eclipseCON™ 2008

Policy Support in Eclipse STP
www.eclipse.org/stp

By Jerry Preissler & David Bosschaert

 |

Agenda

- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies?
- How can you extend the STP policy editor ?
  - Exercise 3

What is a policy?

In a general context:

"a definite course or method of action selected from among alternatives and in light of given conditions to guide and determine present and future decisions"

www.merriam-webster.com

In a technical context:

A standardized description of the capabilities, requirements or general characteristics of an entity

based on WS-Policy 1.2

For automated processing, policies must posses some key traits

- a standardized, machine-processable syntax

  WS-Policy

- formal definitions for the actual properties that are expressed

  domain specific
  WS-Addressing, WS-RM Policy, WS-Atomic Transaction, WS-BusinessActivity, WS-SecurityPolicy

- a defined method to associate policies with policy subjects

  WS-PolicyAttachment

WS-Policy provides a syntax for expressing policies

```
<wsp:Policy Name="http://www.example.com/policies/P1"
      xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
      xmlns:wsp="http://www.w3.org/ns/ws-policy" >
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:SignedParts>
        <sp:Body/>
      </sp:SignedParts>
      <sp:EncrytedParts>
        <sp:Body/>
      </sp:EncryptedParts>
    </wsp:All>
    <wsp:All>
      <sp:SignedParts>
        <sp:Body/>
      </sp:SignedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Alternative 1

Alternative 2

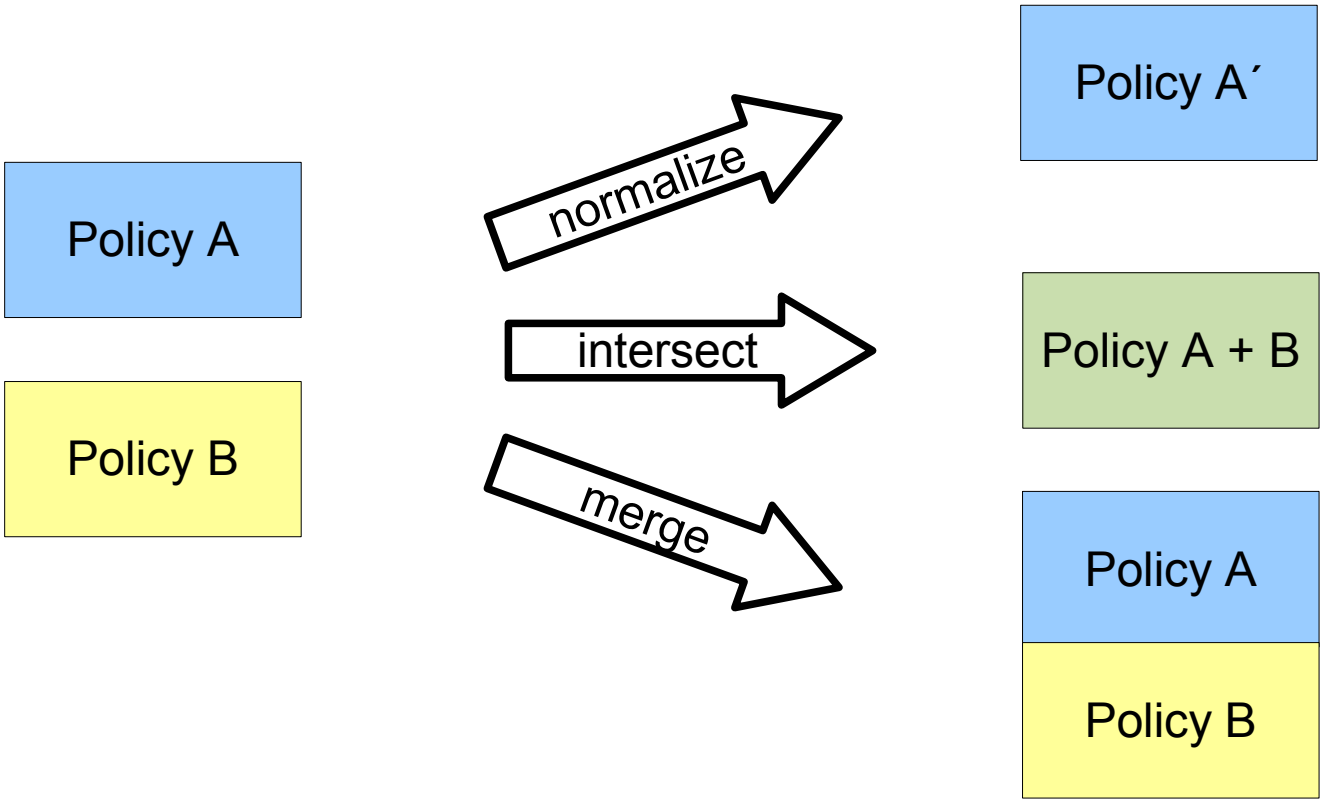
Policy

Alternative

Compositors

Assertions

WS-Policy provides operations to work with policies

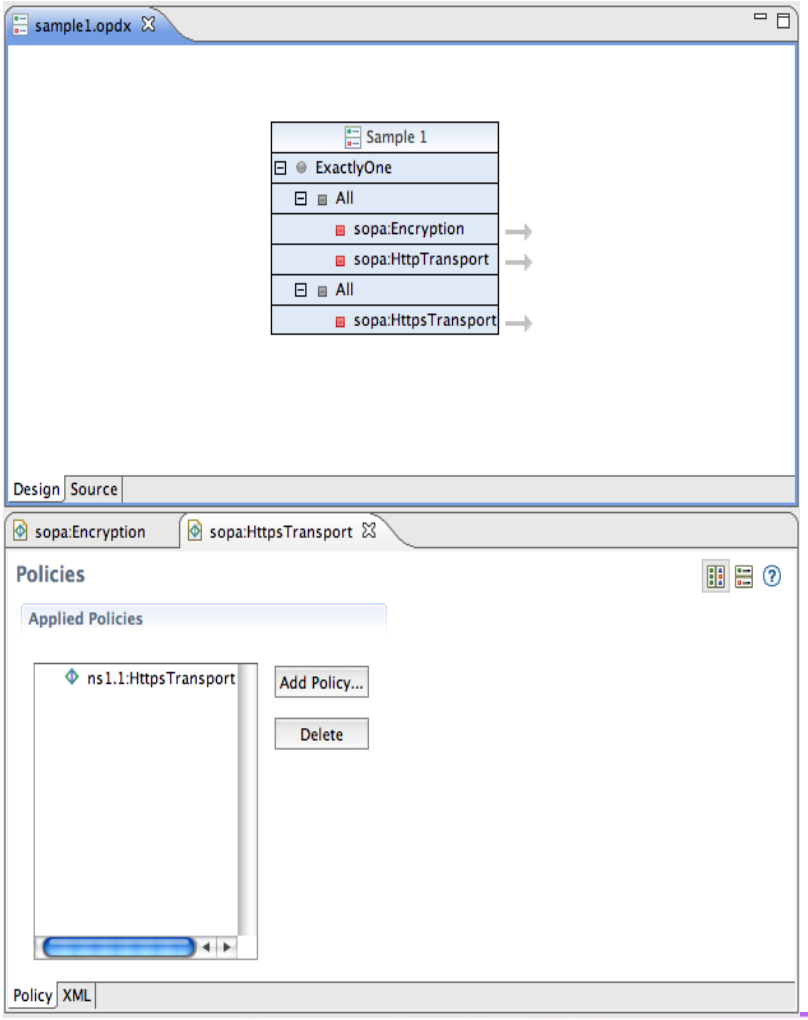


 |

Agenda

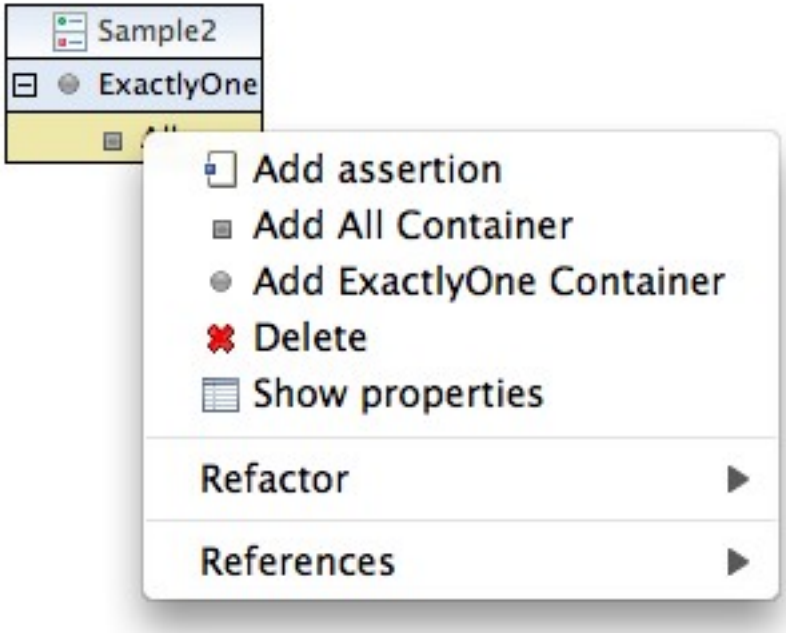
- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies?
- How can you extend the STP policy editor ?
  - Exercise 3

Policy Editor overview



- The policy editor provides two editor windows:
- The high level editor shows the complete structure of the policy
- The detail editor shows one selected policy assertion together with all attributes

The high level editor manipulates the structure of the policy



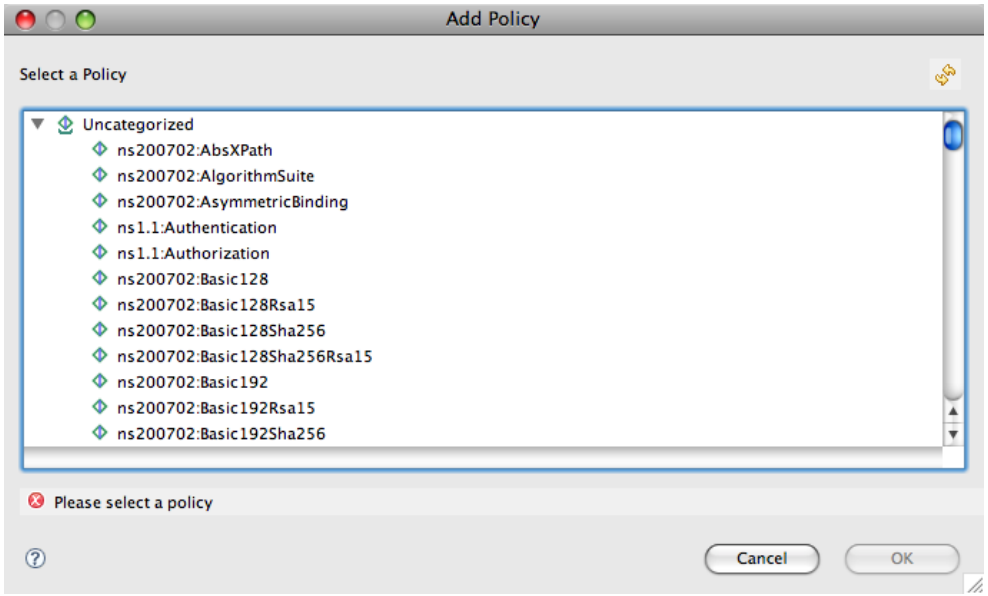
From the high level editor, you can

- add and remove compositors

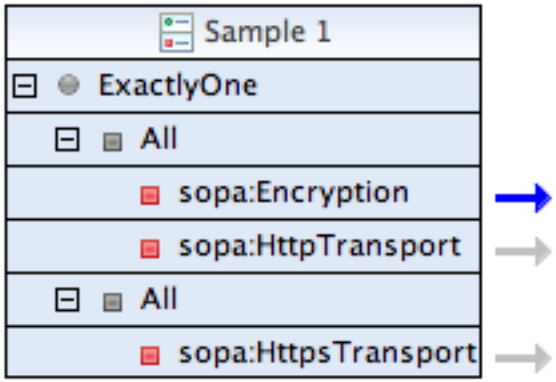
The high level editor manipulates the structure of the policy

From the high level editor, you can

- add and remove compositors
- add and remove individual assertions



The high level editor manipulates the structure of the policy

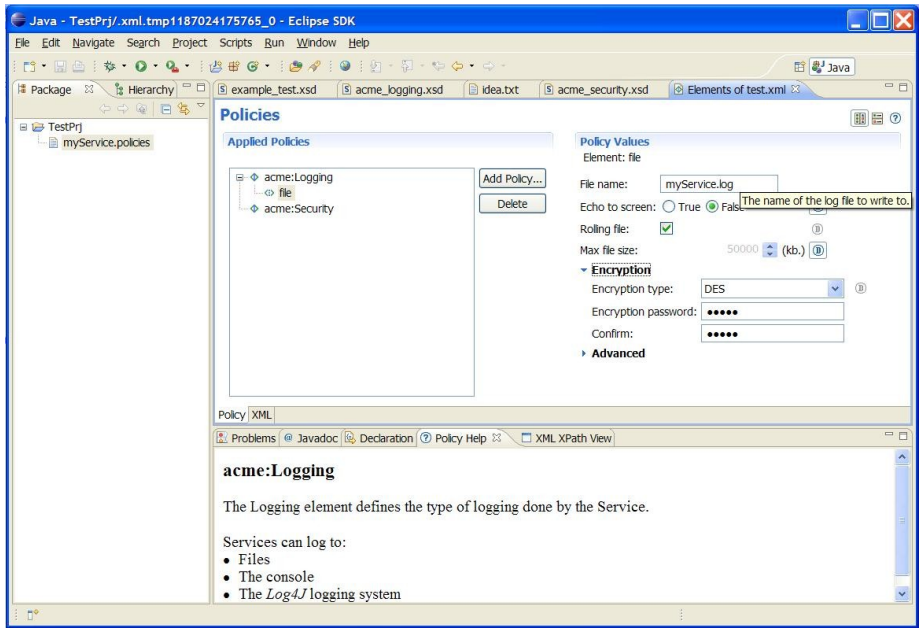


From the high level editor, you can

- add and remove compositors
- add and remove individual assertions
- switch to the detail editor to work with an individual assertion

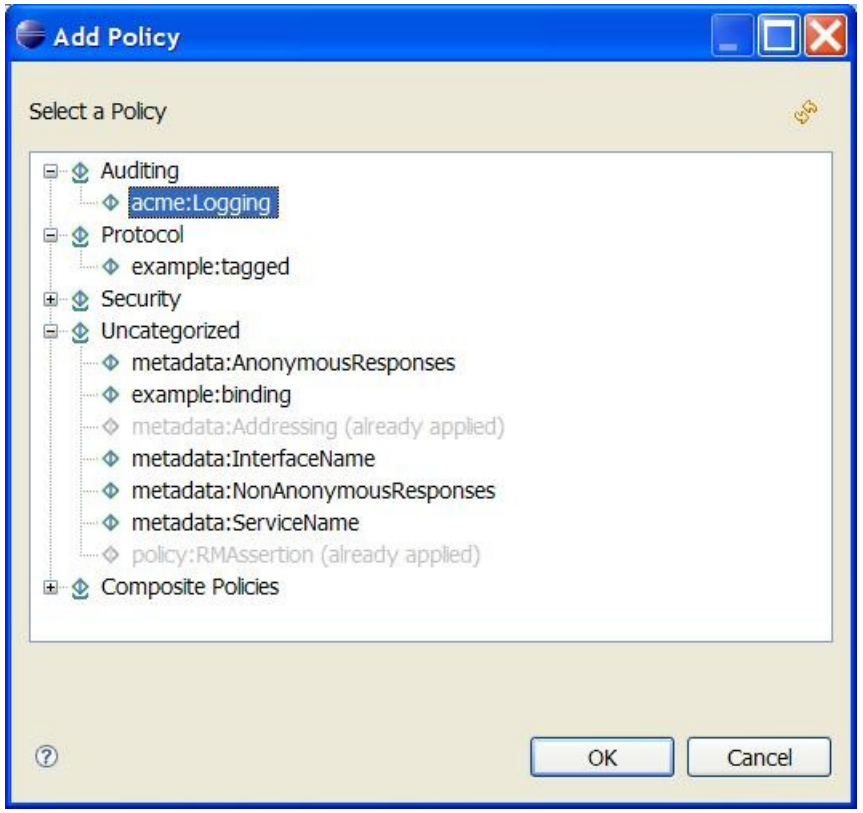
Details Editor

- Similar in look & feel to PDE Extension Point editor
