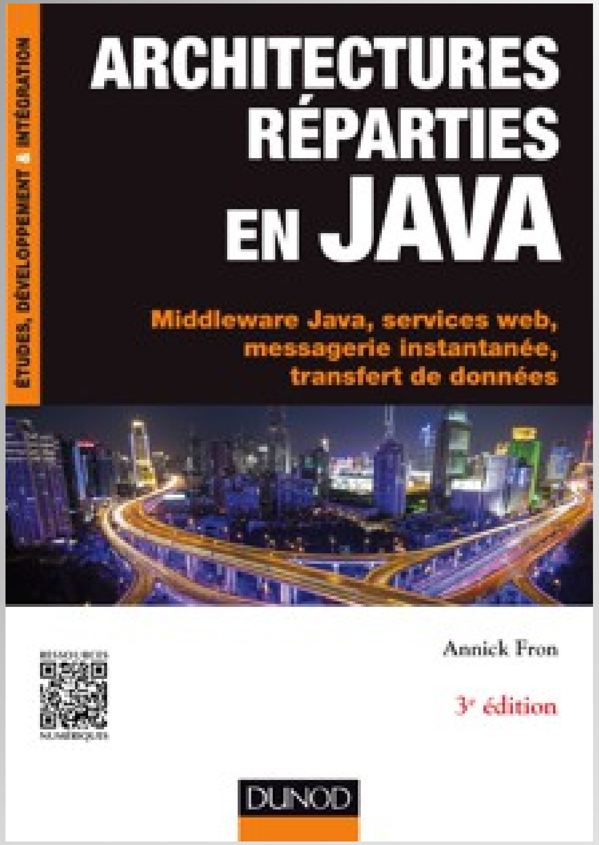
RPC in Smalltalk

Annick Fron
http://www.afceurope.com
http://www.fencingfox.com

Distributed applications


ARCHITECTURES RÉPARTIES EN JAVA
Middleware Java, services web, messagerie instantanée, transfert de données
ÉTUDES, DÉVELOPPEMENT & INTÉGRATION
Annick Fron
3e édition
DUNOD

Dunod 2015, 3rd edition

Java Distributed Architecture

Software consultant, not academic

8 fallacies in Distributed Computing (Peter Deutsch 1994)

- The network is reliable : redundancy, intermediate storage
- Latency is zero : 30s for light between US and Europe ; latency using Ajax
- Bandwidth is infinite : packets are limited in size
- The network is secure : must understand firewalls, passwords,etc.
- Topology doesn't change : endpoints, alias, abstract naming, host names...
- There is one administrator : monitoring, interoperability contracts
- Transport cost is zero : routers, servers...
- The network is homogeneous : try to stick to standards

The Smurfs

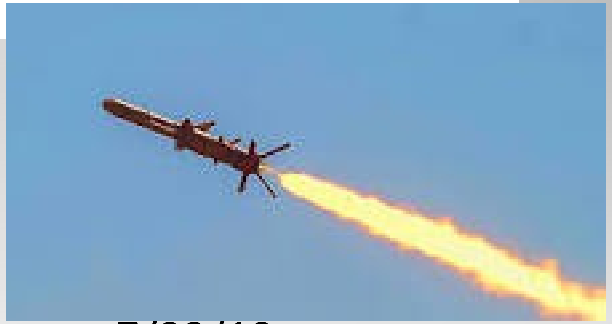
HTTP,

Cloud,

Relational

database

RPC is hard : all weapons welcome!



Sockets : don't understand objects !

- UDP
- Allows broadcast
- Limited size messages
- Faster (no handshake)
- Used by video

- TCP
- More "reliable" because of handshake... but less tolerant to network disconnection
- Used by HTTP

Swiss Army knife for ANY language !

RPC ?

- Remote Procedure Call
- Remote events
- Data sharing
- Streaming and web sockets
- Remote notification
- And then non functional properties : redundancy, security, reliability, resource pooling...

