JCSP Networking 2.0
(or maybe JCSP 1.1 rc4)

Kevin Chalmers
School of Computing
Napier University
Aims

• Update to JCSP 1.1
  – Poisonable network channels
  – Remove pesky rejectable channels
  – Extended rendezvous
  – No networked AltingBarrier (yet!)

• Reduce overheads
  – No process per channel
  – No LoopbackLink
  – LinkManager now a passive data object
  – Smaller message size
Aims

• Extensibility, configurability and error handling
  – Layered model – easier to add extensions
  – NetworkBarrier!
  – Better NetworkConnection (soon)
  – All networked channels mobile (maybe)
  – Priority of communication layer
  – Buffer size
  – Quick creation of channels (no Channel Name Server required)
  – JCSPNetworkException

• Interaction
  – Towards a universal protocol
JCSP 2.0: Generics, NIO
JCSP: No Object Serialization, limited resources
JCSPme: No Object Serialization, very limited resources

KRoC & pony: What’s an Object?
PyCSP: What’s Java?
CSP.NET (Jibu): What’s Java?
C++CSP: What’s Java?
Towards a Universal Protocol

• Messages are no longer objects
  – SEND | Destination | Source | <data>
  – <1, 0, 0, 0, 54, 0, 0, 0, 49, <data>>

• Data encoding and decoding handled at channel level
  – User defined methods possible
  – Object serialization default, raw data and class loading provided
Layered Model

Application Process

Net Channel Input

Net Channel Output

Channel Manager

Link RX

Link TX

Link Manager

Link Server

Communication Mechanism
Creating an Application

- Old way
- Use Channel Name Server
  - Can use names – implies lookup on receiving Node

Node.getInstance().init(new TCPIPNodeFactory("CNS_IP"));
NetChannelInput in = CNS.createNet2One("channel_In");
NetChannelOutput out = CNS.createOne2Net("channel_Out");
Creating an Application

• New way

```java
Node.getInstance().init(new TCPIPNodeAddress(5000));
// Create Link to remote Node
TCPIPNodeAddress remoteAddr = new TCPIPNodeAddress("192.168.1.100", 4000);
// Get NodeID
NodeID remoteNode = LinkFactory.getLink(remoteAddr).getRemoteNodeID();
// Create channels
NetChannelInput in = NetChannel.numberedNet2One(55);
NetChannelOutput out = NetChannel.one2net(remoteNode, 49);
```

• Other methods possible
  – Original method
  – From NodeAddress and VCN
  – From NetChannelLocation
Other Channel Options

• Poison
NetChannelInput in = NetChannel.net2one(10);
NetChannelOutput out = NetChannel.one2net(location, 10);

• Specified encoder / decoder
NetChannelInput in =
    NetChannel.net2one(new RawNetworkMessageFilter.FilterRX());
NetChannelOutput out = NetChannel.one2net
    (location, new RawNetworkMessageFilter.FilterTX());
NetChannelInput in =
    NetChannel.net2one(new CodeLoadingChannelFilter.FilterRX());
NetChannelOutput out = NetChannel.one2net
    (location, new CodeLoadingChannelFilter.FilterTX());
NetworkBarrier

- Two tier approach
  - Declaring (server) end
  - Multiple connecting (client) ends
  - Each end has \( n \) enrolled processes
  - Server end **MUST** have one enrolled process
NetworkBarrier

• Creation methods – as channels
  – Barrier Name Server (BNS)
    ```java
    NetBarrier servBar = BNS.netBarrier("barrier", 10, 10);
    NetBarrier clientBar = BNS.netBarrier("barrier", 10);
    ```
  – Numbered barrier ends
    ```java
    NetBarrier servBar =
        NetBarrierEnd.numberedNetBarrier(55, 10, 10);
    NetBarrier clientBar =
        NetBarrierEnd.netBarrier(nodeID, 55, 10);
    ```

• Server end declares both locally enrolled and expected remote client ends.
Error Handling

- Channels can throw JCSPNetworkException or NetworkPoisonException
  - Unchecked exceptions – no need to explicitly catch
  - If connection to input end fails, the output end will throw a JCSPNetworkException
  - If there is a problem during I/O (including encoding / decoding) a channel will throw a JCSPNetworkException
  - If the input end is destroyed, the output end will throw a JCSPNetworkException during next write operation
  - If a message is sent to an input channel that does not exist, a JCSPNetworkException will be thrown
  - If a channel end is poisoned with sufficient strength, every complement end will throw a NetworkPoisonException
Error Handling

• Barriers can only throw JCSPNetworkException
  – If the connection to the server NetBarrier fails, a client NetBarrier will throw a JCSPNetworkException and fail.
  – If the connection to a client NetBarrier fails, a server end will throw a JCSPNetworkException, decrement the enrolled network process count, and allow reuse if required.
  – If a client end tries to enrol on a non-existent server end, a JCSPNetworkException will be thrown.
  – If the locally enrolled count on the server end reaches zero, a JCSPNetworkException will be thrown.
Mobility

• Non-running process mobility via code mobility
  – Code loading channel filter
  – Reduced model from last year's paper

• Running processes still require termination
  – Poison
  – Migration event

• Channel mobility via message boxes
  – Updated model soon....
  – Built into protocol?
Wrapping up

- New JCSP networking available on the JCSP repository (under the Networking-2 branch)
- More information and examples given in handouts
  - Set up
  - Channel creation, operations and error handling
  - Custom encoders and decoders
  - Network barriers
  - Mobility
  - Custom Link protocol creation
Wrapping up

• Hopefully everyone’s existing programs will still work
  – Same interfaces
  – Some packages not replicated (dynamic, remote, security, settings)

• More updates soon, once I’m finished writing up
  – NetConnections
  – Better channel mobility
  – AltingBarrier?

• Any requests for functionality / information let me know, and I’ll try and help as much as I can.
Questions?