- Can edit the details of WS-Policy assertions as well as other types of XML files that contain embedded elements.
- Editor dynamically synthesizes a GUI based on the schema definition of the policy assertions.
- GUI works with most standard XML Schema definitions
- Based on XEF (also part of STP)



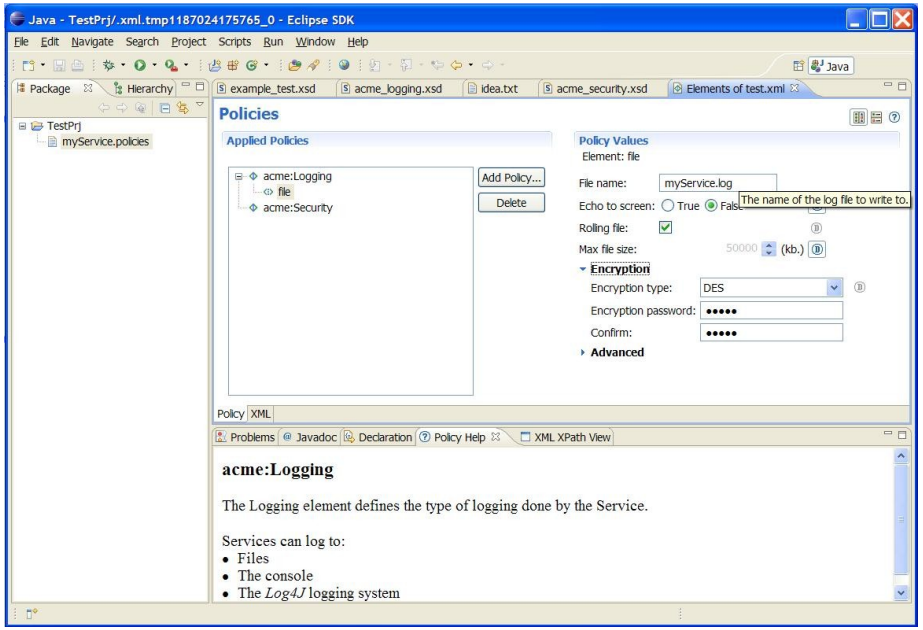
Details Editor – Policy Catalogue

- When editing policies, new ones can be added from a Catalogue.
- The catalog has a simple interface
  - Can serve policies from local filesystem
  - Look up catalog in database
  - WTP XMLSchema Catalog integration



Details Editor – Features

- Widgets for many XSD data types
- Display names, tooltips
- Context-sensitive help
- Display of defaults values
- Required fields
- Enumerated values
- Password fields
- Representation of xs:choice, xs:sequence and xs:any
- Much more...



Details editor – What is being edited

- You can view/edit the source XML too, could look like this (WTP XML Editor):

Details editor – Launching

- Current use is primarily embedded in applications, launching is done by opening an editor by calling `IDE.openEditor()` with a

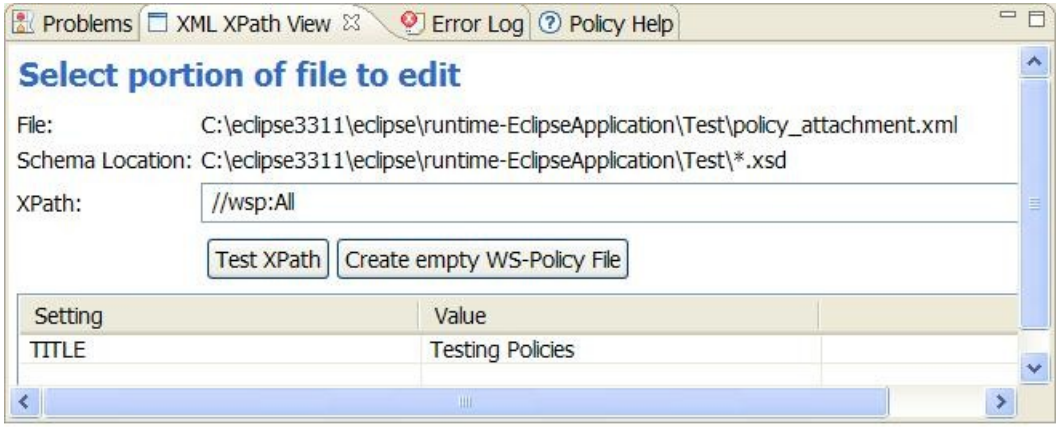
  `org.eclipse.stp.policy.common.editors.IPolicyDetailEditorInput`

  or

  `org.eclipse.stp.ui.xef.editor.XMLProviderEditorInput`

  Editor ID: `org.eclipse.stp.ui.xef.editor.XefEditor`

- For testing there's the XML XPath View:



- It allows you to specify a policy file, what part in the file needs to be edited (as XPath), settings and then open the editor

 |

Agenda

- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies?
- How can you extend the STP policy editor ?
  - Exercise 3

XEF Tutorial part 1 – editing policies

Summary:

Open the details editor directly, using the XML XPath View, to edit:

- WS-PolicyAttachment file
- Embedded WS-Policies in a WSDL file
- A CXF Configuration file (non-WS-Policy)

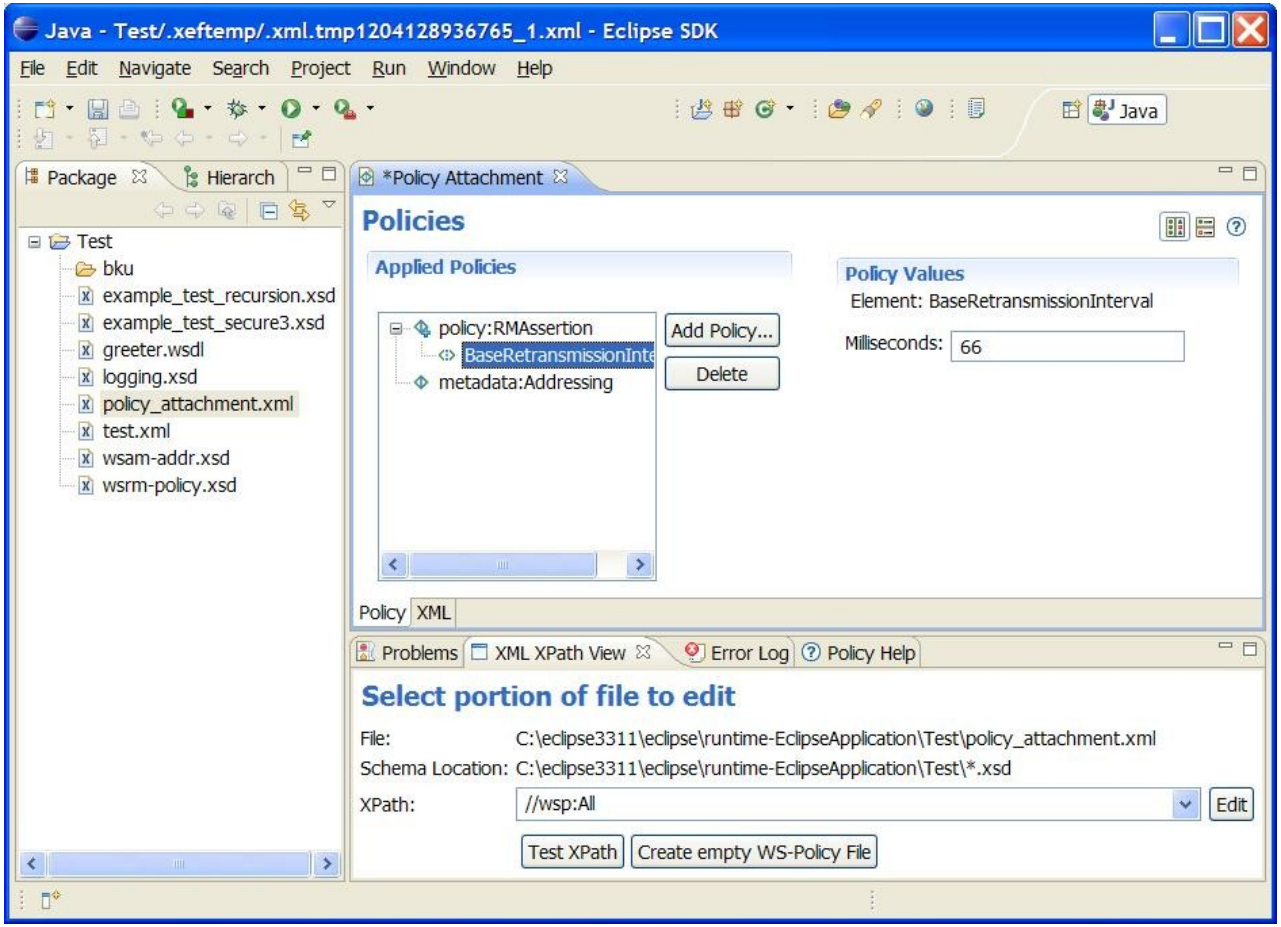
XEF Tutorial part 1 – editing policy documents

Exercises:

1. Add a ws-addressing policy to the policy_attachment.xml file
   - Use the XML XPath view to edit the policies in policy_attachment.xml
   - XPath: `//wsp:All`
2. Open the provided greeter.wsdl file and change the retransmission timeout of the WS-RM policy to 70000.
   - Use the XML XPath view to edit the policies in greeter.wsdl
   - XPath: `//*[local-name()='Policy' and namespace-uri()='http://www.w3.org/ns/ws-policy']`
3. Edit features in a cxf-features.xml file.
4. Stretch exercise – open the editor from code on a memory object (which has no file).

XEF Tutorial part 1 – editing policy documents

The editor will look like this:



XEF Tutorial Part 2 – Create your own Policy Type

Summary:

- Create your own logging policy definition
- Use the policy
- Make it look nice

XEF Tutorial Part 2 – Create your own Policy Type

Exercise:

Create a new logging policy of which an instance looks like this:

```
<acme:Logging xmlns:acme="http://www.acme.com/xsd/2007/08/logging">
  <file filename="mylogfile.log" write_interval="5000" echo="true" />
</acme:Logging>
```

- With two sub-elements: console logger and file logger
- File logger has:
  - a required field ‘filename’
  - echo to screen field (boolean)
  - a write interval (default: 10000 ms)

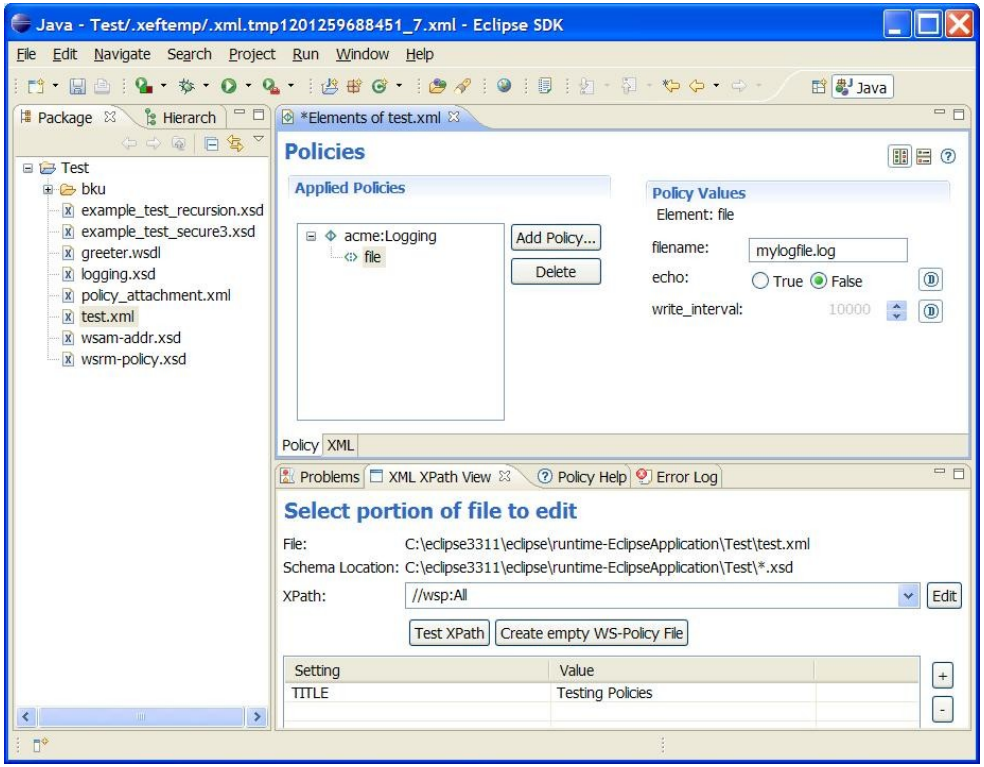
XEF Tutorial Part 2 – Create a Basic Logging Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xef="http://schemas.eclipse.org/stp/xsd/2006/05/xef"
  xmlns:xefgui="http://schemas.eclipse.org/stp/xsd/2006/05/xef/gui"
  targetNamespace="http://www.acme.com/xsd/2007/08/logging"
  xmlns:tns="http://www.acme.com/xsd/2007/08/logging">

  <xs:element name="Logging">
    <xs:complexType>
      <xs:choice>
        <xs:element name="file" type="tns:fileLoggingType"/>
        <xs:element name="console" />
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="fileLoggingType">
    <xs:attribute name="filename" type="xs:string" use="required"/>
    <xs:attribute name="echo" type="xs:boolean" default="false"/>
    <xs:attribute name="write_interval" type="xs:positiveInteger"
       default="10000" />
  </xs:complexType>
</xs:schema>
```

XEF Tutorial Part 2 – Use the new Policy

- Add this logging.xsd to your current Project.
- Edit a policy document, e.g. `test.xml`
- Add the logging policy
- Add the `file` subelement

Raw, but functional editor →



 |

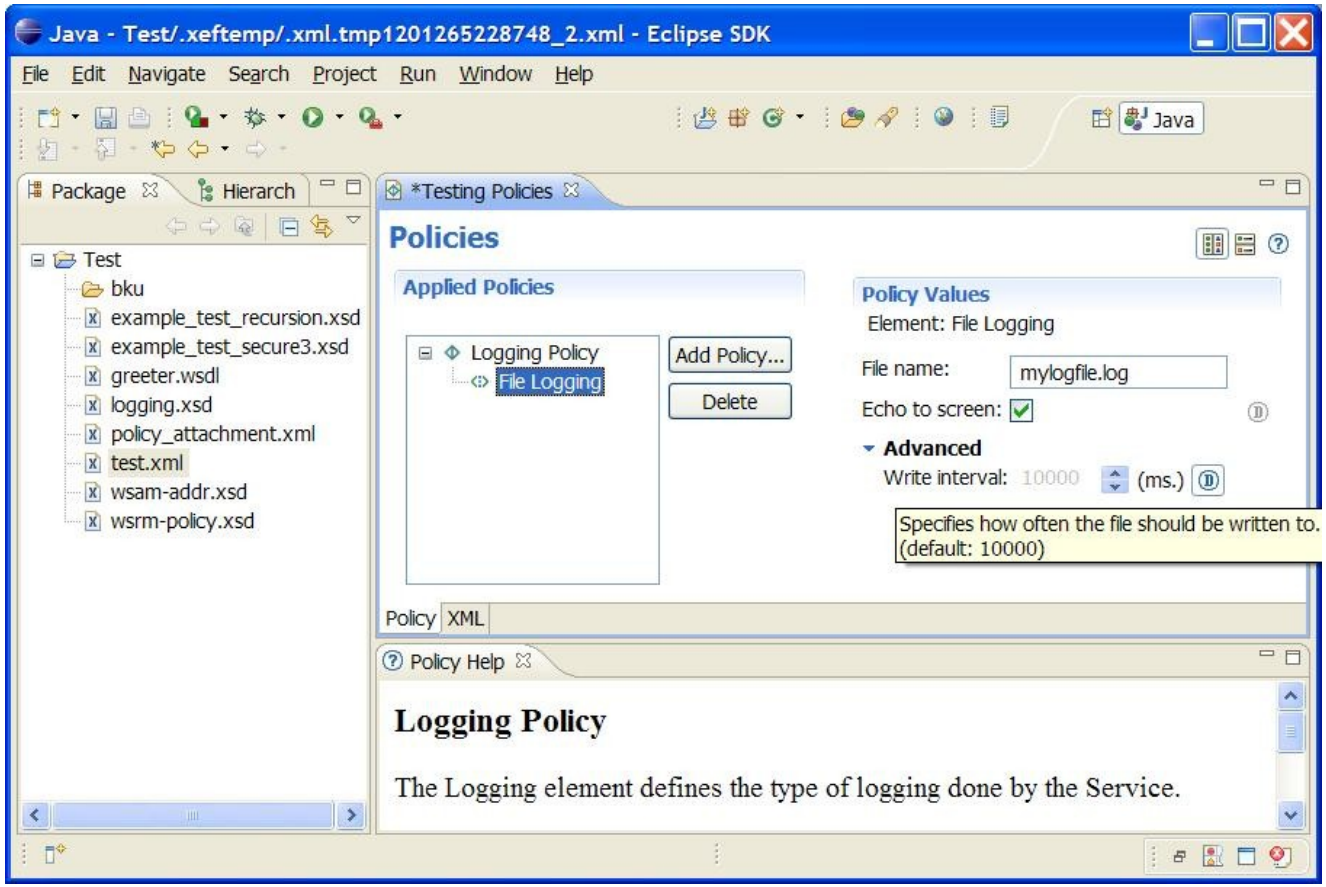
XEF Tutorial Part 2 – Prettify the new policy in the UI