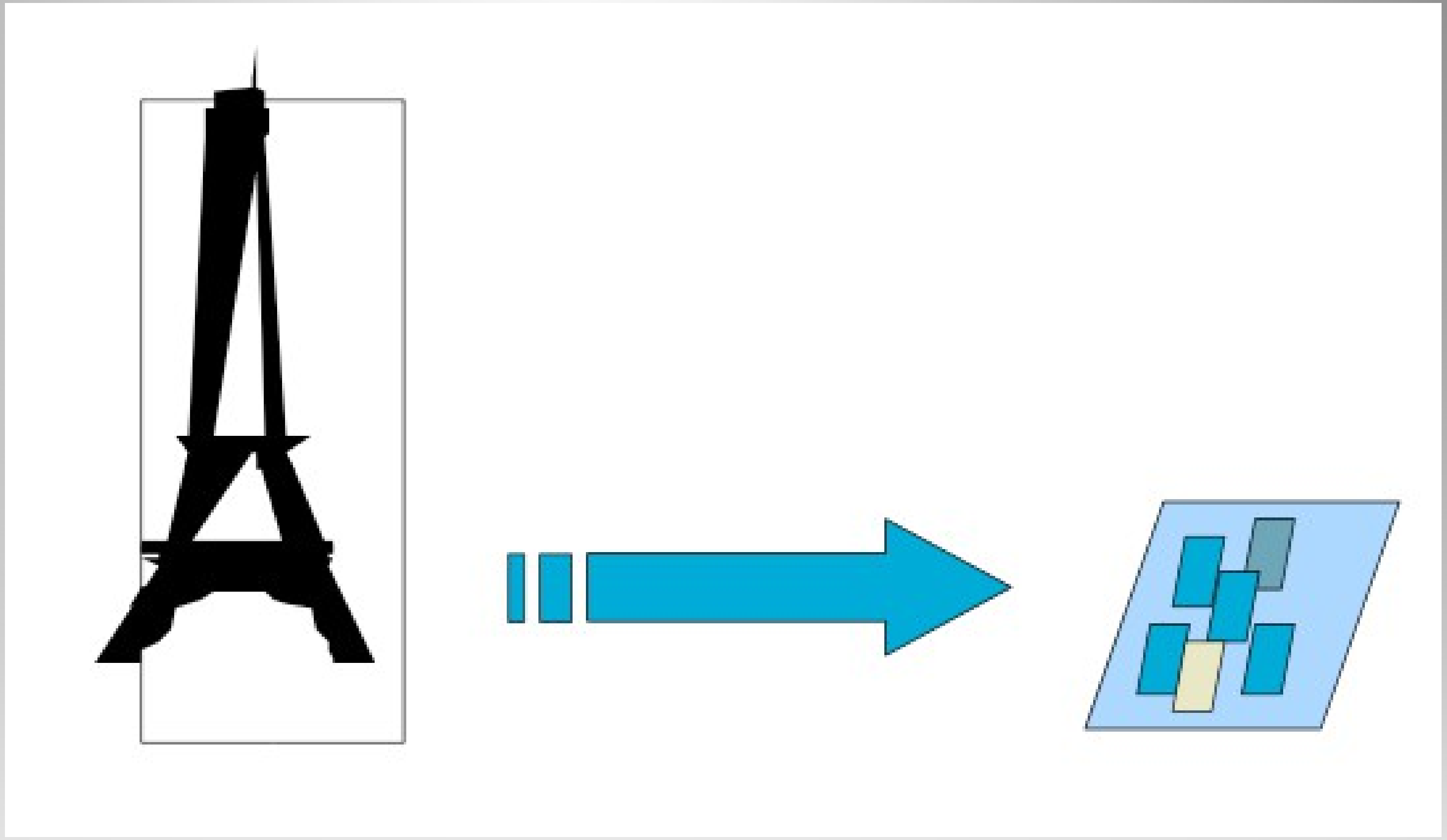
RPC

- Contact point : URL, registry key, endpoint(web service);
  - extensions to pool of workers or security policies
- Transport : TCP, UDP, HTTP, MQ ...
- Marshalling/unmarshalling

