QuickStart Apache Synapse:

Adding Service Mediation to your Network

Paul Fremantle, *pzf@apache.org*
Synapse Committer, ASF Member
VP of Technical Sales @ WSO2

Some things you should learn today

- How to add a virtualization layer to your SOAP and XML/HTTP communications
- How to enable and disable protocols like WSSecurity and WSReliableMessaging without writing any code or changing your SOAP stack
- How to add load-balancing and fail-over to your services
- A high-level view of Synapse performance and architecture
- Deployment options and approaches
- What is the Synapse config language and how can you use it
- How to extend Synapse to do more than out-of-the-box

Plan of Attack! – take cover

- Part 1
  - Synapse Overview, Getting Started, Deployment Approaches, Simple Routing Scenarios
- Part 2
  - Simple patterns
  - Content-based routing, transformations, headers, faults, filtering
  - Class mediators
- Part 3
  - Registry concept
  - Transport switching, JMS, WS-Security, WS-RM
  - Understanding the non-blocking HTTP transport

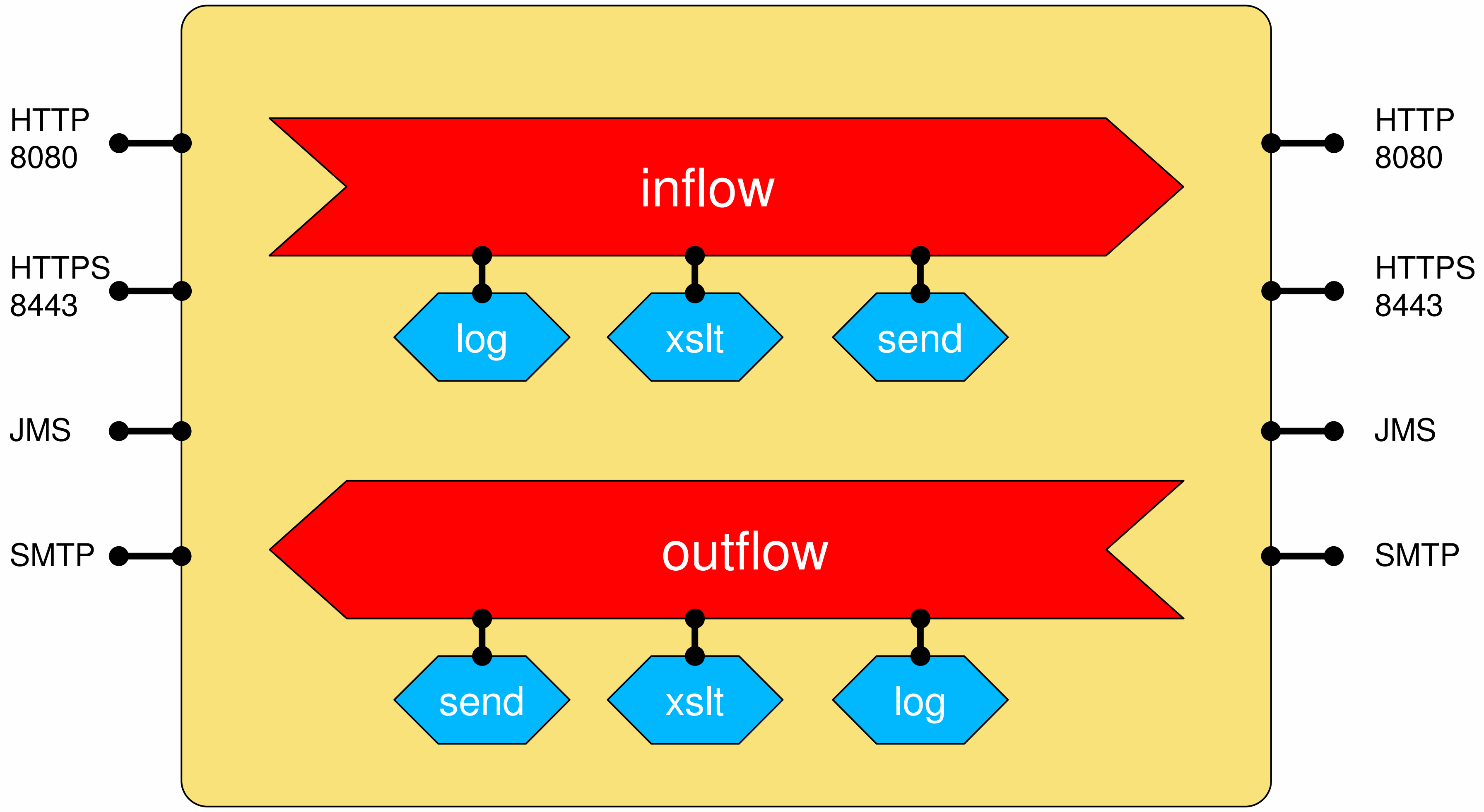
Plan of attack – Part 1

- What is Apache Synapse
- Overview of Service Mediation
- Installing Synapse
- Running Synapse
  - Demonstrating the proxy endpoints
- Deployment approaches
  - Synapse as an HTTP Proxy
- Using simple sequences

What is Apache Synapse?

- A lightweight **Enterprise Services Bus (ESB)**
  - Available as a WAR file, **NT Service, Linux Daemon**
  - Runs as a process with its own Listeners, Tasks and Senders
  - Can be deployed standalone or part of a cluster or distributed network
  - High performance, **asynchronous, streaming** design
  - Can initiate work – scheduled **tasks**
  - Supports multiple transports including HTTP, JMS, TCP, SMTP and (S)FTP
  - Simple to extend

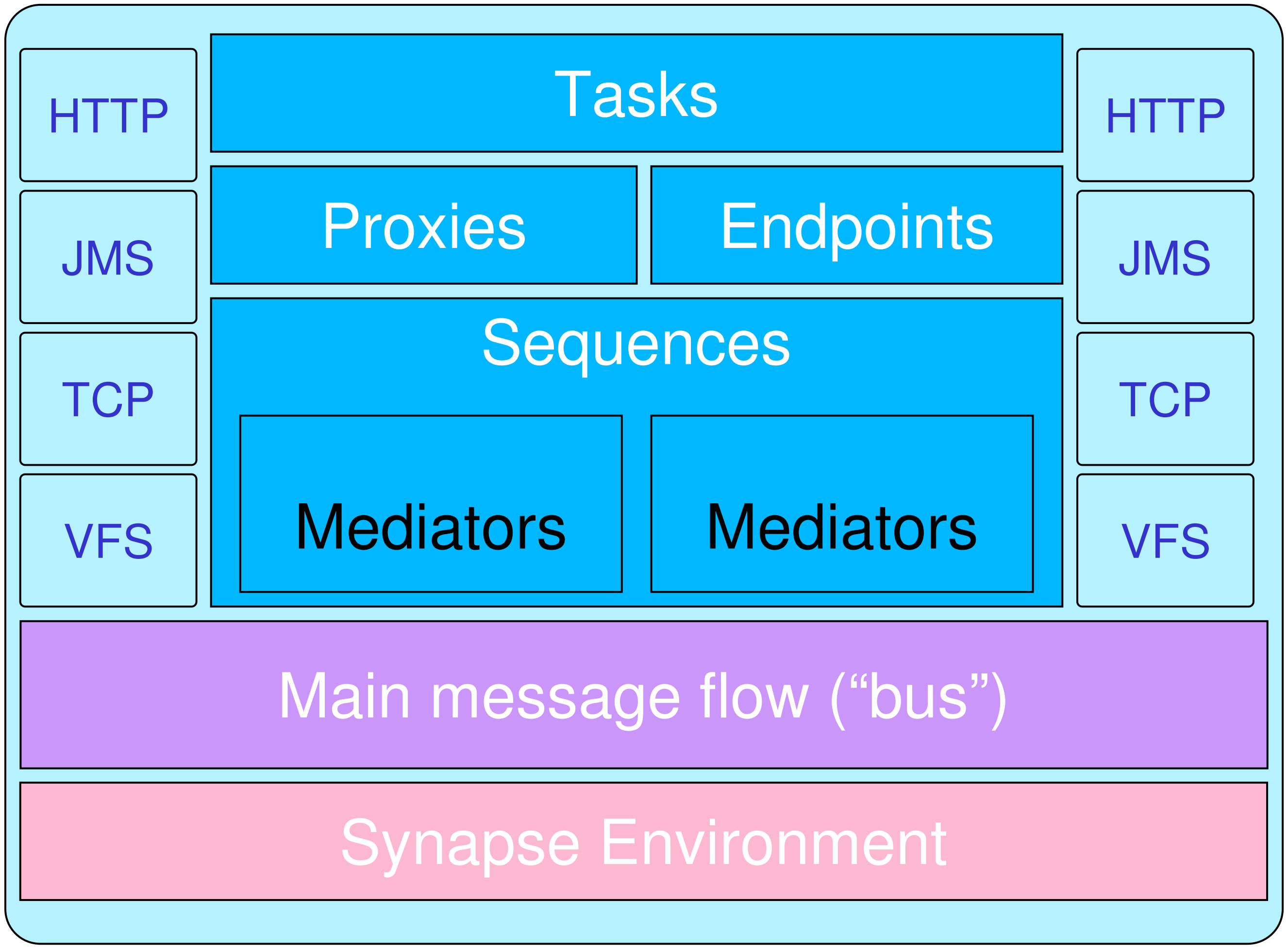
Flows


HTTP 8080
HTTPS 8443
JMS
SMTP
inflow
log
xslt
send
outflow
send
xslt
log
HTTP 8080
HTTPS 8443
JMS
SMTP

What does Synapse do?