When every thing works, make it look nice

- Policy in ‘Audit’ category and is called ‘Logging’
- Make sure all labels have nice display names
- Put write_interval in an advanced section
- Add a ‘milliseconds’ unit to write_interval
- Make sure everything has tooltips and documentation

XEF Tutorial Part 2 – Prettify the UI

Using the XEF reference, add annotations to make the policy look nice:



- The reference guide is also here: http://wiki.eclipse.org/STP/XEF_Reference

XEF Tutorial Part 2 – Influencing the UI

- An enhanced version with more UI features of this tutorial is available on the STP Wiki: http://wiki.eclipse.org/STP/Policy_editor_documentation
- Logging schemas available:
- Basic logging file is called: logging_basic/logging.xsd
- Final logging file is called: logging_full/logging.xsd

Agenda

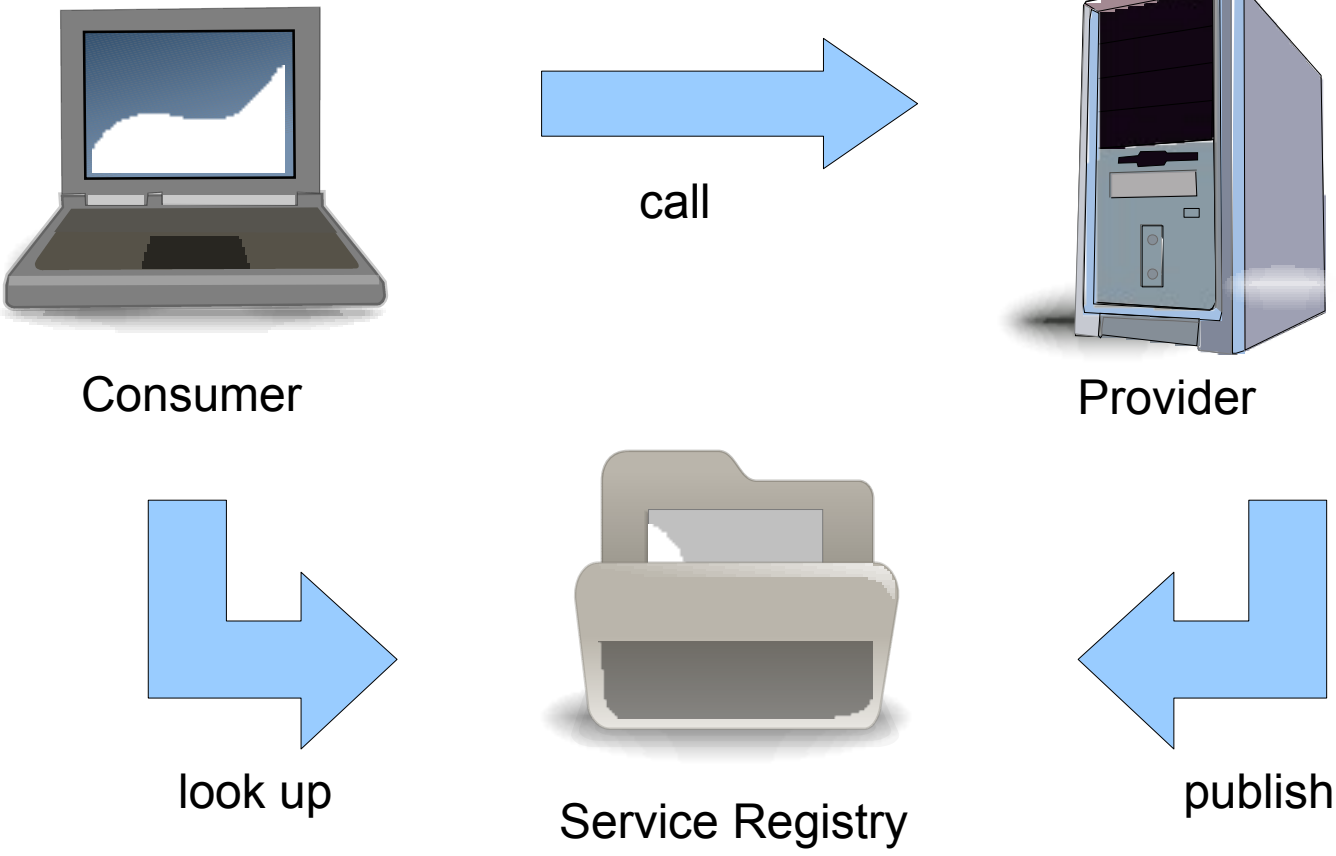
- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies?
- How can you extend the STP policy editor ?
  - Exercise 3

Policy-driven mechanisms can be used to enhance the functionality provided by an SOA

- SOA uses a Service Registry to provide a level of indirection between the service consumer and the service provider
- Non-functional properties of consumers and providers alike can be specified with policies

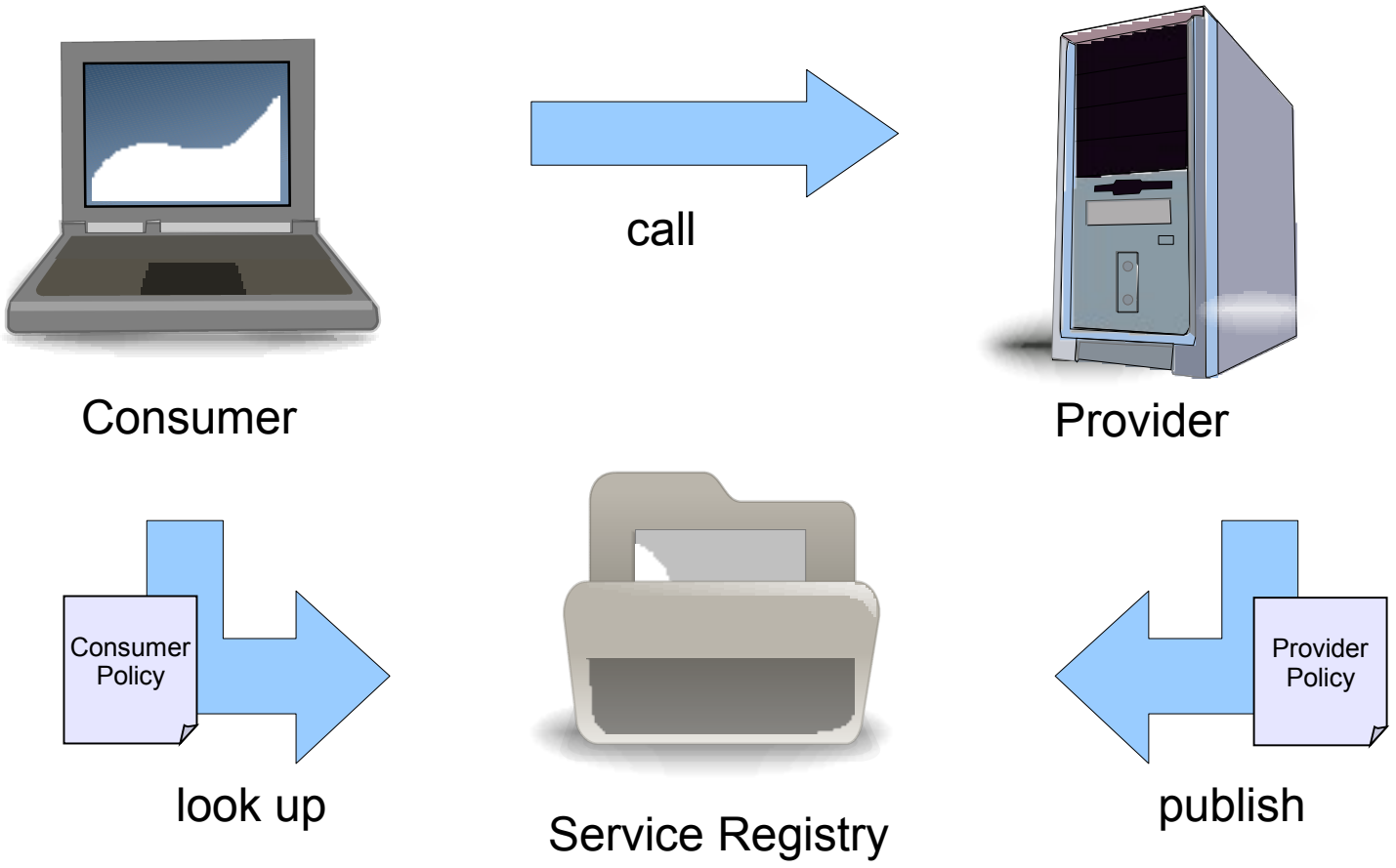
The functionality of an SOA can be enhanced by including policy-driven negotiation into the service provider lookup process

