SynapseOld

*** Attn: This document is DEPRECATED Check http://incubator.apache.org/synapse/ for updated information ***

Synapse Project

Synapse is creating a robust, lightweight implementation of a highly scalable and distributed service mediation framework based on Web services specifications. This project is currently under incubation at the Apache Software Foundation; see http://incubator.apache.org/projects/synapse for details.

IRC Chat

The Synapse team has a weekly development-oriented IRC chat on Freenode (irc.freenode.org) on channel #apache-synapse. It takes place every Thursday at 7AM Pacfic, 10AM Eastern, 8PM Sri Lanka (etc).

Design Notes

- Notes from F2F Meeting in Cupertino, CA on September 23rd to 25th.

Milestone 1

- Simple rule engine, text format or simple xml
- Mediators = classes - no parameters
- Log everything
- XSLT
- Xpath then XSLT or Log

Milestone 2

- Text rules format/ PolicyAttachment syntax
- Mediators with parameters and references (via existing or simple IoC container)
- same scenarios plus
- Script/E4X
- Chain, plus other composite mediators
- Configuration of QoS

Features

- Connect
    1. Protocols
    2. Routing (XPATH)
    3. Loose Source
- Manage
    1. Loggging
    2. JMX
    3. Load Balancing/Failover
    4. Schema Validation
- Transform
    1. XSLT
    2. E4X
    3. BPEL
    4. Versioning

Types

- Explicit Mediation (Expose intranet web services with a explicit firewall with security)
- Implicit Mediation (Transparent)

Rules

	Condition	Mediator	QoS
1	*	Log	~~NA~~
2	wsa:To= http://fremantle.org/*	send to http://dims.com	Security, Reliable, Transaction

Composition Models

1. Several Rules (Declarative Model) 2. Composite Mediator (Procedural Model)
    a. Chain Mediator 2. BPEL Mediator 3. Script Mediator

Mediator

1. Single unit for work 2. Gets a message context 3. Has access to a soap infoset which it can modify 4. Set up using Inversion of control

```
public interface Mediator {
   public boolean mediate(MessageContext mc) throws AxisFault;
}
```

Synapse implementation using Axis2 (Option #1)

1. Rule matching engine will be implemented as a module 2. The Rule Matching Handler in Rule matching module will be configured as the first 3. One Single Dummy Synapse Service with fixed dispatch (using a Synapse Dispatcher) 4. Rule matching engine will engage other modules (ws-security) as required. 5. There is a Synapse Message Receiver at the end 6. Message Receiver will call instances of mediators as needed 7. Control will be sent back to Rule matching engine to kick off the next rule.

Synapse implementation using Axis2 (Option #2)

1. Rule matching engine will be implemented as a Dispatcher 2. Build a chain of modules for each Rule 3. One Single Dummy Synapse Service 4. There is a Synapse Message Receiver at the end 5. Message Receiver will call instances of mediators as needed 6. Control will be sent back to Rule matching Dispatcher to kick off the next rule.

Trade Offs

Advantage of Option #2 is that we can set up the chains for each rule and cache it. Option #1 we have to set up chains every time.

Miscellaneous

1. Addressing is handled as a rule. If a rule needs say wsa:To, addressing rule has to be run/added before that. 2. Mediator can be null. If it is null it means the result of mediation is true (proceed to the next rule) 3. For each rule, there can be only one mediator (which is run by synapse message receiver)

Problems

1. How do we handle request-reponse in the case where we don't have wsa headers or is set to anonymous

Itinerary Example

http://f.org/test/	Security, Send to urn:f:test
urn:f:test	Send To machine b
machine B	reliable, send to urn:f:version_test
urn:f: version_test	send to Y