Marshalling/unmarshalling

Sockets are flat

- ASN1, CDR
- JSON
- XML
- Fuel, SIXX, BOSS
- Object references
-



Naive ideas

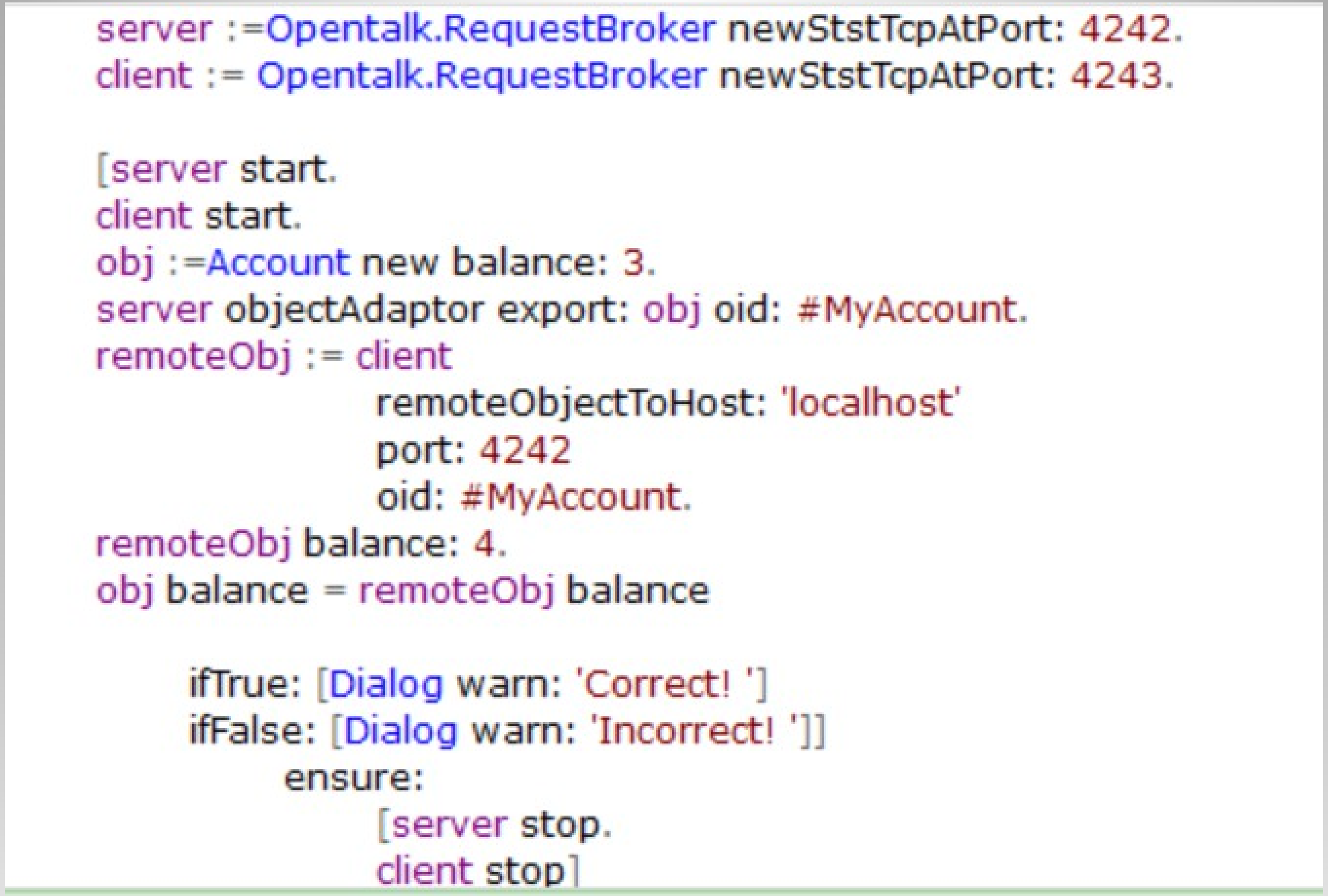
- Using doesNotUnderstand for proxies to modify behaviour
- An object is always able to perform a selector passed as a string