- Transform
  - XSLT, XQuery, Java, Command Pattern, Script
- Route
  - URL-based, Content-based, Static (proxy pattern)
- Initiate
  - Scheduled tasks – repetitive or cron-based
- Manage
  - Logs, statistics, dynamic updates, validate, authorize

Apache Synapse graphically



Installing Synapse

- http://ws.apache.org/synapse/download.cgi
- Binary distributions:
  - synapse-1.1-bin.tar.gz
  - synapse-1.1-bin.zip
- Unzip/Untar to <PARENT> (e.g. c:\, ~/, etc)
- cd <PARENT>
- cd synapse-1.1
- bin\synapse, bin/synapse

You get it fresh off the press… 1.1 was released yesterday!

Alternative installations

- Once installed, you can
  - Windows (32/64bit Intel)
    - bin\install-synapse-service.bat
    - bin\run-synapse-service.bat or
    - net start “Apache Synapse”
  - Unix/Linux (32/64 Intel, Solaris 32 Intel, 32/64 Sparc)
    - sh bin/synapse-daemon.sh
- Or you can deploy the WAR file
  - Tomcat or other J2EE Application Server

Synapse startup

```
 Using Bouncy castle JAR for Java 1.5
Starting Synapse/Java ...
Using SYNAPSE_HOME:    C:\SYNAPS~1.1\bin\..
Using JAVA_HOME:       c:\jdk
Using SYNAPSE_XML:     -
    Dsynapse.xml="C:\SYNAPS~1.1\bin\..\repository\conf\synapse.xml"
2007-11-12 12:16:58,250 [-] [main]  INFO ServerManager Using the Axis2 Repository
    C:\SYNAPS~1.1\bin\..\repository
2007-11-12 12:17:01,921 [-] [main]  INFO SynapseInitializationModule Initializing
    Synapse at : Mon Nov 12 12:17:01 GMT 2007
2007-11-12 12:17:01,937 [127.0.0.1-pzfdell] [main]  INFO
    SynapseInitializationModule Loading mediator extensions...
2007-11-12 12:17:01,937 [127.0.0.1-pzfdell] [main]  INFO
    SynapseInitializationModule Initializing the Synapse configuration ...
2007-11-12 12:17:01,968 [127.0.0.1-pzfdell] [main]  INFO XMLConfigurationBuilder
    Generating the Synapse configuration model by parsing the XML configuration
   (some deleted)
2007-11-12 12:17:04,359 [127.0.0.1-pzfdell] [main]  INFO HttpCoreNIOSender HTTP
    Sender starting
2007-11-12 12:17:04,968 [127.0.0.1-pzfdell] [main]  INFO HttpCoreNIOListener HTTPS
    Listener starting on port : 8443
2007-11-12 12:17:04,968 [127.0.0.1-pzfdell] [main]  INFO ServerManager Starting
    transport https on port 8443
2007-11-12 12:17:05,046 [127.0.0.1-pzfdell] [main]  INFO ServerManager Ready for
    processing
```

Testing Synapse – **SMOKE TEST**

- To test Synapse you need to have some services running somewhere
- We thought of that!

1. cd <SYNAPSE>\samples\axis2Server
2. cd src\SimpleStockQuoteService
3. ant
   - Will build and deploy service
4. cd ..\..

   Make sure you have NO AXIS2_HOME set already!
5. Windows: SET AXIS2_HOME=
6. axis2server
   - Will start the server
   - Since Synapse already includes Axis2, we use the same Axis2 code to deploy the server

Server startup

```
Using JAVA_HOME    c:\jdk
Using AXIS2_HOME   C:\synapse-1.0-RC1-
   SNAPSHOT\samples\axis2Server\
[SimpleAxisServer] Using the Axis2 Repository :
   C:\synapse-1.0-RC1-
   SNAPSHOT\samples\axis2Server\repository
[SimpleAxisServer] Using the Axis2 Configuration File :
   C:\synapse-1.0-RC1-
   SNAPSHOT\samples\axis2Server\repository\conf\axis2.xml
[main] INFO  HttpCoreNIOSender - HTTPS Sender starting
[main] INFO  HttpCoreNIOSender - HTTP Sender starting
[main] INFO  HttpCoreNIOListener - HTTPS Listener starting
   on port : 9002
[main] INFO  HttpCoreNIOListener - HTTP Listener starting
   on port : 9000
[I/O reactor worker thread 5] INFO  PipeImpl - Using
   simulated buffered Pipes for event-driven to stream IO
   bridging
```

Now try the client

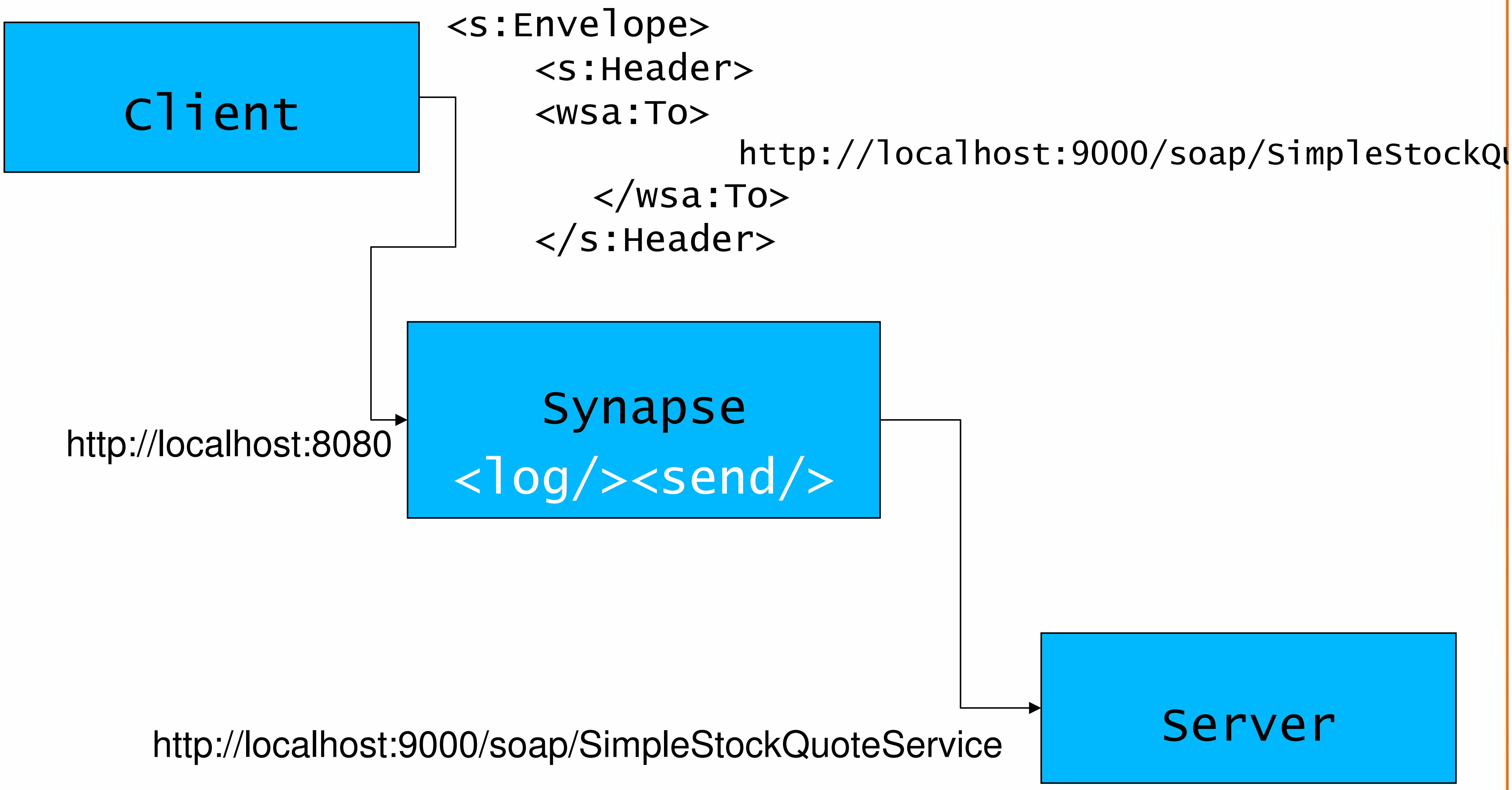
- Start a new command window/shell
- cd <SYNAPSE>/samples/axis2Client
- **ant smoke**

```
Buildfile: build.xml
init:
    [mkdir] Created dir: C:\synapse-1.0-RC1-
   SNAPSHOT\samples\axis2Client\target\classes
compile:
    [javac] Compiling 9 source files to C:\synapse-1.0-
   RC1-SNAPSHOT\samples\axis2Client\target\classes
smoke:
     [java] Standard :: Stock price = $87.36470681025163

BUILD SUCCESSFUL
Total time: 16 seconds
```

Synapse console log

```
2007-11-12 12:56:49,812 [127.0.0.1-pzfdell] [main]  INFO ServerManager Ready
    for processing
2007-11-12 12:56:58,062 [127.0.0.1-pzfdell] [I/O dispatcher 7]  INFO PipeImpl
    Using simulated buffered Pipes for event-driven to stream IO bridging
2007-11-12 12:56:58,187 [127.0.0.1-pzfdell] [HttpServerWorker-1]  INFO
    LogMediator To: http://localhost:9000/soap/SimpleStockQuoteService,
    WSAction: urn:getQuote, SOAPAction: urn:getQuote, ReplyTo:
    http://www.w3.org/2005/08/addressing/anonymous, MessageID:
    urn:uuid:761389B80D31F94EF41194872217881, Direction: request, Envelope:
    <?xml version='1.0' encoding='utf-8'?><soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"><soapenv:Header><wsa:To>h
    ttp://localhost:9000/soap/SimpleStockQuoteService</wsa:To><wsa:MessageID>u
    rn:uuid:761389B80D31F94EF41194872217881</wsa:MessageID><wsa:Action>urn:get
    Quote</wsa:Action></soapenv:Header><soapenv:Body><m0:getQuote
    xmlns:m0="http://services.samples/xsd"><m0:request><m0:symbol>IBM</m0:sym
bol></m0:request></m0:getQuote></soapenv:Body></soapenv:Envelope>

2007-11-12 12:56:58,250 [127.0.0.1-pzfdell] [HttpServerWorker-1]  INFO
    TimeoutHandler This engine will expire all callbacks after : 86400
    seconds, irrespective
of the timeout action, after the specified or optional timeout
```

