Timber: Time as a Basis for Embedded real-time systems

Andrew Black, Magnus Carlsson,
Mark Jones, Dick Kieburtz,
Johan Nordlander

OGI SCHOOL OF SCIENCE & ENGINEERING
OREGON HEALTH & SCIENCE UNIVERSITY

Timber objectives:

- Design a language with **explicit time** behavior
- Explore **reactivity** as the basic programming model
- Combine the power of a **functional** language with those of mutable objects
- Support **static** timing analysis as well as dynamic **adaptivity**

Our starting point

- Experience with *real-rate* applications
  – Streaming video over various networks
- Belief that applications do not *control* the world, but must *react* to it
- View of real-time systems as those in which
  – events occur in the environment, but not more quickly that t_l
  – application must react to those events within t_r
  – concurrency exists in both events and reactions

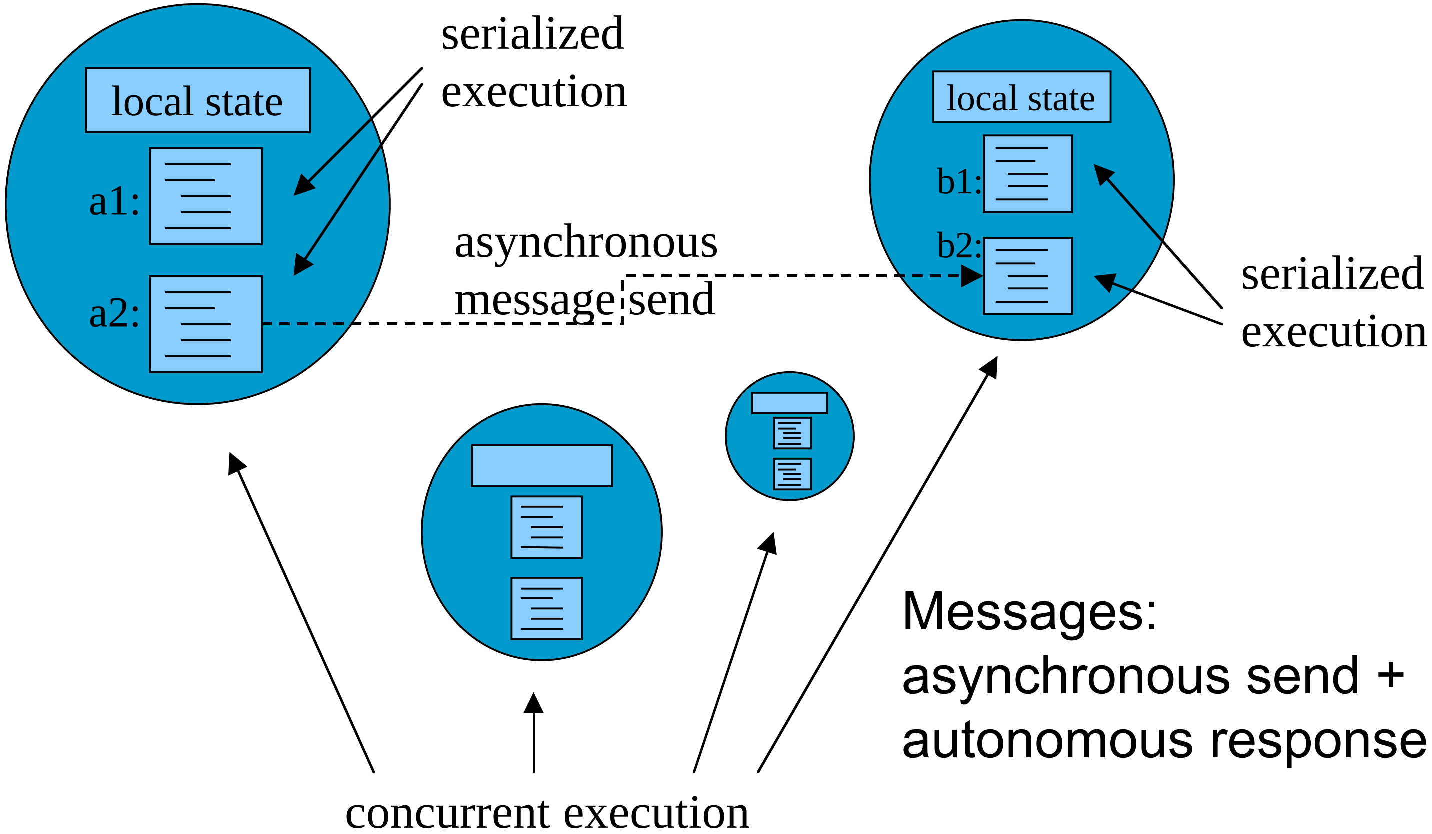
Where we did ***not*** start

- Threads
- Priorities & Scheduling Algorithms
- Communication & Synchronization Primitives
- Real-time Java
- Real-time Middleware

These *may* be appropriate solutions
They are *not* part of the problem statement

Concurrency and Objects

Object: encapsulated, mutable state + identity



Messages:
asynchronous send +
autonomous response

Reactivity

- Event = method invocation = message send
  - Output event: sending a message (*f a*)
  - Input event: being invoked (*x*–>*e*)