Passing by reference / by value

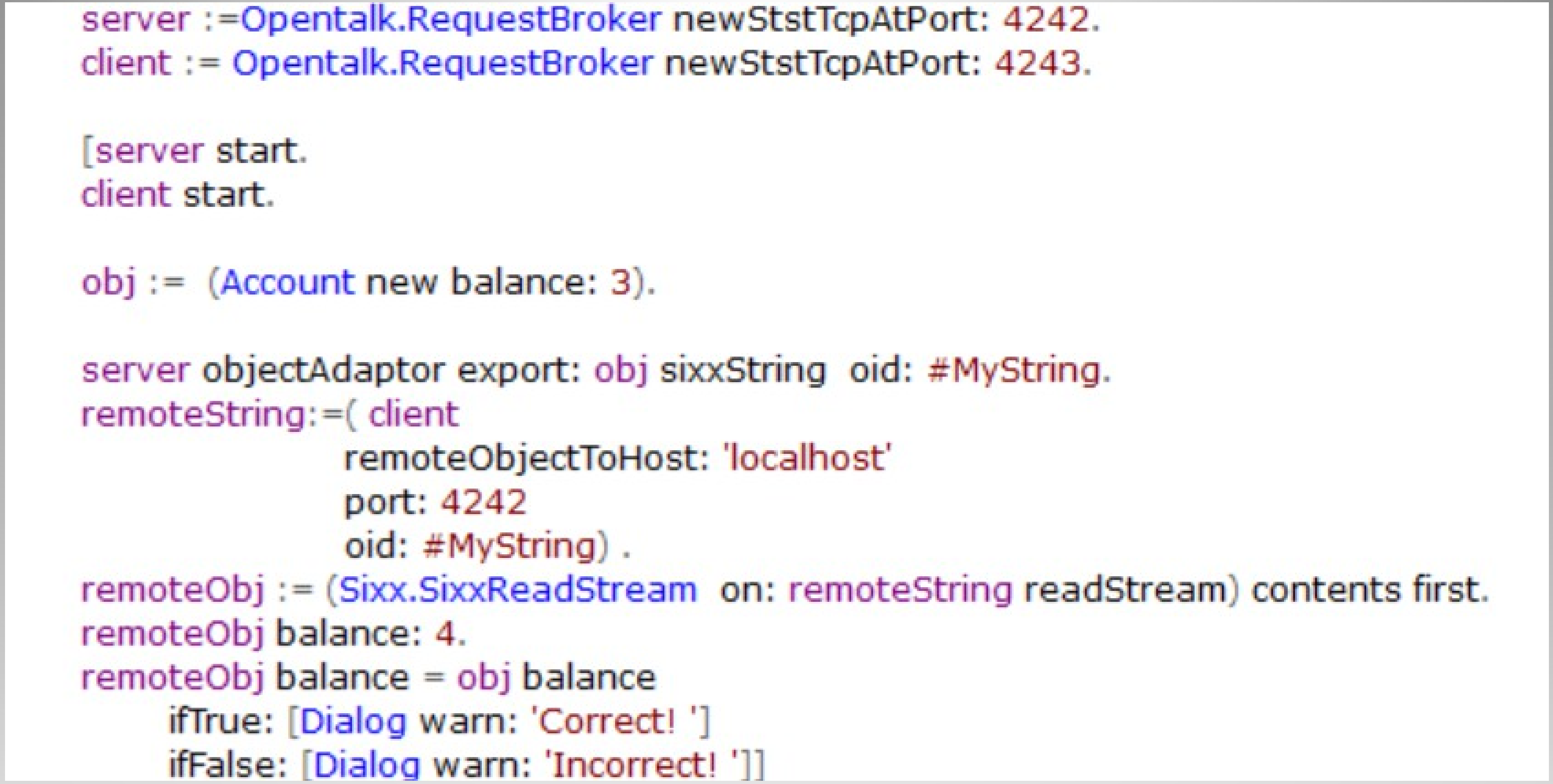
- Reference
- Easy for large objects
- May imply handling distributed GC
- Client and server in sync
- Ping pong effect for nested objects
- Implies a registry to one or more root references

- Value
- Client may work independently, but looses sync on server
- Large chunks being passed on the network