What’s going on?



```
<s:Envelope>
    <s:Header>
    <wsa:To>
             http://localhost:9000/soap/SimpleStockQu
      </wsa:To>
    </s:Header>
```

Synapse.xml

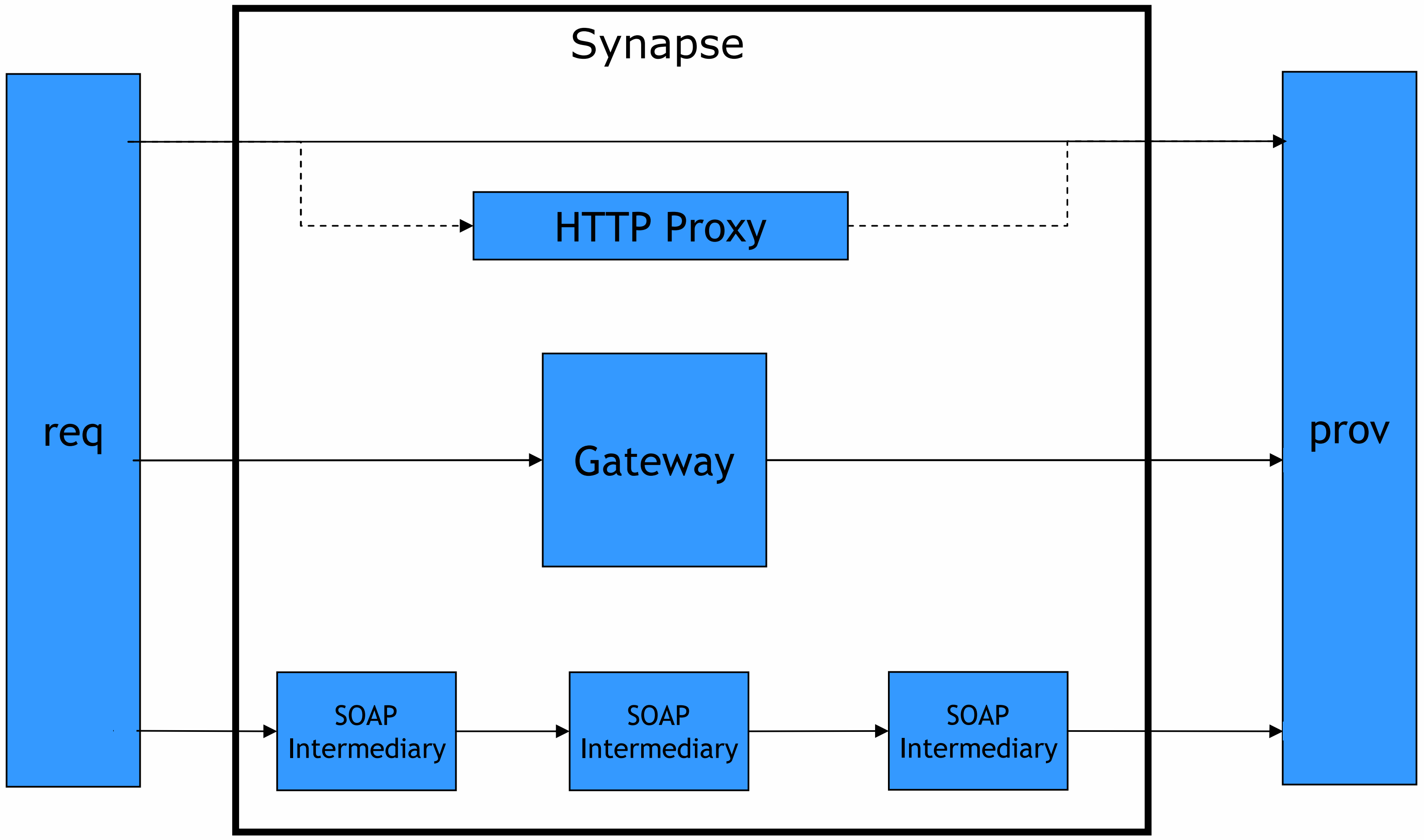
```
<!-- A simple Synapse configuration -->
<definitions
  xmlns="http://ws.apache.org/ns/synapse">

  <!-- Log all messages passing through -->
  <log level="full"/>

  <!-- Send the messages where they have been
  sent (i.e. implicit "To" EPR) -->
  <send/>

</definitions>
```

Open Proxy!

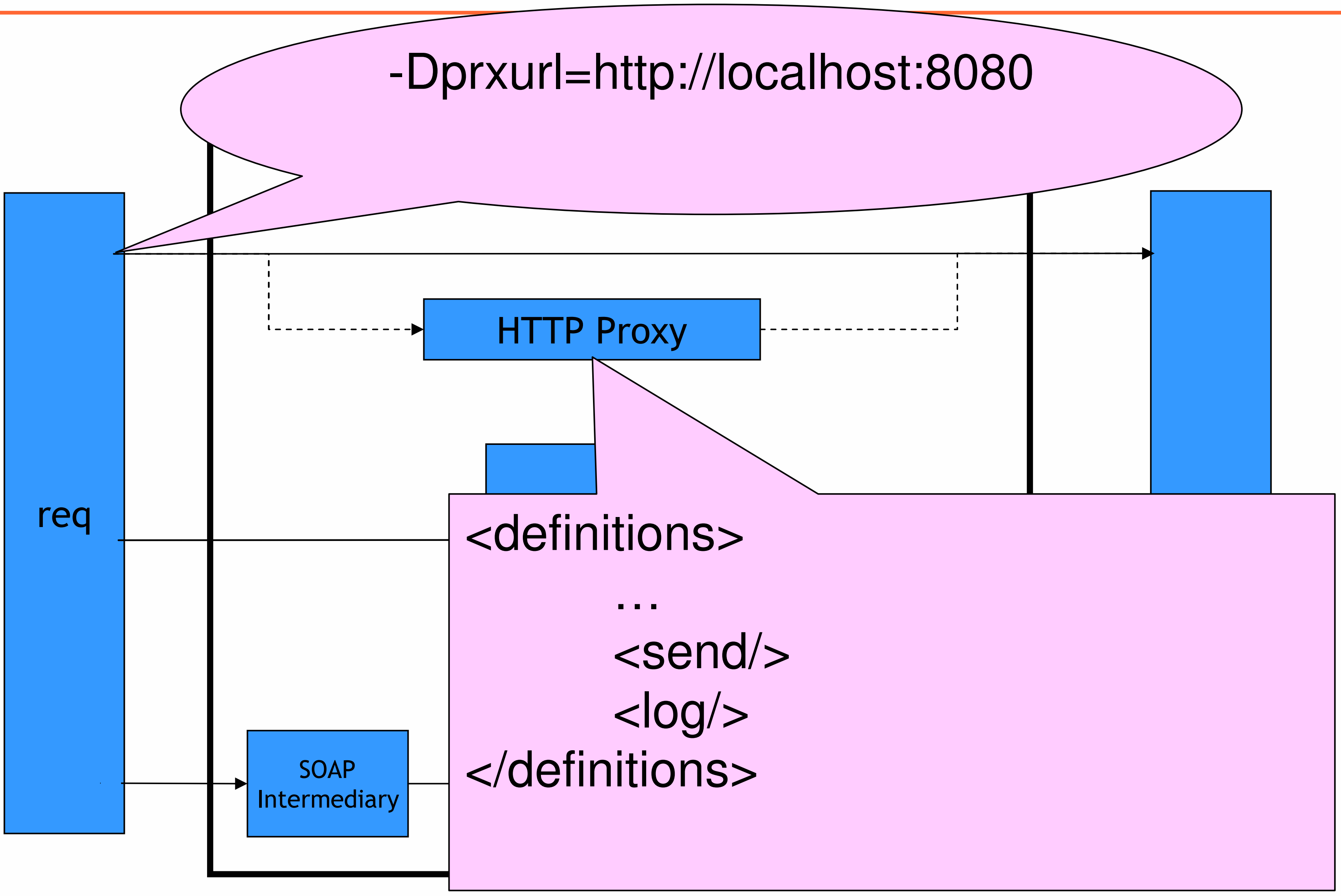
- http://en.wikipedia.org/wiki/Open_proxy
- Generally thought to be a security hole – especially if running within the firewall
- **Be aware that several of the samples implement an open proxy!**
- We changed the default synapse.xml

```
<in>
   <filter source="get-property('To')"
      regex="http://localhost:9000.*">
      <send>
  </filter>
 </in>
```

