included
Kamaelia Status

- Kamaelia 0.1.2:
  - Tested on Linux, Windows (variety), Mac OS X
  - Subset on Nokia Series 60 mobiles
- Ease of use hypothesis has been tested with 1 pre-university trainee, looks promising
Kamaelia Motivations

- Large Scale Streaming
  - Several million streams per day
  - Big events have tens of thousands of concurrent viewers
  - Want to scale to handling millions of concurrent viewers
    - Since this could happen.
Kamaelia Motivations

• **What If** 10 years from now...

• the BBC opened the entire archive?
  • Creative Archive is NOT that ambitious! (AFAIK)

• the entire UK got broadband?

• Instantly hit long tail problems
  • 20 million homes?
  • 20 million different things?
  • Not like 20 million people watching BBC1!
Kamaelia Motivations

• Key Problems:
  • RTP was originally conceived for A/V conferencing/telephony
  • Aspects don’t scale well for large scale unidirectional streaming
  • Need a platform for designing, implementing and testing new open standards to scale in this way.
  • Scalability and ability to experiment often conflict.
  • Large scale means highly parallel
  • Scalable concurrency often has a high barrier to entry
    • Limits new ideas, collaboration
Axon

• Kamaelia's Core Concurrency System

• Key aims:
  • Scalable approach
  • Reusable
  • Simple - easy enough for novice programmer to pick up and produce useful systems.
    • Novices see possibilities, not problems
  • Safe - it should be possible to write programs without worrying about race hazards
    • Non locking if possible
Scaling Concurrency

- "Threads are for people who cant program state machines." – Alan Cox (http://tinyurl.com/a66es)

- Processes/Threads/Build your own
  - Processes and threads are well known to be not scalable cross platform.
  - Build your own:
    - Normally means state machines
    - What about people who "cant program state machines"?
      - (Not a dig at Alan !)
Scalability : State machines

• Hard to get 100% right - especially for novices
• Debugging someone else’s - twice as hard
• *State machine is a piece of sequential processing that can release control half way and be restarted retaining state*
• Twisted - at it’s heart very state machine based.
  • Provides a *very* good framework for this and provides *lots* of high quality assistance
  • Still has this barrier to entry (my personal opinion, YMMV)
Scalability or ease?
Do we really have to choose?

- Consider:
  - A **state machine** is a piece of sequential processing that can release control half way and be restarted
  - A **generator** is a piece of sequential processing that can release control half way and be restarted

- Twisted also recognises this: twisted.flow
  - Takes a different approach to composition

- Kamaelia uses generators
  - Hypothesised this would be easier for novices
Kamaelia vs Twisted?

• **NO!**

  - Kamaelia could be integrated into twisted (or vice versa) - we just haven't looked at that yet

  • **Twisted is stable, mature and usable for production systems**

  • Kamaelia isn't mature or suitable for production systems at present
    
    - Won't always be that way, but even when it isn’t we’d rather collaborate rather than compete.

  • Lengthy answer in Kamaelia’s blog
Concurrency is Easy?

- Concurrency is hard
  - ... so why do we let sys admins do it?

- Think unix pipelines:
  - `find -type f | egrep -v '/build/|^./MANIFEST' |while read i ;
    do cp ../Source/$i $i done`

- This has 4 logically concurrent units!
  - Do unix sys admins think of themselves concurrent programmers?
  - Do you think of it that way?
Unix Pipelines

- Concurrent sequential processes - linear
- Items don't know what's next in the pipeline
- Simply communicate with local file handles

- Often forgotten “hidden” details:
  - How data passes between processes
  - The system environment
Axon - Key classes

- Components - self pausing sequential objects that send data to local interfaces
- Linkages - a facility for joining interfaces, allowing system composition
- Scheduler - gives components CPU time
- Postman - The facility for tracking linkages, and handling data transferral
- Co-ordinating Assistant/Tracker (cat) - Provides environmental facilities akin to a Linda tuple space
Axon Components

• Classes with a generator method called "main"

• Augmented by:
  • List of Inboxes - defaults: inbox, control
  • List of Outboxes - defaults: outbox, signal