Démo VW



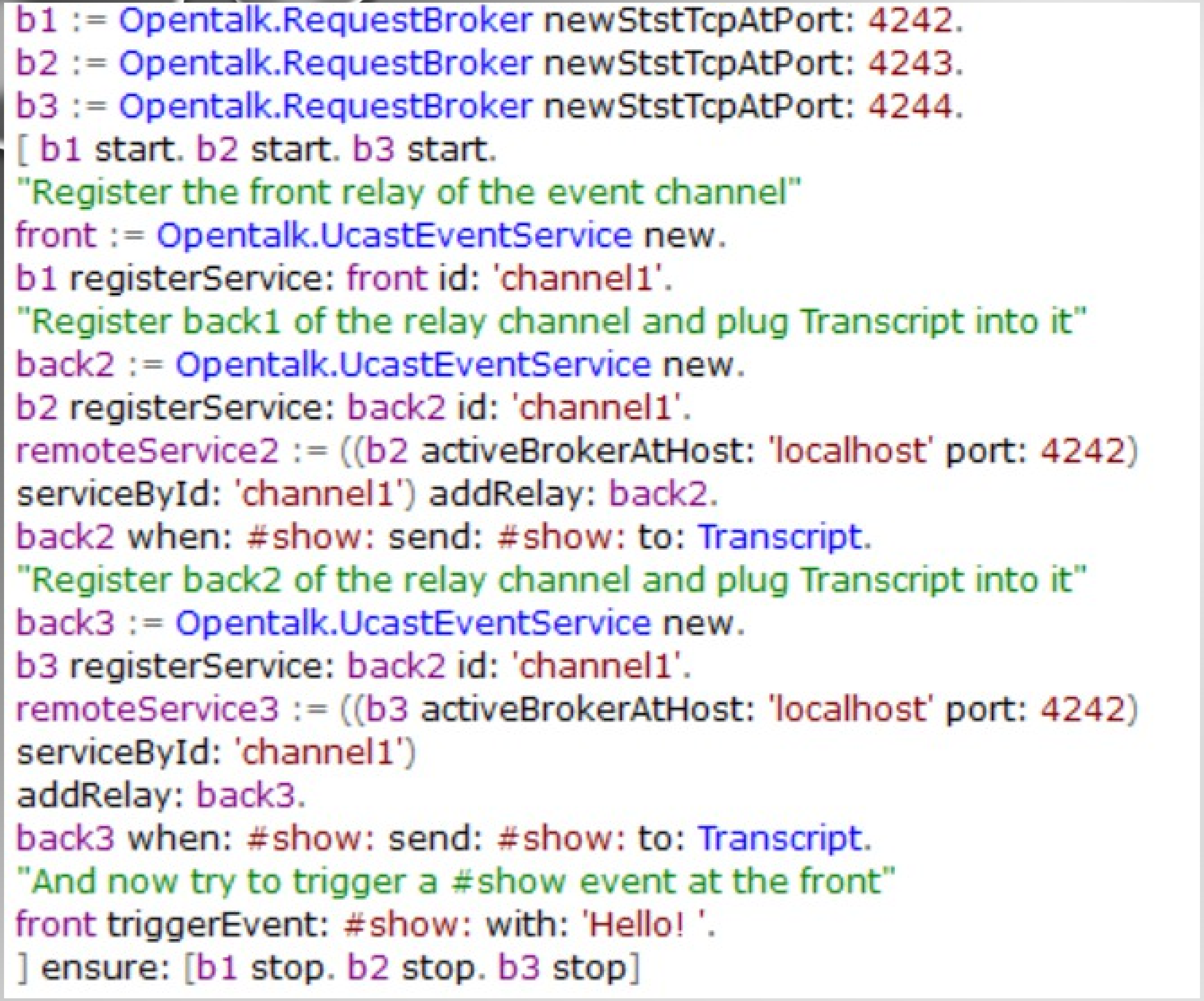
```
server :=Opentalk.RequestBroker newStstTcpAtPort: 4242.
client := Opentalk.RequestBroker newStstTcpAtPort: 4243.

[server start.
client start.
obj :=Account new balance: 3.
server objectAdaptor export: obj oid: #MyAccount.
remoteObj := client
            remoteObjectToHost: 'localhost'
            port: 4242
            oid: #MyAccount.
remoteObj balance: 4.
obj balance = remoteObj balance

    ifTrue: [Dialog warn: 'Correct! ']
    ifFalse: [Dialog warn: 'Incorrect! ']]
        ensure:
            [server stop.
            client stop]
```



```
server :=Opentalk.RequestBroker newStstTcpAtPort: 4242.
client := Opentalk.RequestBroker newStstTcpAtPort: 4243.

[server start.
client start.

obj :=  (Account new balance: 3).

server objectAdaptor export: obj sixxString  oid: #MyString.
remoteString:=( client
                    remoteObjectToHost: 'localhost'
                    port: 4242
                    oid: #MyString) .
remoteObj := (Sixx.SixxReadStream  on: remoteString readStream) contents first.
remoteObj balance: 4.
remoteObj balance = obj balance
      ifTrue: [Dialog warn: 'Correct! ']
      ifFalse: [Dialog warn: 'Incorrect! ']]
```