- No active input ~~(*x* = *e*; *e*')~~
- Method execution = reaction = non-blocking code sequence
- Objects alternate between transient activity and indefinite periods of rest

Update local state / create new objects / send messages

Blocking input considered harmful

- Blocking message send (or procedure call) is the wrong way to get input
  - Program has to choose which message to send
  - This represents a premature commitment
    - Order of external events *not* under program control!
    - Events are missed, or reordered!
- Event loop using a *select*
  - Helps only if all events of interest are encoded uniformly and posted to a single port

Commands and Expressions

Execute a command ≠ evaluate an expression

$c :: Cmd\ Int$ $e :: Int$

- This "monadic semantics" is taken from the language *Haskell*
- *Cmd* replaces Haskell's *IO* monad
  - There are no "I" operations!
- Object creation, message send, & state update are all *Cmd*s

Timber features

- Object templates used to define objects:

```
object init-state
in interface
```

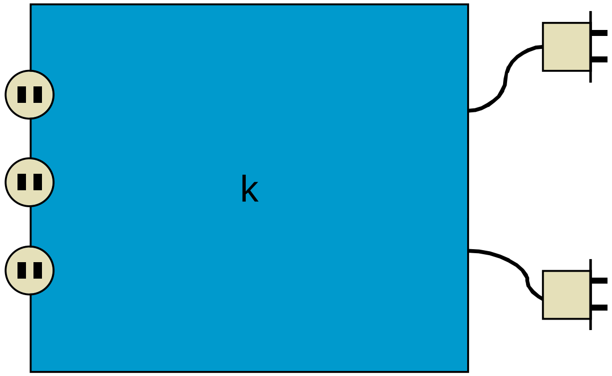
- An object's interface is usually a record of methods

```
name = action cmd-sequence
name = request cmd-sequence
               return expression
```

- Local state update (*instance-var := expr*)
- Subtyping by declaration (*Action < Cmd( )*)

Reactive components

```
record In =
  i1 :: Action
  i2 :: Int -> Action
  i3 :: Action
```