• class Echo(component):
  def main(self):
    while 1:
      if self.dataReady("inbox"):  
        data = self.recv("inbox")
        self.send(data,"outbox")
    yield 1
Axon Scheduler

• Operation
  • Holds a run queue containing activated components
  • Calls the generator for each component sequentially

• Component Activation
  • If the return value is a newComponent object the components contained are activated (essentially their main() method is called, and the resulting generator stored)

• Component Deactivation
  • If the return value is false, the component is removed from the run queue
Linkages

• Normally join outboxes to inboxes between components
  • out-out and in-in also allowed between parent and nest components

• Linkages can only be create inside a component
  • Inboxes and outboxes designed for connection to subcomponents are considered private and have the naming convention of a leading underscore

• Encourages composition and reuse
class SimpleStreamingClient(component):
    def main(self):
        client=TCPClient("127.0.0.1",1500)
        decoder = VorbisDecode()
        player = AOAudioPlaybackAdaptor()
        self.link((client,"outbox"), (decoder,"inbox")
        self.link((decoder,"outbox"), (player,"inbox"))

        self.addChildren(decoder, player, client)
        yield newComponent(decoder, player, client)
        while 1:
            self.pause()
            yield 1
Linkage Example 2

def AdHocFileProtocolHandler(filename):
    class klass(Kamaelia.ReadFileAdaptor.ReadFileAdaptor):
        def __init__(self,*argv,**argd):
            self.__super.__init__(filename, readmode="bitrate", bitrate=400000)
            return klass

class SimpleStreamingServer(component):
    def main(self):
        server = SimpleServer(protocol=AdHocFileProtocolHandler ("foo.ogg"),
                               port=clientServerTestPort)
        self.addChildren(server)
        yield _Axon.Ipc.newComponent(*(self.children))
        while 1:
            self.pause()
            yield 1
class SimpleMulticastStreamingClient(component):
    def main(self):
        client = Multicast_transceiver("0.0.0.0", 1600, "224.168.2.9", 0)
        adapt = detuple(1)
        decoder = VorbisDecode()
        player = AOAudioPlaybackAdaptor()
        self.link((client,"outbox"), (adapt,"inbox"))
        self.link((adapt,"outbox"), (decoder,"inbox"))
        self.link((decoder,"outbox"), (player,"inbox"))

        self.addChildren(decoder, adapt, player, client)
        yield newComponent(decoder, adapt, player, client)

        while 1:
            self.pause()
            yield 1
Co-ordinating Assistant Tracker

• Tracking Services
  • This allows for the concept of services
  • A service is a mapping of name to (component, inbox) tuple
  • Only ever "need" one 'select' statement in a program for example.
    (want is a different matter!)
  • The Kamaelia.Internet.Selector component offers a "selector" service

• Tracking Values
  • Provides look up and modification of values for keys
  • Use case: to enable stats collection in servers
Howto: Example Component

- MIME/RFC2822 type objects are common in network protocols
  - Email, web, usenet, etc..

- Essentially serialised key/value pairs - much like a dict.

- Create a “MIME Dict” component.
  - Accepts dict like objects, but translates them to MIME-like messages
  - Accepts MIME-like messages, and converts them to dicts.
MimeDictComponent

• How it was written
  • First of all a class that could be a "MIME dict" was written
  • Subclasses dict
  • Always adds a __BODY__ key
  • Replaces __str__ with something that displays the dict as an RFC2822/MIME style message
  • Adds a staticmethod "fromString" as a factory method.

• Written entirely test first without a view to being used as a component
MimeDictComponent 2

- Wanted a component thus:
  - **control** - on which we may receive a shutdown message
  - **signal** - one which we will send shutdown messages
  - **demarshall** - an inbox to which you send strings for turning into dicts
  - **marshall** - an inbox to which you send objects for turning into strings
  - **demarshalled** - an outbox which spits out parsed strings as dicts
  - **marshalled** = an outbox which spits out translated dicts as strings
MimeDictComponent 3