Some questions

- Passing classes ? Passed by name. But a class in one instance is recognized as a class in the other instance. This is not true in other languages Corba on C requires mapping of vtables). In Java a class in a class loader is different from the same class in another class loader, or in another memory space. Strong typing is a looser across memory spaces
- Passing errors ? Errors raise errors on the distant image
- Instvars can have individual passing policies (or none)



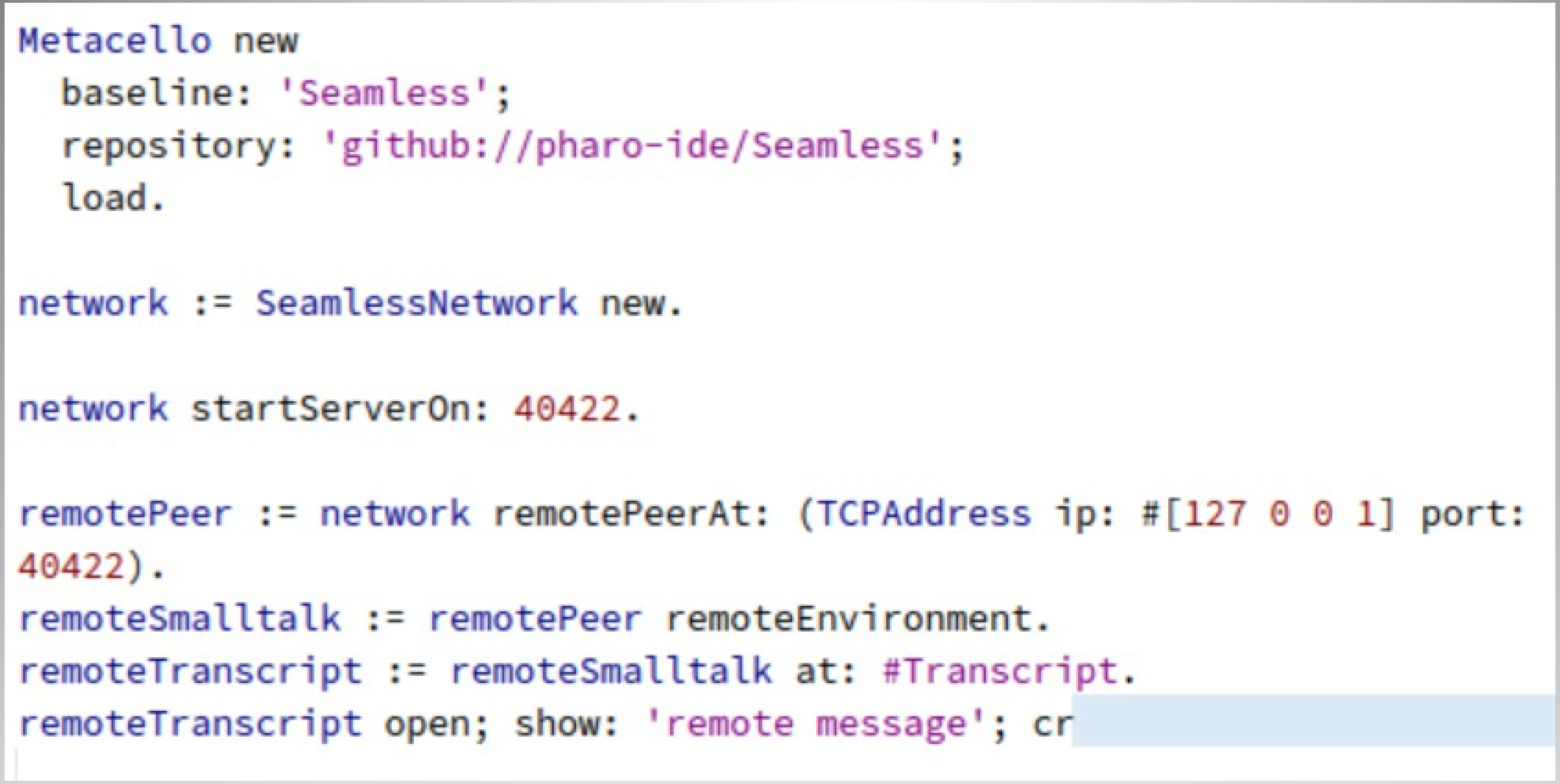
```
b1 := Opentalk.RequestBroker newStstTcpAtPort: 4242.
b2 := Opentalk.RequestBroker newStstTcpAtPort: 4243.
b3 := Opentalk.RequestBroker newStstTcpAtPort: 4244.
[ b1 start. b2 start. b3 start.
"Register the front relay of the event channel"
front := Opentalk.UcastEventService new.
b1 registerService: front id: 'channel1'.
"Register back1 of the relay channel and plug Transcript into it"
back2 := Opentalk.UcastEventService new.
b2 registerService: back2 id: 'channel1'.
remoteService2 := ((b2 activeBrokerAtHost: 'localhost' port: 4242)
serviceById: 'channel1') addRelay: back2.
back2 when: #show: send: #show: to: Transcript.
"Register back2 of the relay channel and plug Transcript into it"
back3 := Opentalk.UcastEventService new.
b3 registerService: back2 id: 'channel1'.
remoteService3 := ((b3 activeBrokerAtHost: 'localhost' port: 4242)
serviceById: 'channel1')
addRelay: back3.
back3 when: #show: send: #show: to: Transcript.
"And now try to trigger a #show event at the front"
front triggerEvent: #show: with: 'Hello! '.
] ensure: [b1 stop. b2 stop. b3 stop]
```