Deployment Approaches



Deployment Approaches

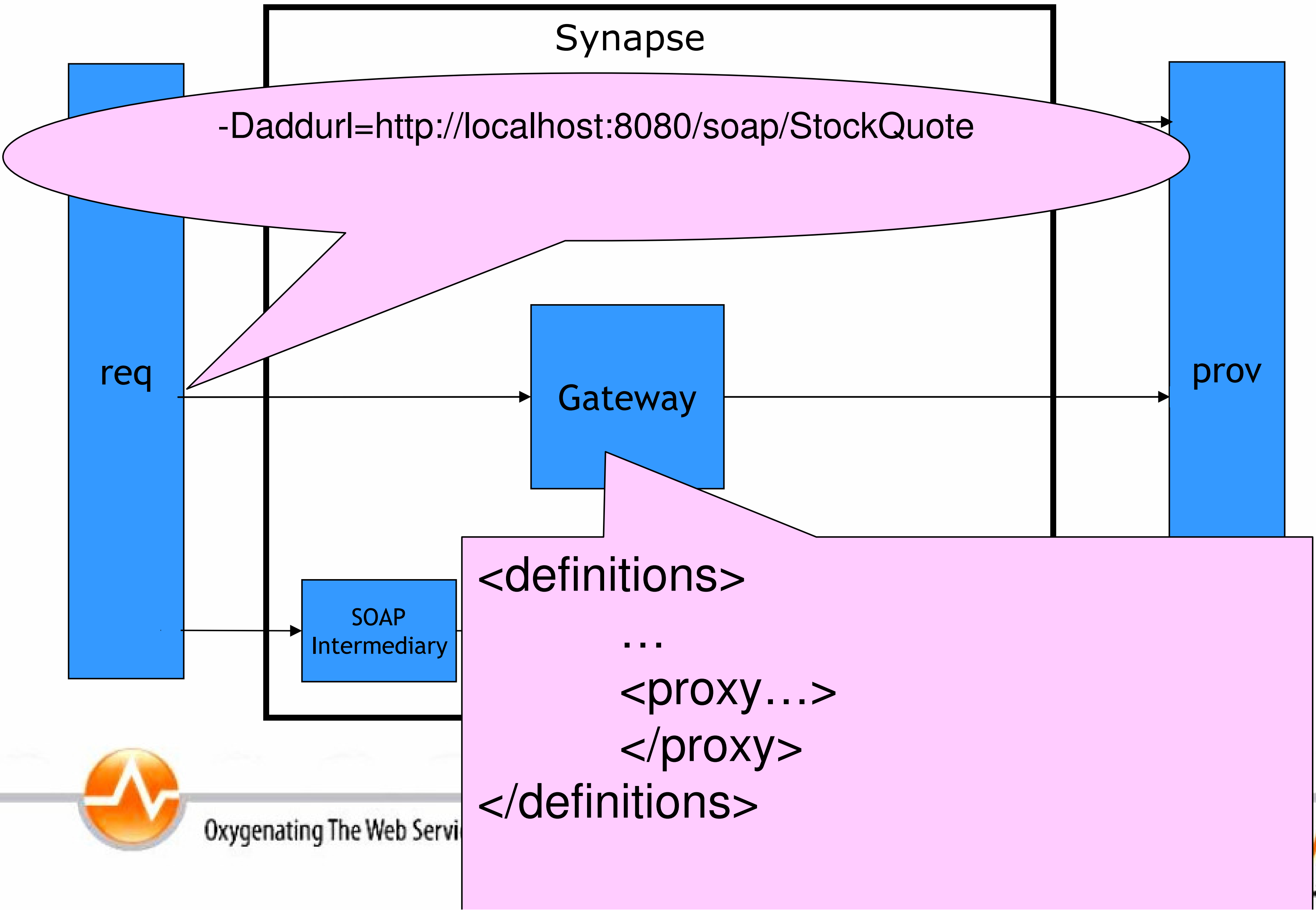


```
<definitions>
        …
        <send/>
        <log/>
</definitions>
```

Benefits of acting as an HTTP Proxy

- Almost every SOAP client can have the proxy redefined without recoding
  - e.g. .NET app.config
  - java –D system properties
- Can define "policies" that apply globally
  - For example, logging
  - Filters can be used to identify particular services
  - Generic XPath expressions can be used to look for certain tags
    - At a performance cost

Deployment Approaches



```
<definitions>
        …
        <proxy…>
        </proxy>
</definitions>
```

Advantages of the Gateway model

- Simple to manage and understand
- Easy to configure which transports are engaged
  - Can specify JMS Queues, SMTP email addresses, etc
- Performant
  - No generic filters required to do things per-service
- Can be used to build a central set of services, hiding implementation details from the clients
  - Each service can be available via multiple options
    - XML/JMS, POX/HTTP, SOAP, RM, Sec etc

Synapse as a SOAP intermediary

- Relies on the client using different URLs for
  - the HTTP transport
  - and for WS-A <wsa:To> header
- The transport points to Synapse
- The <wsa:To> points to the real address

Synapse configuration language

```
<definitions>
    …
    <task name=“string” ...>...</task>
    <sequence name="string">...</sequence>
    <endpoint name="string">...</endpoint>
    <proxy name="string" ...>...</proxy>
    mediator*
</definitions>
```

Endpoints

- A way of defining remote (target) endpoints that can then be called
- A logical concept that can include:
  - Directly defined endpoints (URL)
  - WSDL-defined endpoints
  - A failover group
    - Try each in order until one works
  - A load-balance group
    - Round-robin across the endpoints
  - Other extensions

A sample endpoint

```
<endpoint name="simple">
   <address
    uri="http://l:9000/soap/SimpleStockQuoteService"/>
</endpoint>
```

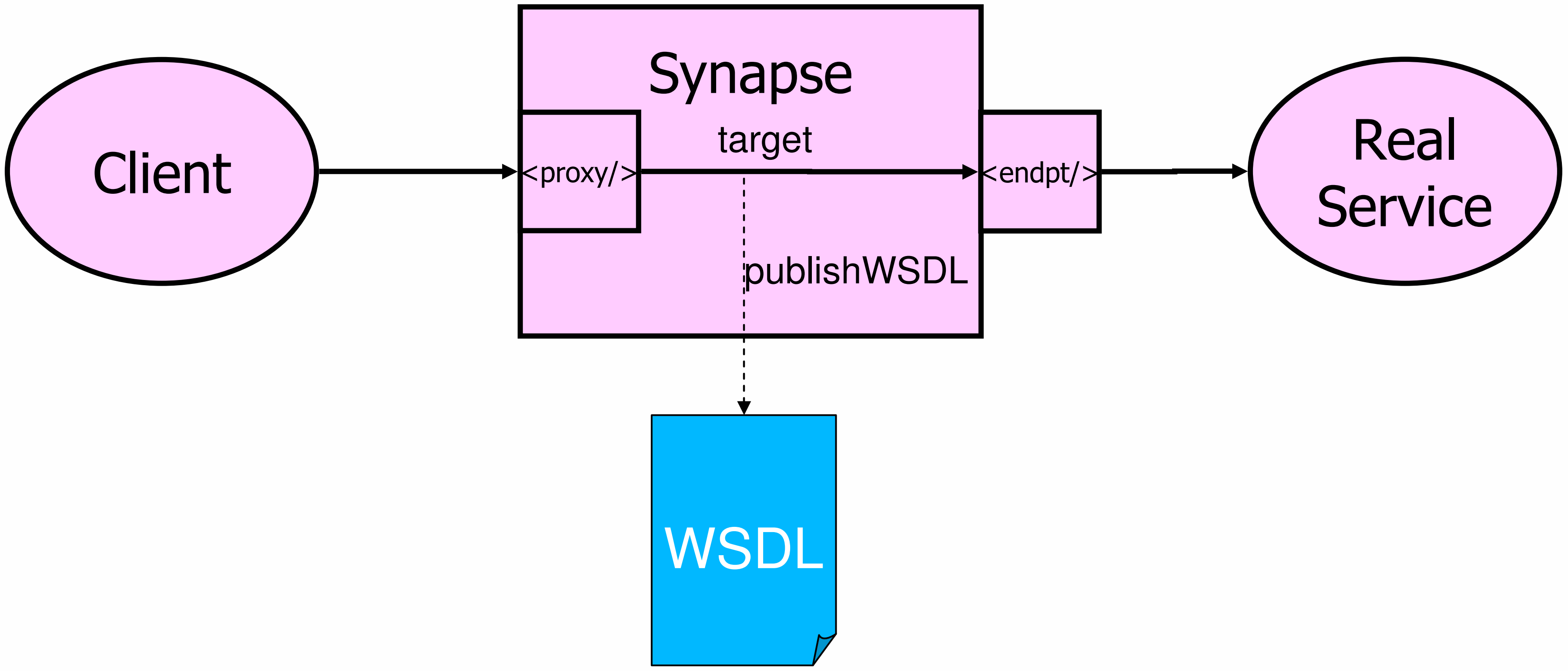
A more complex endpoint:

```
<endpoint name="SOAP12_Addressing_RM">
   <address
    format="soap12"
    uri="http://l:9000/soap/SimpleStockQuoteService"/>
    <enableAddressing/>
    <enableRM/>
</endpoint>
```

Defining proxies - Sample 100

```
<!-- introduction to Synapse proxy services -->
<definitions xmlns="http://ws.apache.org/ns/synapse">
   <proxy name="StockQuoteProxy">
      <!-- name becomes the service name locally-->
      <target>
         <endpoint>
            <address
         uri="http://l:9000/soap/SimpleSQService"/>
         </endpoint>
         <outSequence>
            <send/>
         </outSequence>
      </target>
         <publishWSDL
      uri="file:repository/conf/sample/resources/proxy/sample_proxy_1.wsdl"/>
   </proxy>
</definitions>
```

Proxy



Let's run it

Sample 100

- Synapse

  bin\synapse –sample 100

  Browse

  http://localhost:8080/soap/StockQuoteProxy?wsdl

- Client

  ant stockquote

  -Daddurl=http://localhost:8080/soap/StockQuoteProxy

Default mediators

- send – send message to the default or defined endpoint
- drop – drop this message and end the mediation flow
- log – log this message with log4j
- makefault – create a fault message
- transform – apply XSLT to transform the message
- header – modify headers
- filter – apply sub-mediators when regex and xpath filters match
- switch – do one action of several
- class – call a Java class mediator
- validate – do XSD validation on the message
- property – define properties on the in-memory message context
- sequence – call another sequence
- in – only do sub-mediators for WSDL "in" messages
- out – only do sub-mediators for WSDL "out" messages

What is a sequence?

```
<sequence name="main">
    <log level="full"/>
    <send/>
</sequence>
```

A named ordered list of mediators

The sequence named "main" is applied to incoming messages that aren't targeted at a proxy service endpoint

If there is no sequence called main then it is created out of any mediators in the <definitions> tag.