SOA provides a level of indirection between consumer and provider



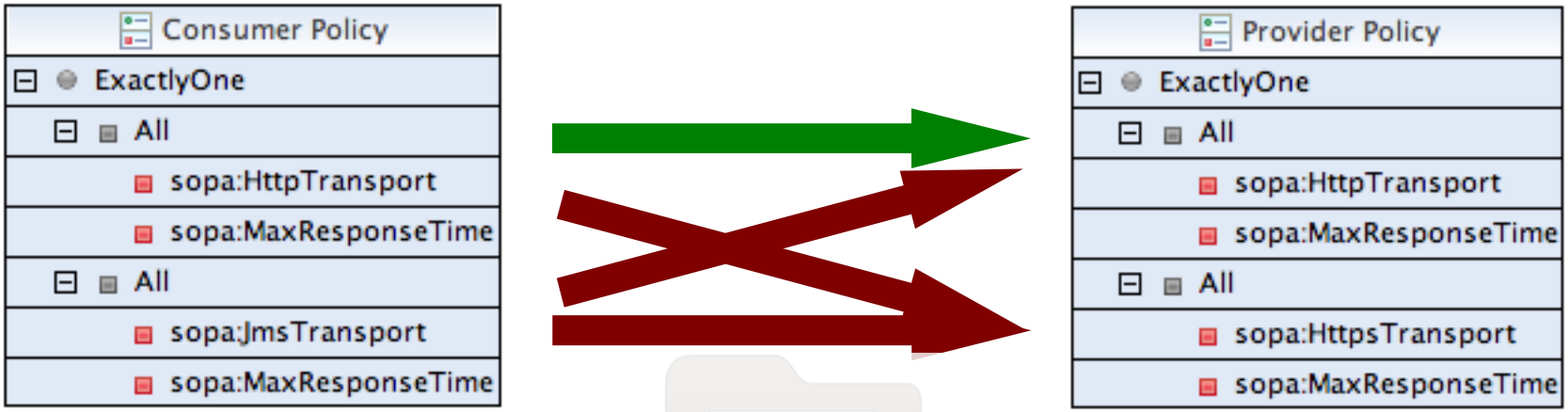
 |

Policies can be integrated in the lookup mechanism



 |

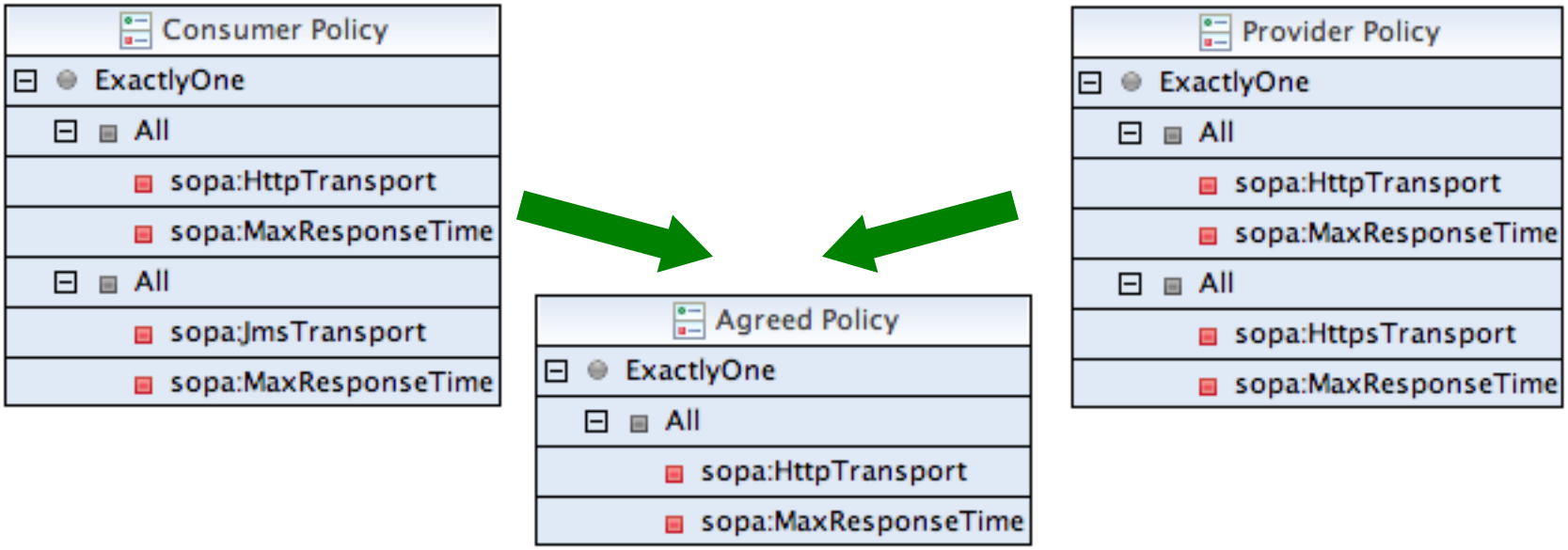
Policies can be integrated in the lookup mechanism



Alternatives are compared crosswise between policies
Non-matching alternatives are rejected
Matching alternatives are included in an “Agreed Policy”

Service Registry

The resulting policy captures the properties that are common to both participants



Matching alternatives are included in an “Agreed Policy”

 |

Policies can be used to control a wide range of behavior

Technical concerns

- Transport selection
- Location-based routing
- Message tracking

Policies can be used to control a wide range of behavior

Technical concerns

Security aspects

- Authentication
- Authorization
- Encryption
- Signature

Policies can be used to control a wide range of behavior

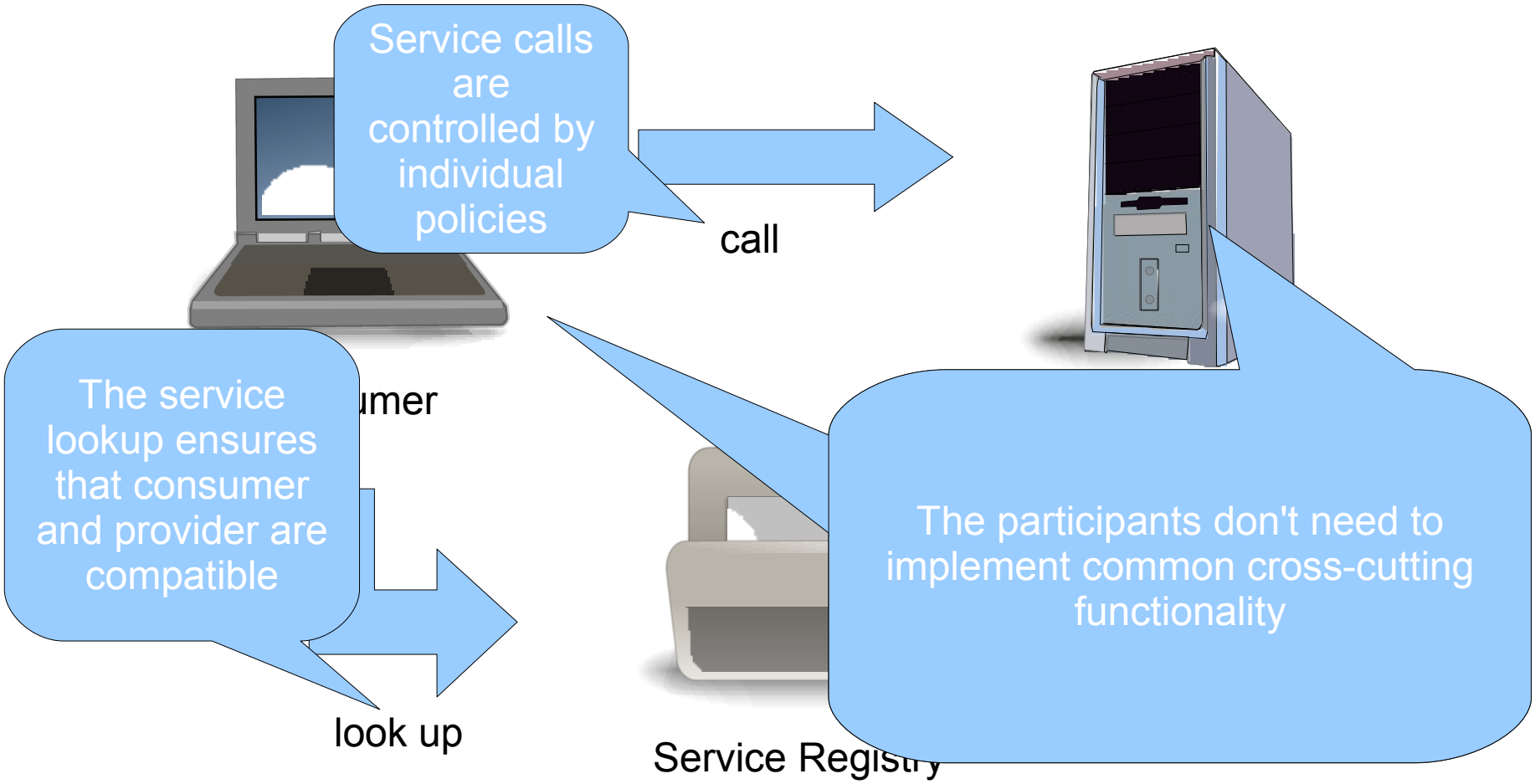
Technical concerns

Security aspects

Quality of service

- Response time
- Reliability
- Cost

The policy-driven mechanism enhances SOA functionality in three important ways

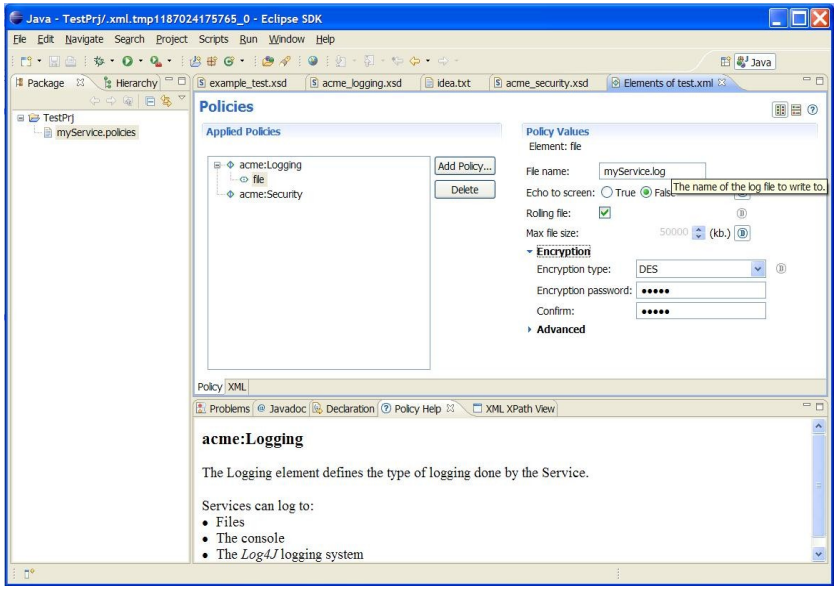


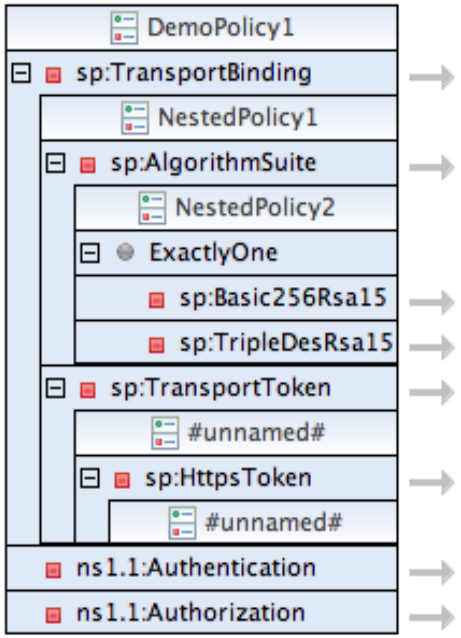
 |

Agenda

- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies ?
- How can you extend the STP policy editor ?
  - Exercise 3