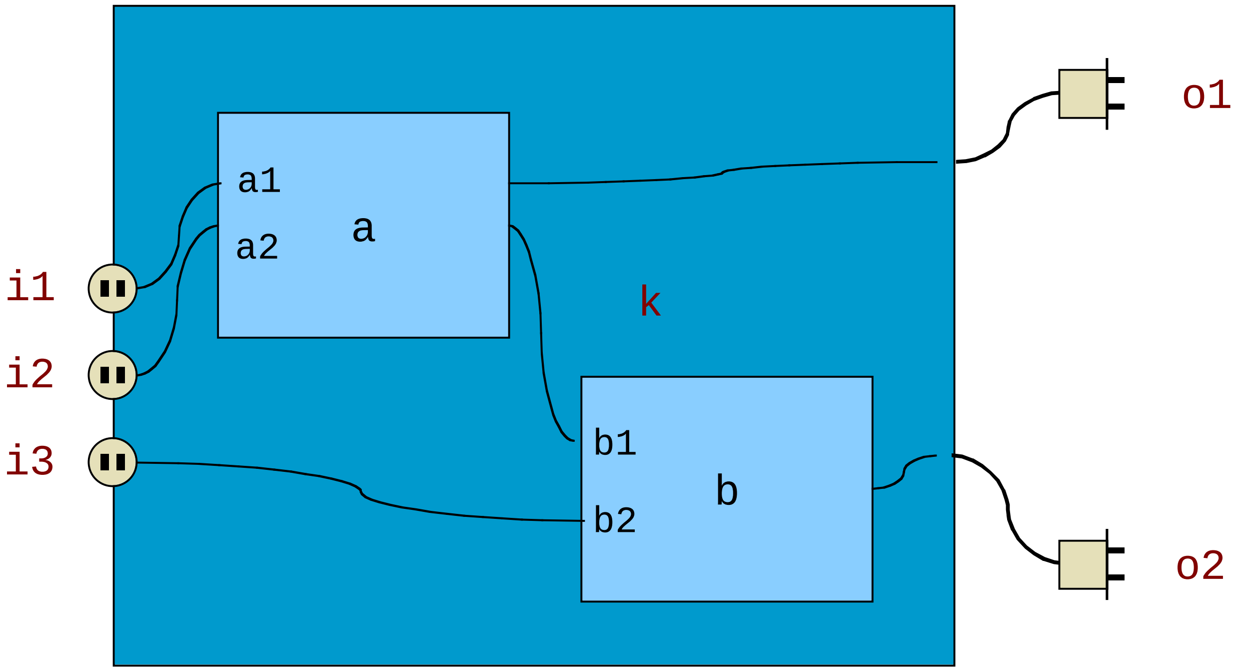



```
record Out =
  o1 :: T -> Action
  o2 :: Action
```

```
k :: Out -> Object In
```

Given an `Out` *interface,* `k` *is a template for objects that exhibit the* `In` *interface*

Composing objects



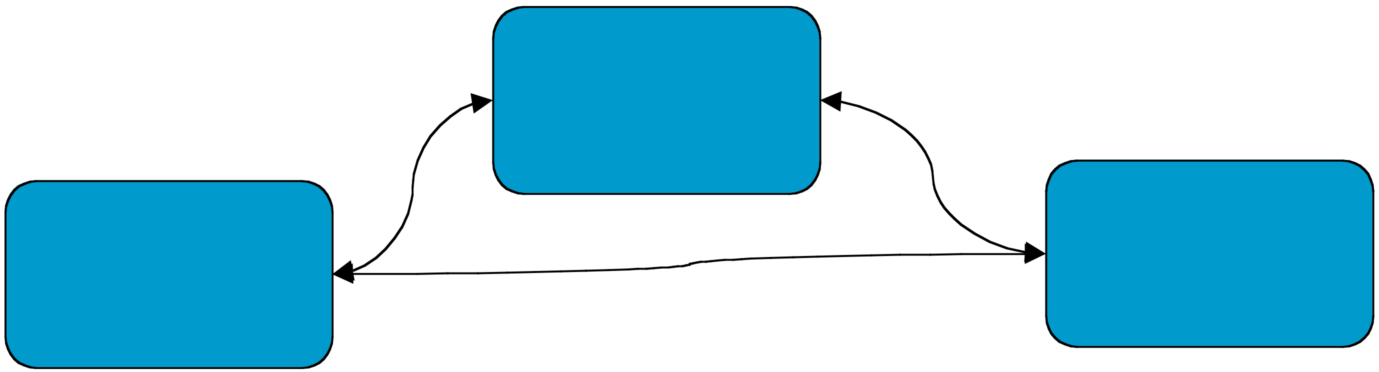

```
make_k out = do   b <- make_b out.o2
                  a <- make_a out.o1 b.b1
                  return record i1 = a.a1; i2 = a.a2; i3 = b.b2
```

Distribution

No location transparency!