ApacheCon leading the wave of open source

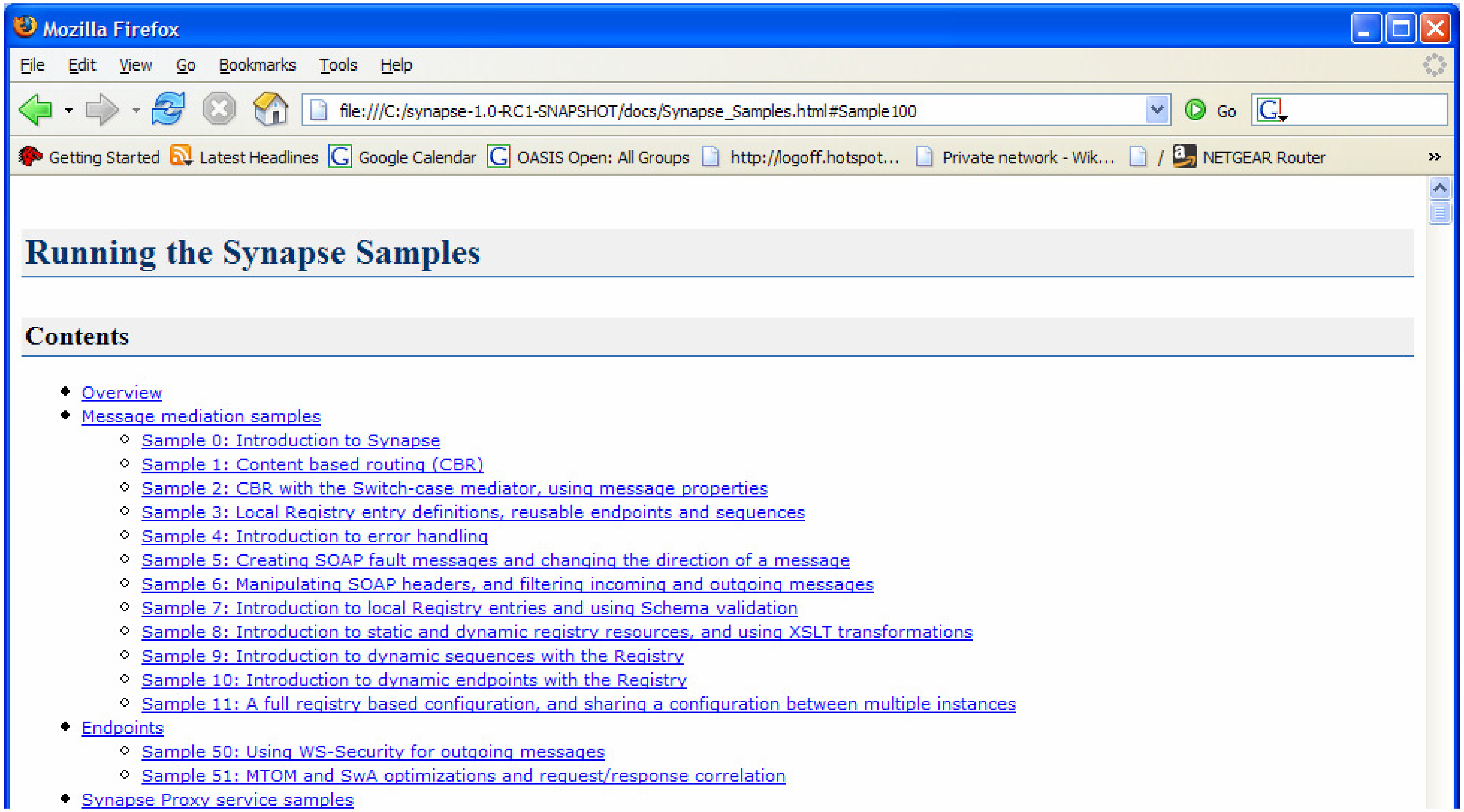
An example use of sequences

```
<sequence name="stockquote">
  …
</sequence>

<sequence name="main">
  <switch source="get-property('To')">
    <case regex=".*/StockQuoteService.*">
      <sequence ref="stockquote"/>
    </case>
    <case regex=".*/stockQuote.*">
      <transform …/>
      <sequence key="stockquote"/>
    </case>
    <default>
      <drop/>
…
```

ApacheCon leading the wave of open source

A word about the samples

<SYNAPSE>\docs\Synapse_Samples.html


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///C:/synapse-1.0-RC1-SNAPSHOT/docs/Synapse_Samples.html#Sample100
Go
Getting Started Latest Headlines Google Calendar OASIS Open: All Groups http://logoff.hotspot... Private network - Wik... / NETGEAR Router

Running the Synapse Samples

Contents

Time for a coffee break!

Recap

- By now you should have a good understanding of:
  - Synapse as an intermediary
  - Different deployment models
  - Getting Synapse running
  - Running a sample
  - How to define a proxy service
  - How to log all messages

Interlude

- How can you get involved?

- Have you already signed up with JIRA?
- Log JIRAs!
- Join us at synapse-dev@ws.apache.org
- Create a class mediator and contribute it
- Submit a patch
- Let us know what you are doing with Synapse
- Become a committer

What next?

- Content-based routing and properties
- Manipulating headers
- Fault handling
- Returning faults
- Filters, switch/case, transformation
- Using scripts
- Writing mediators

Content based routing

- Changing behaviour based on data inside the message
- Not just the SOAP message, but also message properties and context
- Two options

```
<filter…> <!--Only apply mediator if filter matches -->
   <mediator..>
</filter>

<switch source="xpath"> <!– only one will execute -->
   <case regex="string">…</case>
   <default>…</default>
</switch>
```

Filter example

- Sample 1

```
<!-- simple content based routing of messages -->
<definitions xmlns="http://ws.apache.org/ns/synapse">
    <!-- filtering of messages with XPath and regex
   matches -->
    <filter source="get-property('To')"
   regex="http://virtual/StockQuote.*">
        <send>
            <endpoint>
                <address
      uri="http://l:9000/soap/SimpleStockQuoteService"/>
            </endpoint>
        </send>
    </filter>
    <send/>
</definitions>
```

Switch case

- Sample 2

```
<switch source="//m0:getQuote/m0:request/m0:symbol"
    xmlns:m0="http://services.samples/xsd">
```

[Notice we need to define any namespaces that are going to be used in XPath expressions.

Namespaces for XPath expressions can be defined in any XML parent of the expression within the config]

Sample 2 continued

```
<case regex="IBM">
    <!-- the property mediator sets a local property
  on the *current* message -->
    <property name="symbol" value="IBM – not bad"/>
</case>
<case regex="MSFT">
    <property name="symbol" value="MSFT- Are you
  sure?!"/>
</case>
```

Understanding properties

- Properties are defined on the current message
- A bag of properties, together with some "well-known" ones:
  - To, From, WSAction, SOAPAction, ReplyTo, MessageID
- You can also modify underlying properties of Axis2 and the Transport using these
- <property/> mediator sets and removes them:

```
<property name="string"
   [action="set|remove"]
   (value="literal" | expression="xpath")
   [scope=transport|axis2]/>
```

Using properties

- Properties are available as part of the XPath engine using the syntax
  - get-property('To')

- This can be used in filters, switch statements, and other places where expressions are allowed
- For example, copying one property to another:

```
<property name="new" expression="get-property('old')"/>
```

- Later we will see how to use this to set SOAP headers containing content from the body.

Sample 2 continued

```
<default>
    <!-- it is possible to assign the result
  of an XPath expression as well -->
    <property name="symbol"
          expression=
     "fn:concat('Normal Stock - ',
     //m0:getQuote/m0:request/m0:symbol)"

     xmlns:m0="http://services.samples/xsd"/>
</default>
```

Pretty sneaky huh?

Even more Sample 2

Logging the property we have set:

```
<log level="custom">
  <property name="symbol"
    expression="get-property('symbol')"/>

  <property name="epr-url"
    expression="get-property('To')"/>
</log>
```

Back to Synapse Config

- Header manipulation

- Sample 6

```
<definitions
   xmlns="http://ws.apache.org/ns/synapse">
    <in>
      <header name="To"
   value="http://localhost:9000/soap/SimpleStockQuoteService"/>
    </in>
   <send/>
</definitions>
```

Faults

- Synapse has two facilities for dealing with faults
- Firstly, catching faults
  - *like try/catch*
- Secondly, sending back faults
  - *like throw*

Fault handling sequences

- Synapse allows you to specify sequences that run when a fault is detected
  - The ***default*** sequence is run unless one is specified

```
<sequence name="fault">
 <log level="custom">
  <property name="text"
     value="Error occurred"/>
  <property
     name="message"
     expression="get-property('ERROR_MESSAGE')"/>
 </log>
 <drop/>
</sequence>
```

Specifying a fault-handling sequence

```
<sequence name="normal" onError="faultSeq">
…
</sequence>

<sequence name="faultSeq">
  <!– fault handling goes here -->
</sequence>
```

See Sample 4

Sending faults

- Logically in WSDL, faults can go in either direction (in/out)
- <makefault> creates a fault
- You can fully configure the SOAP fault

```
<makefault version="soap11|soap12">
    <code value="tns:Receiver"
      xmlns:tns="http://www.w3.org/2003/05/soap-envelope"/>
    <reason expression="get-property('ERROR_MESSAGE')"/>
    <node>http://some/optional/node/uri</node>
    <role>http://someother/optional/role/uri</role>
    <detail>This is a string explaining what went wrong</detail>
</makefault>
```

- Must change the direction of the request

```
    <property name="RESPONSE" value="true"/>
```

Front-ending POX with SOAP

SAMPLE 102

```
<proxy name="StockQuoteProxy" transports="https">
   <target>
      <endpoint>
         <address
        uri="http://localhost:9000/soap/SimpleStockQuoteService"
        format="pox"/>
      </endpoint>
      <outSequence>
         <send/>
      </outSequence>
   </target>
   <publishWSDL
    uri="file:repository/conf/sample/resources/proxy/sample_proxy_1.wsdl"/>
</proxy>
```