RPC in VW

- Pluggable transport : TCP, UDP
- Mapping to historical CORBA, IIOP : may call other languages like Java, C++ or C
- I3S provides transparent RPC with custom instvar policies (value, reference)
- Event service allows remote event notification

Seamless in Pharo

- Pass by reference or pass by value semantics
- Used for remote debug
- Initial reference : Whole environment

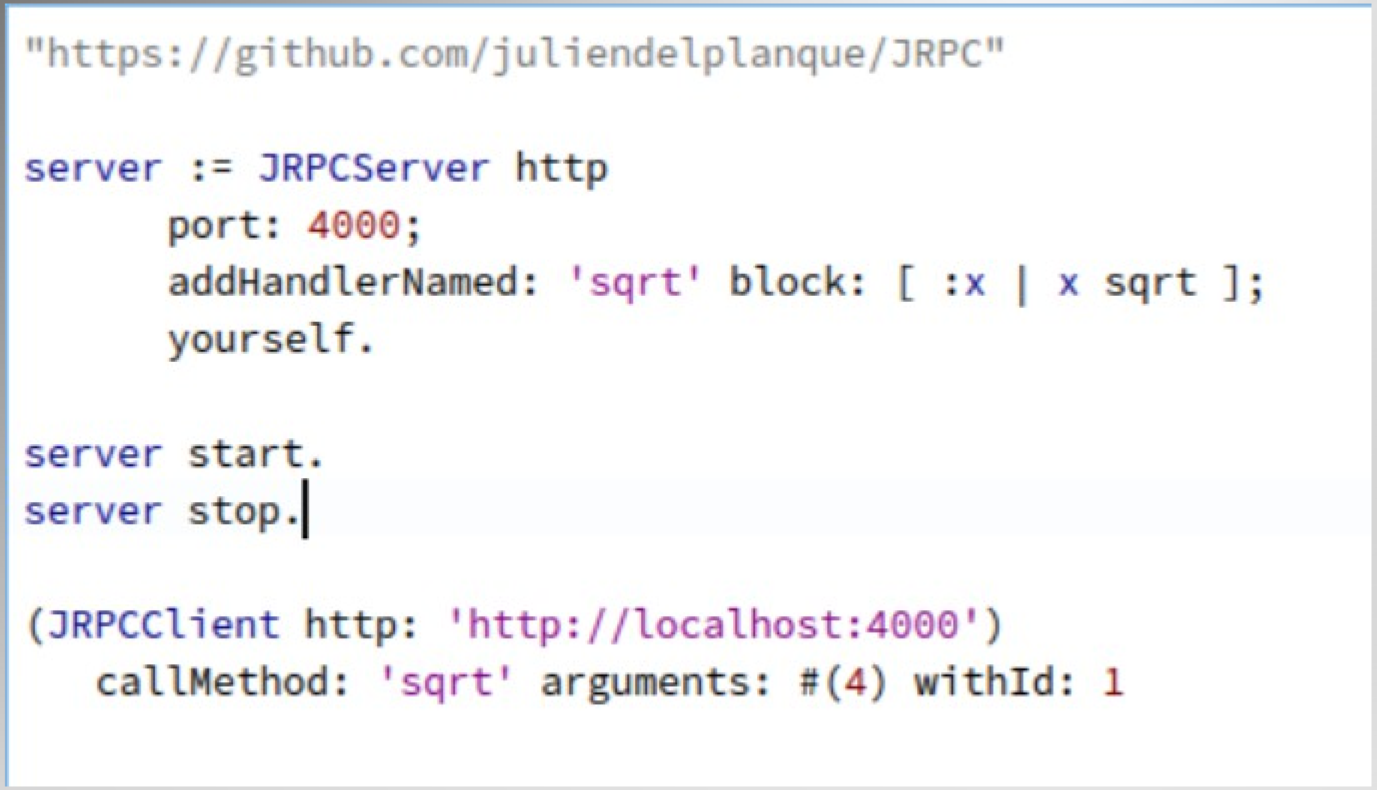


```
Metacello new
  baseline: 'Seamless';
  repository: 'github://pharo-ide/Seamless';
  load.

network := SeamlessNetwork new.

network startServerOn: 40422.

remotePeer := network remotePeerAt: (TCPAddress ip: #[127 0 0 1] port:
40422).
remoteSmalltalk := remotePeer remoteEnvironment.
remoteTranscript := remoteSmalltalk at: #Transcript.
remoteTranscript open; show: 'remote message'; cr
```

Both semantics supported

```
Object>>seamlessDefaultTransferStrategy
    ^SeamlessTransferStrategy defaultByReference

Number>>seamlessDefaultTransferStrategy
    ^SeamlessTransferStrategy defaultByValue
```

- JRPC : using JSON for marshalling



```
"https://github.com/juliendelplanque/JRPC"

server := JRPCServer http
      port: 4000;
      addHandlerNamed: 'sqrt' block: [ :x | x sqrt ];
      yourself.

server start.
server stop.

(JRPCClient http: 'http://localhost:4000')
   callMethod: 'sqrt' arguments: #(4) withId: 1
```

Preparing an image for remoting

- Identify system objects (CairoContext, files, processes …) which can't pass though