– Latencies are important

– Failures may be partial



Within each local world:

- failure mean *total* failures
- message delivery is guaranteed and order-preserving
- a *main* action starts execution

Between local worlds:

- nodes may come up and go down
- message delivery depends on network protocol
- network must be modeled as a component in its own right

Controlling timing

- Each event has a *timeline*, an interval from a *baseline* … to a... *deadline*

- Default timeline is same as that of sender
- Can also be set explicitly:

after (10*seconds) *m*	defers baseline
before (25*milliseconds) *m*	sets deadline

Key idea: code can be *time-dependent*, yet *platform-independent*. Static analysis determines feasibility.

Accessing the timeline

- Built-in constants *baseline* and *deadline*
  - defined only within methods
  - provide access to the baseline and the deadline for the current method execution
- For methods initiated by the environment, timeline must be defined appropriately
  - *e.g.*, for an interrupt, baseline might be time at which the hardware event occurred
  - deadline might be time within which registers must be read

Example: Ping Program

```
-> hosts = ["dogbert", "ratbert", "ratberg”, "theboss"]

-> ping hosts (Port 515)

dogbert: lookup & connect after 20.018 ms
ratbert: lookup & connect after 41.432 ms
ratberg: NetError "Host name lookup failure" after 70.282 ms
theboss: no response within 2 s
```

```
ping hosts port env =
   object
      outstanding := hosts
   in let
      client host start peer =
         record
            connect    = action
               env.putStrLn(host  ++ ": lookup & connect after "
                                  ++ show (baseline-start))
               outstanding := remove host outstanding
               peer.close
            deliver  _ = action done
            neterror e = action
               env.putStrLn(host  ++ ": " ++ show e ++ " after "
                                  ++ show (baseline-start))
               outstanding := remove host outstanding
            close      = action done
      cleanup = action
         forall h <- outstanding do
            env.putStrLn(h ++ ": no response within ″ ++ show timeout)
         env.quit
      timeout = 2*seconds
   in record
      main = action
         forall h <- hosts do
            env.inet.tcp.open h port (client h baseline)
         after timeout cleanup
```