POX to SOAP

- By default Axis2 exposes services as POX
- So any SOAP to SOAP routing is also a POX to SOAP routing

Combining

- For example:
  - simple E4X script to transform
  - Plus, SOAP/WSSec support
- Front-end a complex WS-Security based endpoint with a simple XML/HTTPS one

JMS to SOAP

- Axis2 has a JMS transport
- Supports:
  - XML/JMS (POX)
  - SOAP/JMS
  - Binary/JMS – wrapped as a base64/MTOM element
- See samples 110 and 113

- Can map XML/JMS to SOAP/WSRM
  - for example bridging an existing JMS destination to a .NET server

Extending Synapse

- Main ways of extending Synapse are:
  - Class mediators
  - Tasks
- More advanced extension points include
  - Extension mediators
  - Transports
  - Registry providers

Class Mediators

```
<class name="org.fremantle.myMediator">
   <property name="Blah" value="hello"/>
</class>
```

- Instantiate a class
  - Just one instance across multiple messages
- Use injection to set String or XML properties
- Then for each message calls

  ```
  boolean myMediator.mediate(MessageContext mc);
  ```
- Gives access to the message, any properties, plus also access the overall Synapse configuration
  - return false if you want the message dropped
- Mediators may implement *ManagedLifecycle* interface
  - init / destroy allows resources to be set up and cleaned up

Axiom

```
<soap:Envelope>
 <soap:Header>
  <myNS:Security soap:mustUnderstand="true">
  </myNS:Security>
 </soap:Header>
 <soap:Body>
  <doSomethingCool>
    ... MEGABYTES OF DATA HERE ...
  </doSomethingCool>
 </soap:Body>
</soap:Envelope>
```

$\longleftarrow$ Build object model to here

```
h = envelope.getHeader(securityQName)
```

...and then you can do

```
body = envelope.getBody();

reader = body.getXMLStreamReader();

while (reader.hasNext()) {
  ...
}
```

Axiom is used inside Synapse

- XPath engine (Jaxen) is coded to use Axiom
- The result:
  - Synapse is efficient with
    - XPath expressions on headers
    - Header modification
    - Routing messages
  - But beware the need to understand your XPath expressions
    - For example – explicitly add [0] to ensure it doesn't continue searching
    - Don't use depth-wildcard searches unless you have to

PayloadHelper class

- Simplifies access to the message body

```
org.apache.synapse.util.PayloadHelper

public static int getPayloadType(MessageContext mc)
public static OMElement getXMLPayload(MessageContext
   mc)
public static void setXMLPayload(MessageContext mc,
   OMElement element)
public static DataHandler
   getBinaryPayload(MessageContext mc)
public static void setBinaryPayload(MessageContext
   mc, DataHandler dh)
```

Also Text, Map, StAX (XMLStreamReader)

Simple example: CSV->XML

```
public boolean mediate(MessageContext mc) {
    DataHandler dh = PayloadHelper.getBinaryPayload(mc);
    BufferedReader br;
    new BufferedReader(new
      InputStreamReader(dh.getInputStream()));
    CSVReader csvReader = new CSVReader(br);

    OMFactory fac = OMAbstractFactory.getOMFactory();
    OMElement el = fac.createOMElement("csv", csvNS);
    // create element to hold data
    while ((nextLine = csvReader.readNext()) != null) {
      rownum++;
      // add elements to XML
    }
    br.close();
    PayloadHelper.setXMLPayload(mc, el);
    return true;
}
```

Tasks

- Simple repetitive actions
- Can also be used to start a long-running activity at startup
- Uses the Quartz Scheduler to run items
  - www.opensymphony.com/quartz
- Tasks must implement the *Task* interface

```
package org.apache.synapse.startup;
public interface Task
{
    public abstract void execute();
}
```

- Tasks may implement the *ManagedLifecycle* interface
- Properties are set by injection (String and XML)

Sample task - MessageInjector

```
public class MessageInjector  implements Task,
   ManagedLifecycle
{
   public void setTo(String url)
   { to = url; }
   public void setMessage(OMElement elem)
   { message = elem; }
   public void execute()  {
     MessageContext mc =
         synapseEnvironment.createMessageContext();
     mc.setTo(new EndpointReference(to));
     PayloadHelper.setXMLPayload(mc,
       message.cloneOMElement());
      synapseEnvironment.injectMessage(mc);
   }
}
```

Task configuration

```
<task
  class="org.apache.synapse.startup.tasks.MessageInjector“
  name=“inject">
  <trigger interval="5000"/>
  <property name="to"
   value="http://localhost:9000/soap/StockQuoteService"/>
  <property name="soapAction" value="urn:getQuote"/>
  <property name="message">
    <m0:getQuote xmlns:m0="http://services.samples/xsd">
      <m0:request>
         <m0:symbol>MSFT</m0:symbol>
      </m0:request>
     </m0:getQuote>
   </property>
</task>
```

Adding your own XML config

- As well as a mediator, you need to write a mediator factory and serializer
  - These read the XML and return an instance of your mediator (or vice versa)
- You can then package the mediator, factory and serializer into a JAR
  - META-INF\services\o.a.s.config.xml.MediatorFactory
    - lists additional services
    - See synapse-extensions.jar for an example
- Now any user can drop the JAR into the Synapse classpath and the extension will be supported

Other extension points

- Registry providers
- Endpoints and dispatchers are extensible
  - Support different ways of defining endpoints
    - e.g. UDDI
  - Different session approaches
- Axis2 modules allow other WS-* protocols to be supported
- Axis2 transports allow other transports to be added

Scripts

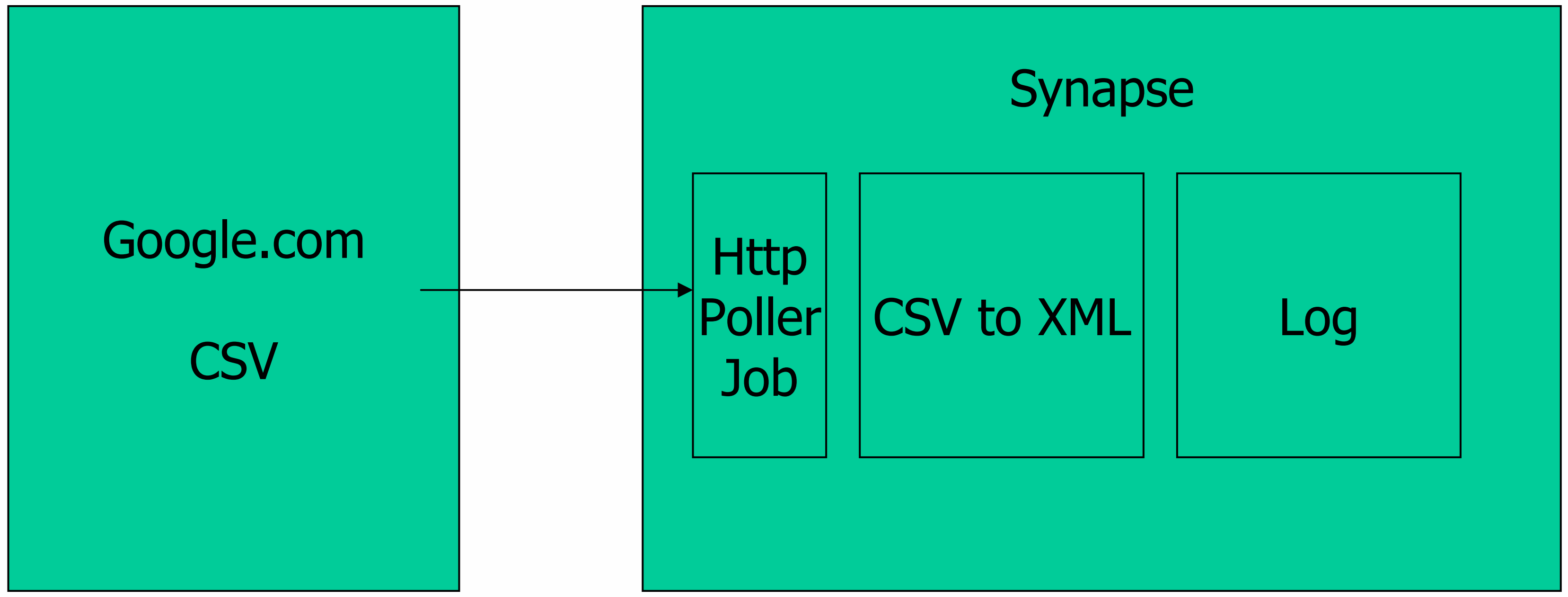
- Synapse supports scripting languages using the Bean Scripting Framework (http://jakarta.apache.org/bsf/)
  - Samples for
    - Javascript/E4X
    - JRuby and REXML
- Scripts can effectively modify the messages as they pass through Synapse
- Intuitive way to change messages