The STP policy editor combines two main contributions

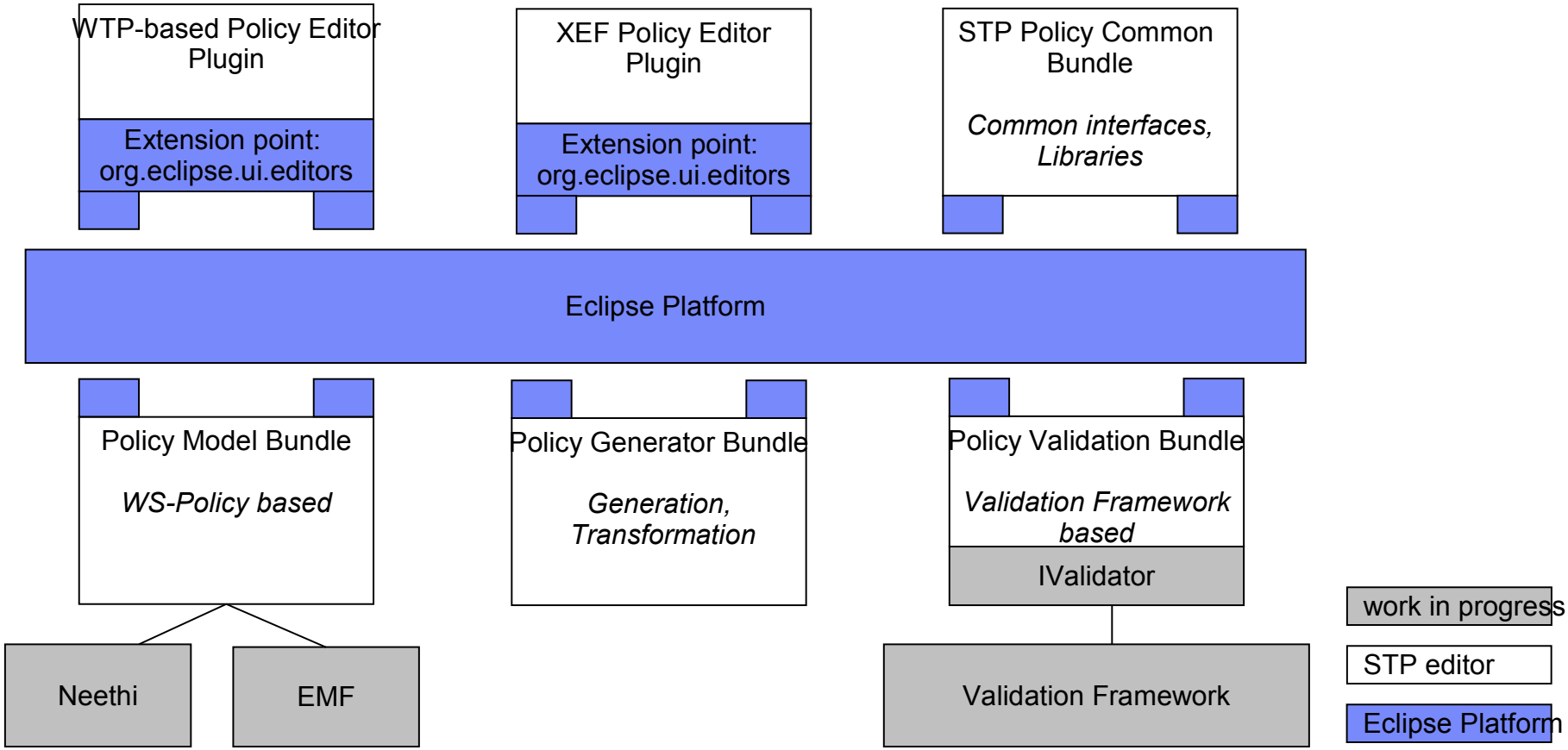




The XEF-based editor was contributed by IONA

The WTP-based editor was contributed by SOPERA

The functionality is distributed across several plugins



Agenda

- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies ?
- How can you extend the STP policy editor ?
  - Exercise 3

XEF Tutorial Part 3 – XEF extension points

Summary:

- Text filters for password fields
- Callback for populating value sets
- Custom field editors

XEF Tutorial Part 3 – Text Filters for Passwords

Password fields can use custom filters to process the value:

XSD Attribute Definition:

```
<xs:attribute name="lock_password" type="xs:string" use="required">
  <xs:annotation>
    <xs:appinfo>
      <xef:displayName>Lock Password</xef:displayName>
      <xef:filter>MyFilter</xef:filter>
      <xefgui:widget>password</xefgui:widget>
    </xs:appinfo>
  </xs:annotation>
</xs:attribute>
```

Plug in filters via Extension Point:

```
<extension
   point="org.eclipse.stp.xef.xefExtension">
   <filter class="org.example.MyFilter"
           filterId="MyFilter" />
</extension>
```

MyFilter (reverses pwd in document):

```
package org.example;

import org.eclipse.stp.ui.xef.editor.TextFilter;

public class MyFilter implements TextFilter {
    public String filter(String data) {
        return new StringBuilder(data).
                    reverse().toString();
    }
}
```

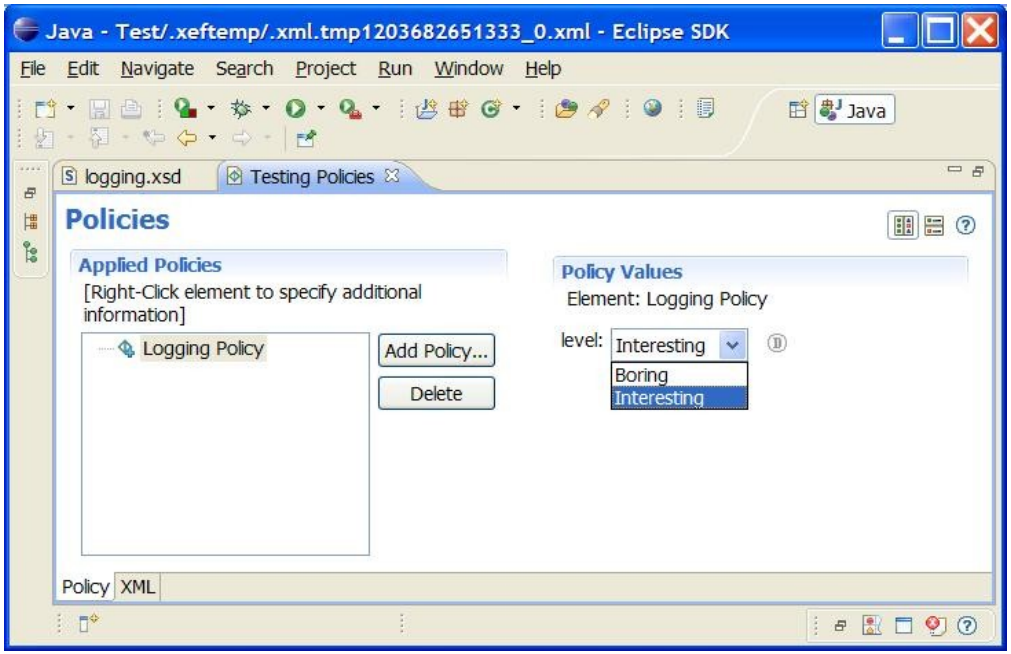
XEF Tutorial Part 3 – Value Proposal Callbacks

- You might want users to select from a prepopulated set of values

  Possible through XSD enumeration:

```
<xs:attribute name="level" default="Info">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Fatal"/>
      <xs:enumeration value="Error"/>
      <xs:enumeration value="Warning"/>
      <xs:enumeration value="Info"/>
      <xs:enumeration value="Debug"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

- Maybe you need a more dynamic approach, where possible values are fed from your application.



XEF Tutorial Part 3 – Value Proposal Callbacks (impl)

- A more dynamic approach is via a callback, declared in XSD:

```
<xs:attribute name="level" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <xefgui:context>
        <xefgui:values>loglevels</xefgui:values>
      </xefgui:context>
    </xs:appinfo>
  </xs:annotation>
</xs:attribute>
```

- Realized through an IContextProvider

```
IContextProvider myCtxProvider = new IContextProvider() {
    public Object getData(String ctxId) {
        return null;
    }
    public String[] getValues(String ctxId, String ctxFilter) {
        if ("loglevels".equals(ctxId)) {
            return new String [] {"Boring", "Interesting"};
        }
        return null;
    }
};
new XMLProviderEditorInput(settings, selectedFile.getProject(),
        new XPathXMLProvider(...), schemaProvider, myCtxProvider),
```

- Currently only supported via XMLProviderEditorInput

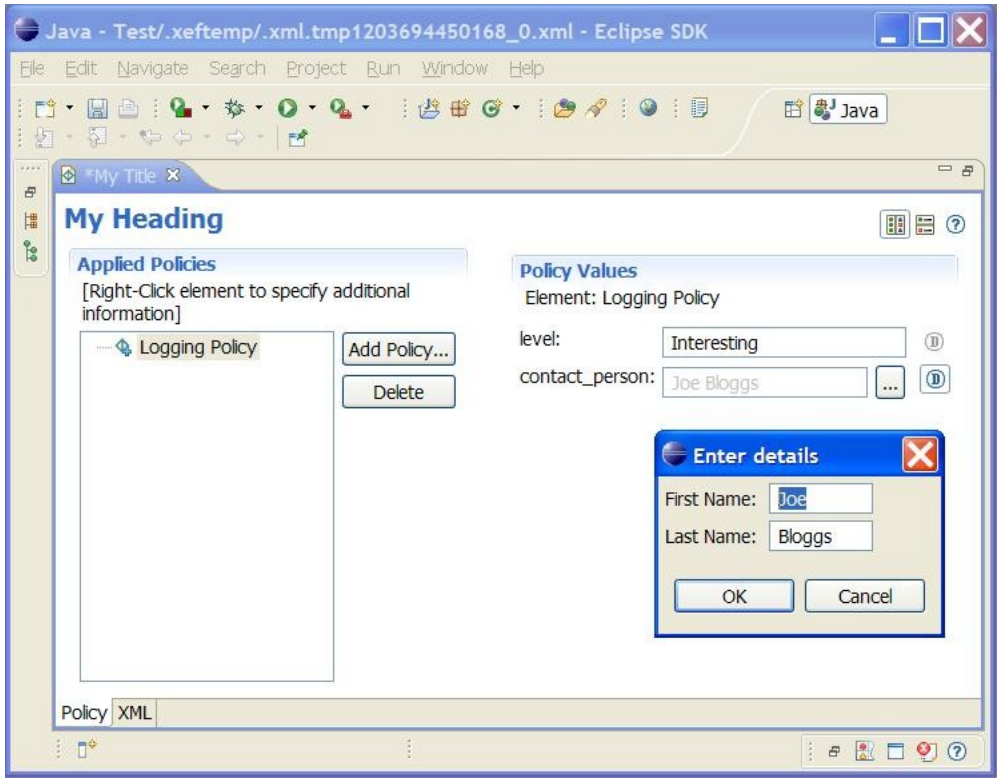
  Hopefully in IPolicyDetailEditorInput in Ganymede

 |

XEF Tutorial Part 3 – Custom field editors

Some field may need their own complex editors

- These can be plugged in via an Extension point
- Example:



 |

XEF Tutorial Part 3 – Custom field editors

XSD Attribute Definition:

```
<xs:attribute name="contact_person" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <xef:fieldEditor>nameFieldEditor</xef:fieldEditor>
    </xs:appinfo>
  </xs:annotation>
</xs:attribute>
```

Plug in field editor via Extension Point:

```
<extension point="org.eclipse.stp.xef.xefExtension">
  <fieldEditor class="org.eclipse.stp.xef.test.MyFieldEditor"
               fieldEditorId="nameFieldEditor">
  </fieldEditor>
</extension>
```

XEF Tutorial Part 3 – Custom field editor impl

```
public class MyFieldEditor extends AbstractFieldEditor {
    private Text firstName;
    private Text lastName;
    private String result;

    public MyFieldEditor() {
        super(null);
    }
    protected Control createDialogArea(Composite parent) {
        Composite area = (Composite) super.createDialogArea(parent);
        final GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 2;
        gridLayout.makeColumnsEqualWidth = false;
        area.setLayout(gridLayout);

        new Label(area, SWT.NONE).setText("First Name: ");
        firstName = new Text(area, SWT.BORDER);
        new Label(area, SWT.NONE).setText("Last Name: ");
        lastName = new Text(area, SWT.BORDER);
        return area;
    }

    protected void okPressed() {
        result = firstName.getText() + " " + lastName.getText();
        super.okPressed();
    }

    public String getFieldText() {
        return result;
    }

    // Some details ommitted, look at the
    // org.eclipse.stp.ui.xef.editor.QNameFieldEditor for a full example
}
```


 |

The end

Thank you for your attention

Any questions?

References

- Policy Editor Quick Start

  http://wiki.eclipse.org/STP/Policy_Component/Policy_editor_documentation
- XEF Reference Guide

  http://wiki.eclipse.org/STP/Policy_Component/XEF_Reference
- Latest info / getting the sources

  http://wiki.eclipse.org/STP
- Getting involved

  stp-dev@eclipse.org
- WS-Policy Standard

  http://www.w3.org/2002/ws/policy/
- Understanding WS-Policy processing

  http://www-128.ibm.com/developerworks/webservices/library/ws-policy.html

Policy Support in Eclipse STP
www.eclipse.org/stp

By Jerry Preissler & David Bosschaert

Agenda

- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies?
- How can you extend the STP policy editor ?
  - Exercise 3

What is a policy?

In a general context:

"a definite course or method of action selected from among alternatives and in light of given conditions to guide and determine present and future decisions"

www.merriam-webster.com

In a technical context:

A standardized description of the capabilities, requirements or general characteristics of an entity

based on WS-Policy 1.2

For automated processing, policies must posses some key traits

- a standardized, machine-processable syntax

  WS-Policy

- formal definitions for the actual properties that are expressed

  domain specific

  WS-Addressing, WS-RM Policy, WS-Atomic Transaction, WS-BusinessActivity, WS-SecurityPolicy

- a defined method to associate policies with policy subjects

  WS-PolicyAttachment

WS-Policy provides a syntax for expressing policies