Java Version

http://java.sun.com/j2se/1.4/docs/guide/nio/example/Ping.java

```
Boulder:Users:black:Andrew  Black:talks
Wednesday, 5 December 2001

import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
import java.util.*;
import java.util.regex.*;


public class Ping {

    static int DAYTIME_PORT = 13;

    static int port = DAYTIME_PORT;


    static class Target {

    InetSocketAddress address;
    SocketChannel channel;
    Exception failure;
    long connectStart;
    long connectFinish = 0;
    boolean shown = false;

    Target(String host) {
        try {
        address = new InetSocketAddress(Inet/
                port);
        } catch (IOException x) {
        failure = x;
        }
    }

    void show() {
        String result;
        if (connectFinish != 0)
        result = Long.toString(connectFinish
        else if (failure != null)
        result = failure.toString();
        else
        result = "Timed out";
        System.out.println(address + " : " +
        shown = true;
    }

    }


    static class Printer
    extends Thread
    {
    LinkedList pending = new LinkedList();

    Printer() {
        setName("Printer");
        setDaemon(true);
    }

    void add(Target t) {
        synchronized (pending) {
```

```
Boulder:Users:black:Andrew  Black:talks:Timber:Ping.java          Page 2 of 4
Wednesday, 5 December 2001                                   Printed:  17:16:00

        pending.add(t);
        pending.notify();
        }
    }

    public void run() {
        for (;;) {
        Target t = null;
        synchronized (pending) {
            try {
            pending.wait();
            } catch (InterruptedException x) {
            return;
            }
            while (pending.size() > 0) {
            t = (Target)pending.removeFirst();
            }
        }
        t.show();
        }
    }
    }


    static class Connector
    extends Thread
    {
    Selector sel;
    Printer printer;

    LinkedList pending = new LinkedList();

    Connector(Printer pr) throws IOException {
        printer = pr;
        sel = Selector.open();
        setName("Connector");
    }

    void add(Target t) {
        SocketChannel sc = null;
        try {

        sc = SocketChannel.open();
        sc.configureBlocking(false);
        sc.connect(t.address);

        t.channel = sc;
        t.connectStart = System.currentTimeMillis()

        synchronized (pending) {
            pending.add(t);
        }

        sel.wakeup();

        } catch (IOException x) {
        if (sc != null) {
            try {
            sc.close();
            } catch (IOException xx) { }
        }
        t.failure = x;
```

```
Boulder:Users:black:Andrew  Black:talks:Timber:Ping.java
Wednesday, 5 December 2001

        printer.add(t);
        }
    }

    void processPendingTargets() throws IOException {
        synchronized (pending) {
        while (pending.size() > 0) {
            Target t = (Target)pending.removeFirst();

            SelectionKey sk;
            try {
            sk = t.channel.register(sel,
                        SelectionKey.OP_CONNECT);
            } catch (IOException x) {
            t.channel.close();
            t.failure = x;
            printer.add(t);
            continue;
            }

            sk.attach(t);

        }
        }
    }

    void processSelectedKeys() throws IOException {
        for (Iterator i = sel.selectedKeys().iterator(); i.hasNext();

        SelectionKey sk = (SelectionKey)i.next();
        i.remove();

        Target t = (Target)sk.attachment();
        SocketChannel sc = (SocketChannel)sk.channel();

        try {
            if (sc.finishConnect()) {
            sk.cancel();
            t.connectFinish = System.currentTimeMillis();
            sc.close();
            printer.add(t);
            }
        } catch (IOException x) {
            sc.close();
            t.failure = x;
            printer.add(t);
        }
        }
    }

    volatile boolean shutdown = false;

    void shutdown() {
        shutdown = true;
        sel.wakeup();
    }

    public void run() {
        for (;;) {
        try {
            int n = sel.select();
            if (n > 0)
```

```
Boulder:Users:black:Andrew  Black:talks:Timber:Ping.java
Wednesday, 5 December 2001                                          Pri

            processSelectedKeys();
            processPendingTargets();
            if (shutdown) {
            sel.close();
            return;
            }
        } catch (IOException x) {
            x.printStackTrace();
        }
        }
    }

    }


    public static void main(String[] args)
    throws InterruptedException, IOException
    {
    if (args.length < 1) {
        System.err.println("Usage: java Ping [port] host...");
        return;
    }
    int firstArg = 0;

    if (Pattern.matches("[0-9]+", args[0])) {
        port = Integer.parseInt(args[0]);
        firstArg = 1;
    }

    Printer printer = new Printer();
    printer.start();
    Connector connector = new Connector(printer);
    connector.start();

    LinkedList targets = new LinkedList();
    for (int i = firstArg; i < args.length; i++) {
        Target t = new Target(args[i]);
        targets.add(t);
        connector.add(t);
    }

    Thread.sleep(2000);
    connector.shutdown();
    connector.join();

    for (Iterator i = targets.iterator(); i.hasNext();) {
        Target t = (Target)i.next();
        if (!t.shown)
        t.show();
    }

    }

}
```

OGI SCHOOL OF SCIENCE & ENGINEERING
OREGON HEALTH & SCIENCE UNIVERSITY

Comparison

- Timber version
  - all actions are defined inside *ping* object
    - can safely manipulate *outstanding* in mutual exclusion
  - solution is straightforward:
    - one object, one instance variable
- Java version
  - 10 class variables
  - 3 threads
    - timeout, printing, de-multiplex of connection events
  - Less concurrency (*gethostbyname* bug!)