Example E4X

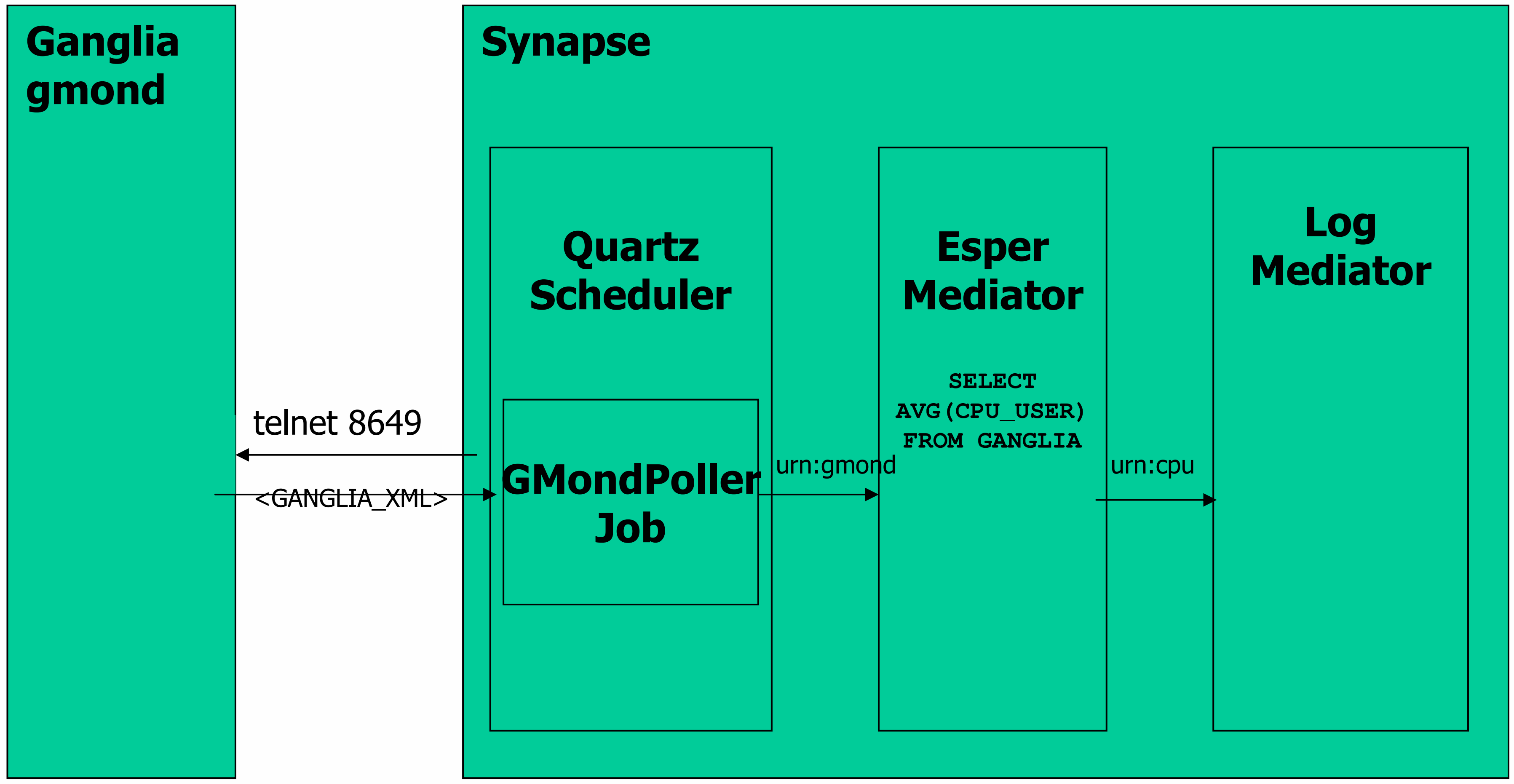
```
 <!-- transform the custom quote request into a
   standard quote requst expected by the service -->
<script language="js"><![CDATA[
    var symbol =
        mc.getPayloadXML()..*::Code.toString();
    mc.setPayloadXML(
        <m:getQuote
            xmlns:m="http://services.samples/xsd">
            <m:request>
                <m:symbol>{symbol}</m:symbol>
            </m:request>
        </m:getQuote>);
]]></script>
```

Some examples

Google Spreadsheet and CSV



Ganglia, Quartz, Esper



```
SELECT
AVG(CPU_USER)
FROM GANGLIA
```

I'm a coffee addict

In case you hadn't already guessed

Recap

- By now you should have a good understanding of:
  - Fault handling
  - Filters
  - Switch/case handling
  - Properties
  - How to create Mediators and Tasks

What next?

- Registries
- Non-blocking IO
- Load-balancing and failover
- Transport switching
  - XML/HTTP and SOAP
  - JMS
- WS-Security
- WS-ReliableMessaging

Understanding "Registries"

- Synapse doesn't implement a registry
  - But can use one

- Motivations:
  - Have a set of Synapse instances using a shared config
  - Moving away from a monolithic synapse.xml
  - By having multiple XML fragments, different people can manage different endpoints
  - By setting cache timeouts, make Synapse both dynamic and efficient

What is a "Registry"?

- We don't really care ☺
- Any mapping of "keys" to XML fragments
- Defined by an interface, and a plug-point
- Synapse comes with a URL-based registry by default
  - Allows HTTP retrieval of XML fragments

Entries

- Registry entries can be used in lots of places instead of directly incorporating the data into the synapse.xml
- An entry can be a string, XML element or imported URL
- Can be used for:
  - Sequence definitions
  - Endpoint definitions
  - Schemas
  - WS-Policies
  - WSDLs
  - XSLTs
  - Scripts

localEntry

```
<localEntry key="mytext">Text</localEntry>

<localEntry key="validate_schema">
   <xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.apache-synapse.org/test"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   targetNamespace="http://services.samples/xsd">
      <xs:element name="getQuote">
      …
      </xs:element>
   </xs:schema>
</localEntry>
```

localEntry

- A local entry has higher precedence than a remote entry (i.e. a real key in the remote registry)
- A simple way of setting a value against a key
- You don't need a remote registry to use local keys
- Can also be set with a URL

```
<localEntry key="test"
   src="http://my.com/my.xml"/>
```

Example Schema Validation

- Sample 7

```
<in>
    <validate>
        <schema key="validate_schema"/>
        <on-fail>
            <!-- if the request does not validate against
   schema throw a fault -->
            <makefault>
                <code value="tns:Receiver"
   xmlns:tns="http://www.w3.org/2003/05/soap-envelope"/>
            <reason value="Invalid custom quote request"/>
            </makefault>
            <property name="RESPONSE" value="true"/>
            <header name="To"
               expression="get-property('ReplyTo')"/>
        </on-fail>
    </validate>
</in>
```

Remote registries

- In this case we will demonstrate using just file-based URLs
- In real life more likely HTTP store
  - Could be HTTPD, SVN, CVS, or other

```
<registry
    provider="org.apache.synapse.registry.url.SimpleURLRegistry">
  <!-- the root property of the simple URL registry
    helps resolve a resource URL as root + key -->
  <parameter name="root">
      file:./repository/conf/sample/resources/
  </parameter>
  <!-- all resources loaded from the URL registry
      would be cached for this number of milliseconds -->
  <parameter name="cachableDuration">15000</parameter>
</registry>
```

Examples of using resources

<xslt key="transform/transform_back.xslt"/>

Read's
    file:./repository/conf/sample/resources/transform/transform_back.xslt

Applies it to the message

The file will be re-read every time the mediator runs – except cached for the ***cachableDuration***

A few more examples

Sample 9:

```
<sequence key="sequence/dynamic_seq_1.xml"/>
```

Will apply the sequence from that xml file

Sample 10:

```
<send>
   <endpoint key="endpoint/dynamic_endpt_1.xml"/>
</send>
```

Will send the message to a dynamically defined endpoint

Sample 11:

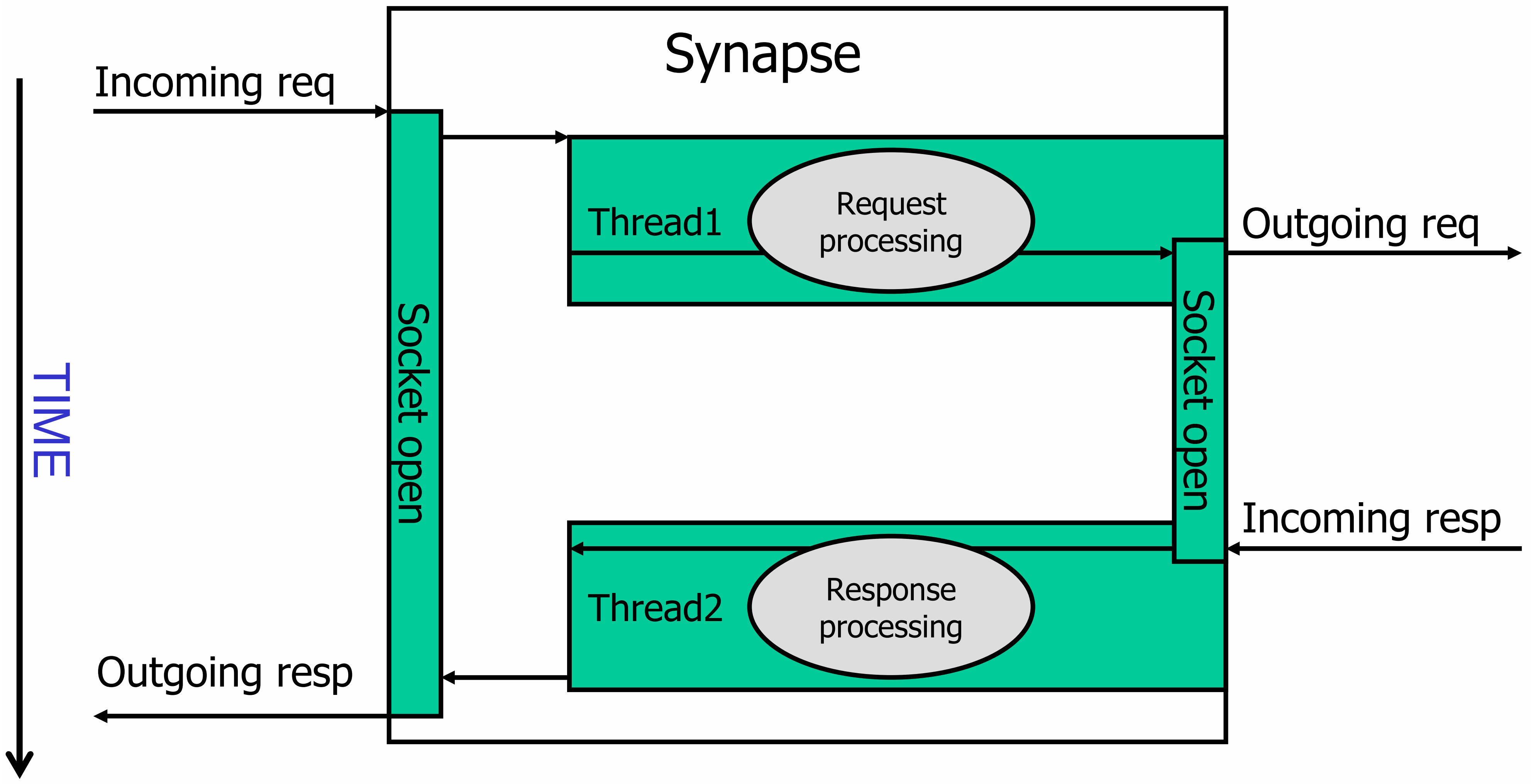
```
<definitions xmlns="http://ws.apache.org/ns/synapse">
    <registry
   provider="org.apache.synapse.registry.url.SimpleURLRegi
   stry">
    </registry>
</definitions>
```