```
<wsp:Policy Name="http://www.example.com/policies/P1"
      xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
      xmlns:wsp="http://www.w3.org/ns/ws-policy" >
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:SignedParts>
        <sp:Body/>
      </sp:SignedParts>
      <sp:EncrytedParts>
        <sp:Body/>
      </sp:EncryptedParts>
    </wsp:All>
    <wsp:All>
      <sp:SignedParts>
        <sp:Body/>
      </sp:SignedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Alternative 1

Alternative 2

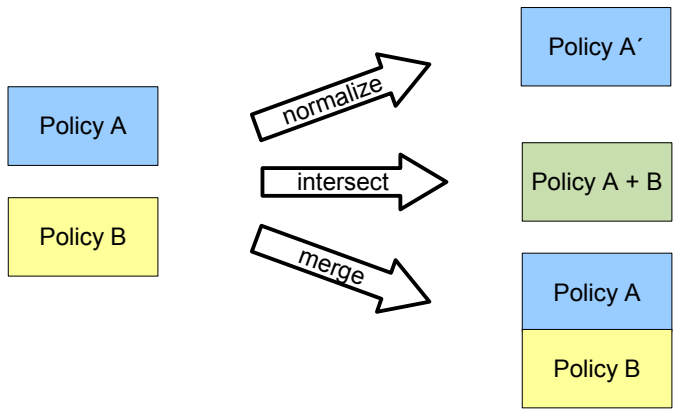
Policy

Alternative

Compositors

Assertions

WS-Policy provides operations to work with policies



Normalization transforms one policy in a defined format so the following statements are true:

· Every behavior that is compatible with Policy A is also compatible with Policy A'

· Every behavior that is not compatible with Policy A is also not compatible with Policy A'

· If two policies A and B describe an identical behavior, their normal form will be identical (modulo ordering of alternatives and assertions inside alternatives)

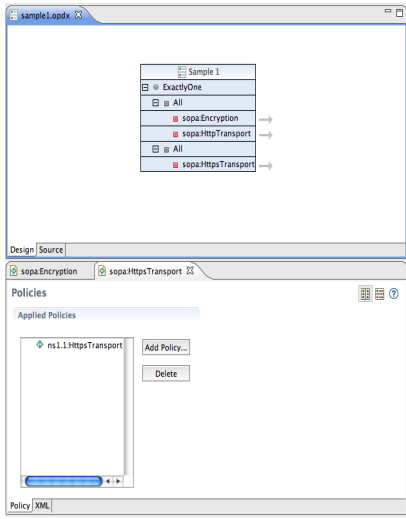
Intersection defines an operation that compares two input policies and returns a policy that contains the common alternatives

Merge is an operation that combines alternatives from two input policies. This operation is not specified by the standard, but some common implementations provide it

Agenda

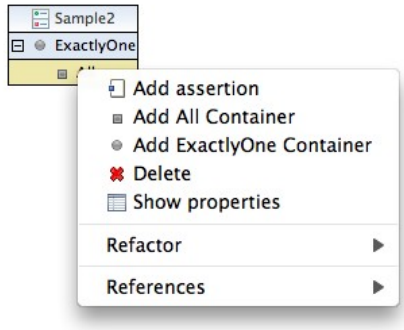
- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies?
- How can you extend the STP policy editor ?
  - Exercise 3

Policy Editor overview



- The policy editor provides two editor windows:
- The high level editor shows the complete structure of the policy
- The detail editor shows one selected policy assertion together with all attributes

The high level editor manipulates the structure of the policy



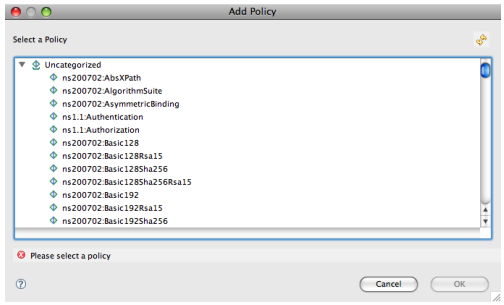
From the high level editor, you can

- add and remove compositors

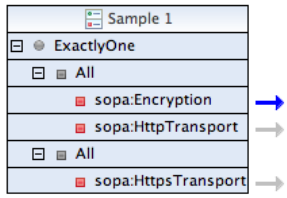
The high level editor manipulates the structure of the policy

From the high level editor, you can

- add and remove compositors
- add and remove individual assertions



The high level editor manipulates the structure of the policy

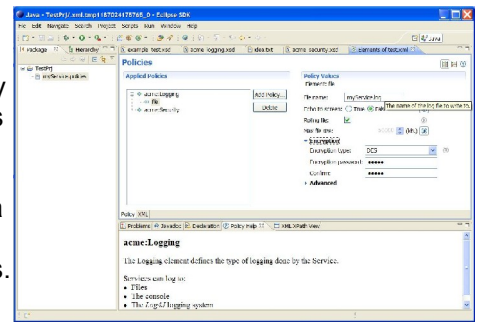


From the high level editor, you can

- add and remove compositors
- add and remove individual assertions
- switch to the detail editor to work with an individual assertion

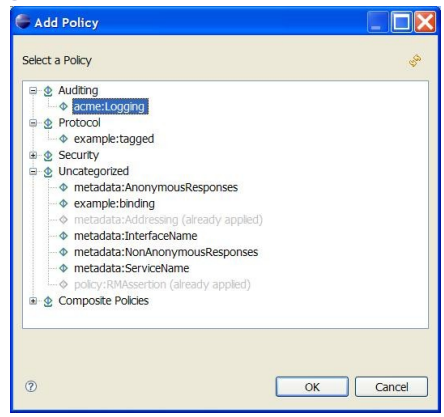
Details Editor

- Similar in look & feel to PDE Extension Point editor
- Can edit the details of WS-Policy assertions as well as other types of XML files that contain embedded elements.
- Editor dynamically synthesizes a GUI based on the schema definition of the policy assertions.
- GUI works with most standard XML Schema definitions
- Based on XEF (also part of STP)



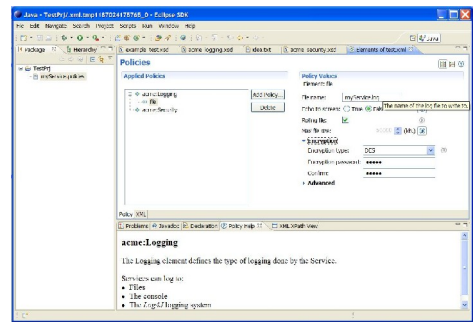
Details Editor – Policy Catalogue

- When editing policies, new ones can be added from a Catalogue.
- The catalog has a simple interface
  - Can serve policies from local filesystem
  - Look up catalog in database
  - WTP XMLSchema Catalog integration



Details Editor – Features

- Widgets for many XSD data types
- Display names, tooltips
- Context-sensitive help
- Display of defaults values
- Required fields
- Enumerated values
- Password fields
- Representation of xs:choice, xs:sequence and xs:any
- Much more...



Details editor – What is being edited

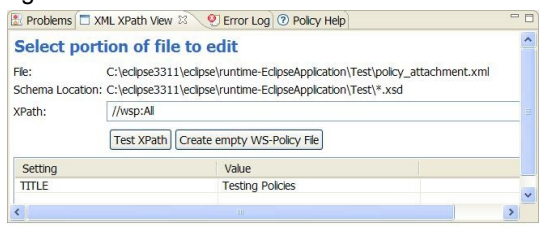
- You can view/edit the source XML too, could look like this (WTP XML Editor):

eclipseCON™ 2008

Details editor – Launching

- Current use is primarily embedded in applications, launching is done by opening an editor by calling `IDE.openEditor()` with a
  `org.eclipse.stp.policy.common.editors.IPolicyDetailEditorInput`
  or
  `org.eclipse.stp.ui.xef.editor.XMLProviderEditorInput`
  Editor ID: `org.eclipse.stp.ui.xef.editor.XefEditor`

- For testing there's the XML XPath View:



- It allows you to specify a policy file, what part in the file needs to be edited (as XPath), settings and then open the editor

So the XML XPath View is really just a way to open the editor without having to create an IPolicyDetailEditorInput object

Agenda

- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies?
- How can you extend the STP policy editor ?
  - Exercise 3

XEF Tutorial part 1 – editing policies

Summary:

Open the details editor directly, using the XML XPath View, to edit:

- WS-PolicyAttachment file
- Embedded WS-Policies in a WSDL file
- A CXF Configuration file (non-WS-Policy)

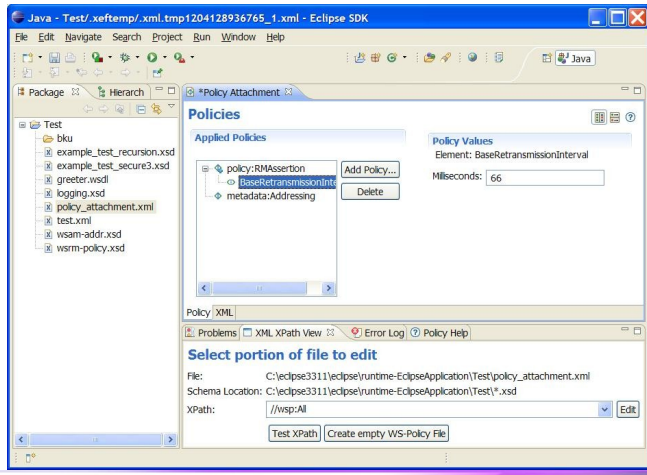
XEF Tutorial part 1 – editing policy documents

Exercises:

1. Add a ws-addressing policy to the policy_attachment.xml file
   - Use the XML XPath view to edit the policies in policy_attachment.xml
   - XPath: `//wsp:All`
2. Open the provided greeter.wsdl file and change the retransmission timeout of the WS-RM policy to 70000.
   - Use the XML XPath view to edit the policies in greeter.wsdl
   - XPath: `//*[local-name()='Policy' and namespace-uri()='http://www.w3.org/ns/ws-policy']`
3. Edit features in a cxf-features.xml file.
4. Stretch exercise – open the editor from code on a memory object (which has no file).