- *Turned out to be simpler to write a generic marshalling component instead, main loop looked like this:*

  ```python
  while 1:
    self.pause()
    if self.dataReady("control"):
      data = self.recv("control")
      if isinstance(data, Axon.Ipc.producerFinished)
        self.send(Axon.Ipc.producerFinished(), "signal")
        return
    if self.dataReady("marshall"):
      data = self.recv("marshall")
      self.send(str(data),"marshalled")
    if self.dataReady("demarshall"):
      data = self.recv("demarshall")
      self.send(self.klass.fromString(data),"demarshalled")
    yield 1
  ```
MimeDictComponent 4

- Subclassing approach:
  - class MimeDictMarshaller(MarshallComponent):
    def __init__(self,*argv,**argd):
      self.__super__.__init__(MimeDict, *argv,**argd)

- Class decoration approach:
  - def MarshallerFactory(klass):
    class newclass(MarshallComponent):
      def __init__(self,*argv,**argd):
        self.__super__.__init__(klass, *argv,**argd)
    return newclass

MimeDictMarshaller=MarshallerFactory(MimeDict)
Summary: New Components

• Longer tutorial based around a multicast transceiver on the website.

• Same approach:
  • Don't worry about concurrency, write single threaded
  • When code works, then convert to components
  • Change control methods into inboxes/outboxes
Ease of use?

- Tested on Ciaran Eaton, a pre-university trainee
  - Happy to let me call him a novice programmer (triple checked)
  - Previous experience: A-Level computer studies - small amount of Visual Basic programming and Access
- 3 Month placement with our group
  - Started off learning python & axon (2 weeks)
  - Created a “learning system” based around parsing a Shakespeare play:
    - Performs filtering, character identification, demultiplexing etc
    - Used pygame for display, stopped short of using pyTTS...
Ease of use? 2

• Ciaran’s project:
  • Created a simplistic low bandwidth video streamer
  • Server has an MPEG video, and takes a frame as JPEG every n seconds
  • This is sent to the client over a framing protocol Ciaran designed and implemented
    • The client then displays the images as they arrive
    • On a PC this uses pygame
    • On a series 60 mobile this uses the native image display calls
  • The idea is this simulates previewing PVR content on a mobile
Ease of use? 3

- Project was successful, Ciaran achieved the goals
- Ciaran wrote all the components for every part of the description.
- Relied on a “SimpleServer” and simple “TCPclient” components - but these only provide reliable data transfer over the network.
- He’s noted that it was a fun experience
  - I find it interesting it was not frustrating given his background.
CSP vs State Machines

• Is this approach inherently worse or better?

• We would suggest neither.

• State machine systems often have intermediate buffers (even single variables) for handoff between state machines

• This is akin to outboxes and inboxes. If they are collapsed into one, as planned, this is equivalent

  • If we do collapse outboxes into inboxes when we create linkages, then the system **should** be as efficient as frameworks like twisted.

• This is currently hypothetical.

(c) 2005 BBC R&D
Integration with other systems

• Default component provides a default main, which calls 3 default callbacks.

• Looks like this:

  • def main(self):
    result = self.initialiseComponent()
    if not result:
      result = 1
    yield result
    while(result):
      result = self.mainBody()
      if result:
        yield result
    yield self.closeDownComponent()
Integration: 2

• Purpose of the 3 callback form is for 2 main reasons

• For those who find callback forms easier to work with

• To allow these methods to be overridden by classes written in:
  • Pyrex
  • C
  • C++
  • ie optimisation of components
Futures

• C++ Version.
  • Simple “miniaxon” version including C++ based generators working. see: cvs:/Code/CPP/Scratch/miniaxon.cpp

• Python Axon will be optimised

• Syntactic Sugar will be added

• Automated component distribution over clusters

• Kamaelia Component Repository

• More protocols, experimental servers:
  • RTSP/RTP initially. New protocols to follow!
Finally: Collaboration

- If you’re interested in working with us, please do
  - If you find the code looks vaguely interesting, please use and give us feedback
  - We’re very open to exploring changes to the system and willing to give people CVS commit access in order to try their ideas.
  - Anyone working with twisted is very welcome to come and criticise and suggest new ideas - integration would be very nice!

- Contacts, project blog:
  - michaels@rd.bbc.co.uk, kamaelia-list@lists.sourceforge.net
  - http://kamaelia.sourceforge.net/cgi-bin/blog/blog.cgi