- Analyse the calling sequence to minimize ping pong : can require creating new objects which are “summaries” of some other objects, like views in a database
- Decide on pass by value/ pass by reference
- Always release ressources

Web sockets

Wraps http request with handler

```
ZnServer default delegate: (ZnWebSocketDelegate handler:
   [ :webSocket |
      [ | message |
         message := webSocket readMessage.
         webSocket sendMessage: message ] repeat ]).
```

Web sockets

- Same issues as sockets : strings and byte arrays, no objects
- One web socket per page needs to be parsed to different fields
- Handling disconnections

Web services

- Marshalling: XML
- Endpoint : URL
- WSDL : IDL
- Transport : HTTP
- Copy semantics
- SOAP Envelope for non functional aspects (?)

Using Gemstone

- May use Gemstone as a distributed shared memory
- Objects are shared in the images
- But notification of change needed to update interface : use of notifySets with the interface as a callback
- Use gemstone signalling between computers

Different notification mechanisms

- Using notifySets to update the GUI

```
session1 addToNotifySet: leTournoi competition poules first.

session1 notificationAction: [:idSet | callbacks do: ….].
```

- Using Gem to gem signalling for workflow

```
session gemSignalAction:

[:aSession :aSignalNumber :aString |

self handleSignalFrom: aSession number: aSignalNumber string: aString].
```

Conclusion

- Missing testing tools !
- MQTT, gRPC (Google)
- Streaming : mixing data and signals