Will read the whole synapse.xml from the registry using key "synapse.xml"

Asynchronous/Non-Blocking

- WS-Addressing or JMS cases are no problem
- The concern is "anonymous" HTTP clients
  - who are blocking waiting for a response on the HTTP backchannel – in other words on the same socket connection
- We do not want Synapse to block in this case
- Unlike a service endpoint (e.g. Axis2), Synapse is not usually busy all the time between receiving the request and sending the response
  - Why not? Waiting for the target service!
- The code is actually a full Axis2 transport, so Axis2 also will get this benefit

Non-blocking graphically

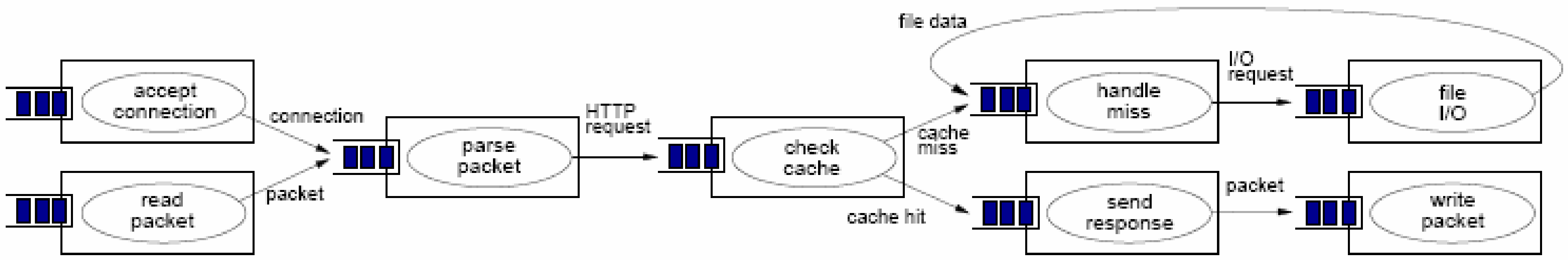


This model means:

1. Synapse threads never blocked during normal processing
2. Number of sockets open >> number of threads

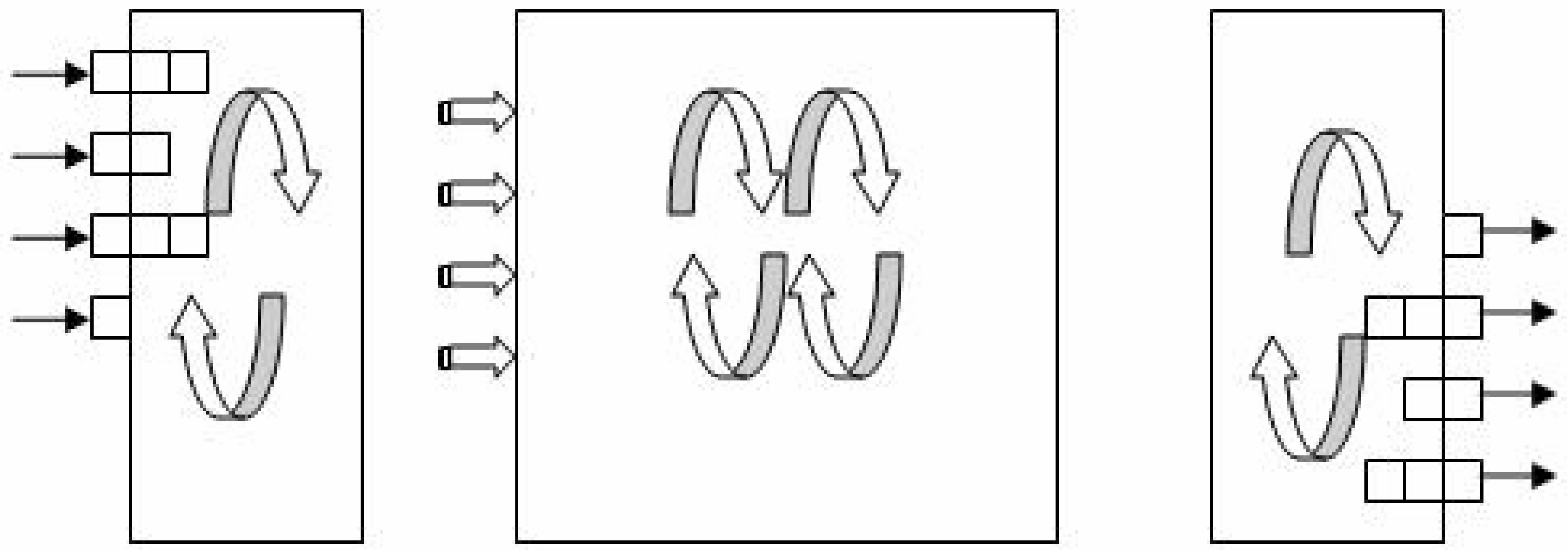
Scalable Event Driven Architecture

- Simple model of stages and queues for handling load
- Matt Welsh's PhD thesis



http://www.eecs.harvard.edu/~mdw/proj/seda/

NIO model is effectively SEDA



NIO Listener with two dedicated threads

Synapse executing using its own thread pool

NIO Sender with two dedicated threads

Demonstrating Non-Blocking

- Synapse by default runs
  - 2 listener threads
  - 2 sender threads
  - 8 worker threads
- Added a 100ms thread sleep to the server
- Ran 250 concurrent clients for 10000 runs
  - Simply would not have run without NIO
- Also did a simple test comparing:
  - 346 bytes in/ 1,170 bytes out
  - Direct to Axis2: 7.4ms
  - Via Synapse: 8.1ms – diff = 0.710ms!!

Load-balancing

- Simple load-balancing endpoint (round-robin) with failover by default

```
<endpoint>
    <loadbalance failover="true|false">
        <session type="soap|http|simpleClientSession"> (optional)
        <endpoint …/>
        <endpoint …/>
    </loadbalance>
</endpoint>
```

Endpoints are defined recursively, so you can have a load-balance across a failover group of WSDL endpoints, for example

Session affinity allows you to use:

HTTP cookies, Axis2 SOAP sessions, or header: <syn:ClientID>

Failover is basic – if an endpoint fails it is removed from the group

WS-Security

- Axis2 module Rampart
  - Supports
    - WS-Security 1.0, 1.1
    - WS-SecurityPolicy 1.1
    - WS-SecureConversation
    - WS-Trust
  - Works together with Sandesha to secure RM 1.0 and 1.1
- In Synapse, completely configured by using WS-SecurityPolicy

WS-Security inbound – sample 103

```
<proxy name="…">
    <enableSec/>
    <policy key="inbound_sec_policy"/>
</proxy>
```

WS-Security outbound - Sample 50

```
<localEntry key="sec_policy"
  src="file:repository/conf/sample/resources
  /policy/policy_3.xml"/>

<endpoint name="secure">
   <address
     uri="http://localhost:9000/soap/SecureStockQuoteService">
     <enableSec policy="sec_policy"/>
     <enableAddressing/>
   </address>
   </endpoint>
```

WS-Security continued

- Remove the header on the way out

```
<out>
<header
   name="wsse:Security"
   action="remove"
   xmlns:wsse="http://docs.oasis-
     open.org/wss/2004/01/oasis-200401-wss-
   wssecurity-secext-1.0.xsd"/>
<send/>
</out>
```

WS-RM

- Supported through the use of Sandesha2
- Supports WSRM 1.0 and 1.1
  - Default in-memory storage
  - Persistent storage code available at WSO2.org
    - uses Hibernate
- Supported both inbound and outbound

Inbound RM

```
<proxy name="rmendpoint">
    <enableRM/>
</proxy>
```

Automatically supports both versions

Outbound RM

```
<endpoint>
    <address uri="…">
        <enableRM policy="rm-policy-key"/>
    </address>
</endpoint>
```

Also available for WSDL endpoints

Default behaviour is to have one sequence per endpoint

Need to set

<property scope="axis2" name="Sandesha2LastMessage" value="true"/> if you want messages flagged "LastMessage"

Recap

- Synapse functionality
  - Proxy services, Rule-based
  - POX, JMS, SOAP, WS-RM, WS-Sec support
    - (plus other Axis2 transports including SMTP, TCP)
  - Filters – XPath and Regex based
  - XSLT transforms
  - Schema validation
  - Extension through Scripting and Java mediators
  - Ability to use dynamic distributed config

Any remaining questions

Resources

- ws.apache.org/synapse
- docs\
  - Synapse_Configuration_Language.html
  - Synapse_Extending.html
  - Synapse_QuickStart.html
  - Synapse_Samples.html
  - Synapse_Samples_Setup.html
- ws.apache.org/axis2
- http://apache-synapse.blogspot.com

- pzf.fremantle.org