XEF Tutorial part 1 – editing policy documents

The editor will look like this:



XEF Tutorial Part 2 – Create your own Policy Type

Summary:

- Create your own logging policy definition
- Use the policy
- Make it look nice

XEF Tutorial Part 2 – Create your own Policy Type

Exercise:

Create a new logging policy of which an instance looks like this:

```
<acme:Logging xmlns:acme="http://www.acme.com/xsd/2007/08/logging">
  <file filename="mylogfile.log" write_interval="5000" echo="true" />
</acme:Logging>
```

- With two sub-elements: console logger and file logger
- File logger has:
  - a required field ‘filename’
  - echo to screen field (boolean)
  - a write interval (default: 10000 ms)

XEF Tutorial Part 2 – Create a Basic Logging Policy

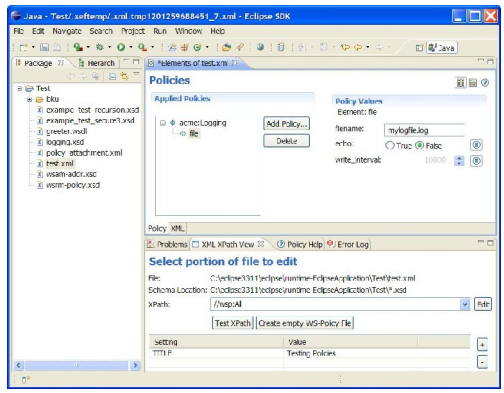
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xef="http://schemas.eclipse.org/stp/xsd/2006/05/xef"
  xmlns:xefgui="http://schemas.eclipse.org/stp/xsd/2006/05/xef/gui"
  targetNamespace="http://www.acme.com/xsd/2007/08/logging"
  xmlns:tns="http://www.acme.com/xsd/2007/08/logging">

  <xs:element name="Logging">
    <xs:complexType>
      <xs:choice>
        <xs:element name="file" type="tns:fileLoggingType"/>
        <xs:element name="console" />
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="fileLoggingType">
    <xs:attribute name="filename" type="xs:string" use="required"/>
    <xs:attribute name="echo" type="xs:boolean" default="false"/>
    <xs:attribute name="write_interval" type="xs:positiveInteger"
      default="10000" />
  </xs:complexType>
</xs:schema>
```

eclipseCON™ 2008

XEF Tutorial Part 2 – Use the new Policy

- Add this logging.xsd to your current Project.
- Edit a policy document, e.g. `test.xml`
- Add the logging policy
- Add the `file` subelement

Raw, but functional editor →



The XML XPath view uses all the XSD files in the current project as the schema catalogue.

The editor created from the raw schema is functional:

•You can add the logging element

•You can add file/console subelements (adherence to xs:choice/xs:sequence)

•Filename is required as per schema

•Echo is a boolean as per schema

•Write_interval is a numeric value as per schema (default visible).

XEF Tutorial Part 2 – Prettify the new policy in the UI

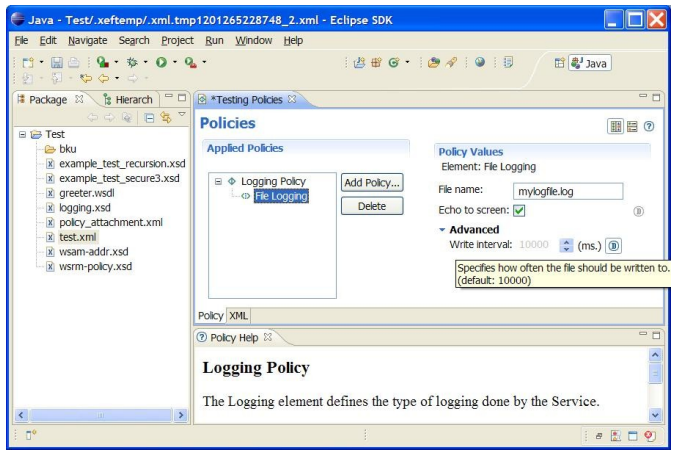
When every thing works, make it look nice

- Policy in ‘Audit’ category and is called ‘Logging’
- Make sure all labels have nice display names
- Put write_interval in an advanced section
- Add a ‘milliseconds’ unit to write_interval
- Make sure everything has tooltips and documentation

eclipseCON™ 2008

XEF Tutorial Part 2 – Prettify the UI

Using the XEF reference, add annotations to make the policy look nice:



- The reference guide is also here: http://wiki.eclipse.org/STP/XEF_Reference

Quite a few things have changed here:

- There is documentation with the policy (from <xs:documentation> annotation)
- Elements have display names
- Tool tips
- Write Interval is in an advanced section and units are displayed.
- Echo to screen is a tickbox (instead of a radio button).

Note that the core of the XML-Schema has not changed. This is done with just extra annotations in it.

The resulting XML file that is being edited is also the same as before, so this is just improving the usability of the editor.

XEF Tutorial Part 2 – Influencing the UI

- An enhanced version with more UI features of this tutorial is available on the STP Wiki: http://wiki.eclipse.org/STP/Policy_editor_documentation
- Logging schemas available:
- Basic logging file is called: logging_basic/logging.xsd
- Final logging file is called: logging_full/logging.xsd

Agenda

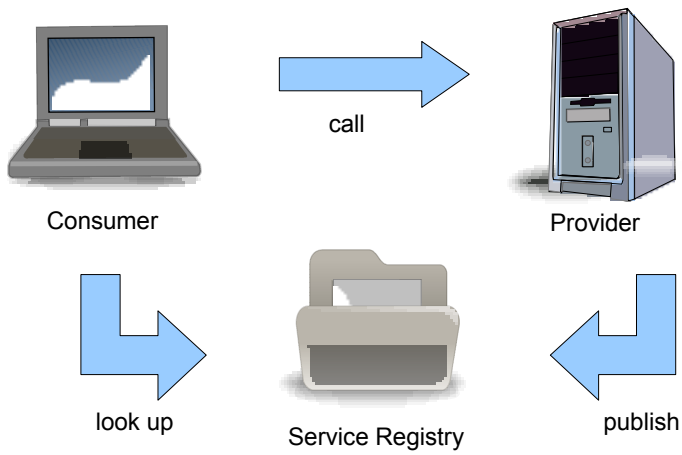
- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies?
- How can you extend the STP policy editor ?
  - Exercise 3

Policy-driven mechanisms can be used to enhance the functionality provided by an SOA

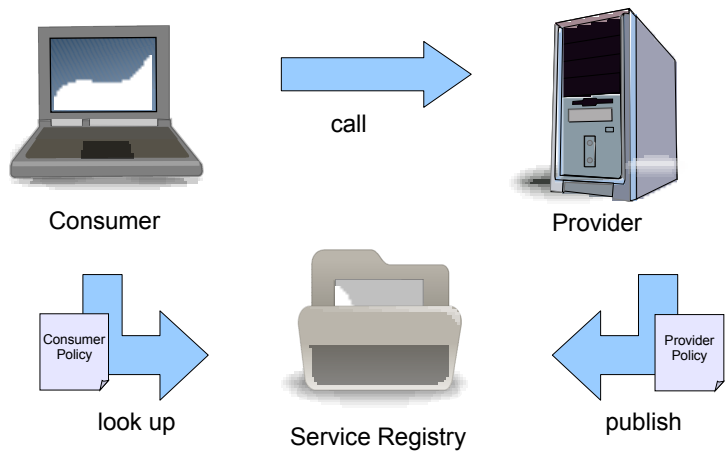
- SOA uses a Service Registry to provide a level of indirection between the service consumer and the service provider
- Non-functional properties of consumers and providers alike can be specified with policies

The functionality of an SOA can be enhanced by including policy-driven negotiation into the service provider lookup process

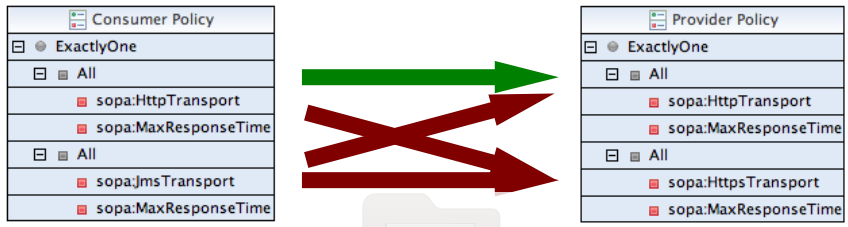
SOA provides a level of indirection between consumer and provider



Policies can be integrated in the lookup mechanism



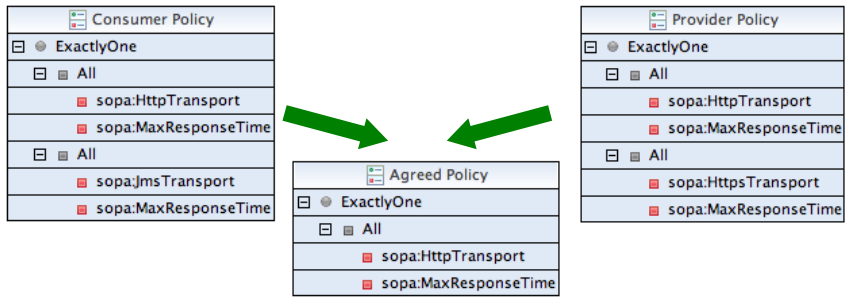
Policies can be integrated in the lookup mechanism



Alternatives are compared crosswise between policies
Non-matching alternatives are rejected
Matching alternatives are included in an “Agreed Policy”

Service Registry

The resulting policy captures the properties that are common to both participants



Matching alternatives are included in an “Agreed Policy”

Policies can be used to control a wide range of behavior

Technical concerns

- Transport selection
- Location-based routing
- Message tracking

Policies can be used to control a wide range of behavior

Technical concerns

Security aspects

- Authentication
- Authorization
- Encryption
- Signature

Policies can be used to control a wide range of behavior

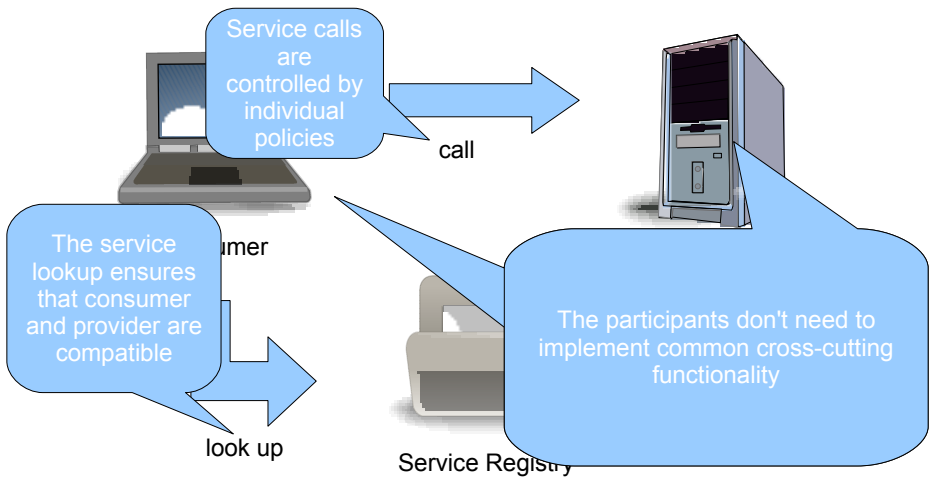
Technical concerns

Security aspects

Quality of service

- Response time
- Reliability
- Cost

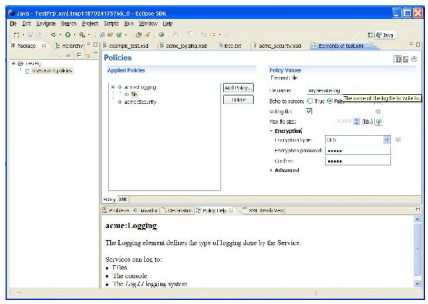
The policy-driven mechanism enhances SOA functionality in three important ways



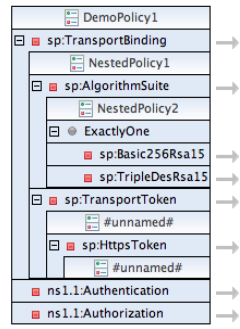
Agenda

- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies ?
- How can you extend the STP policy editor ?
  - Exercise 3

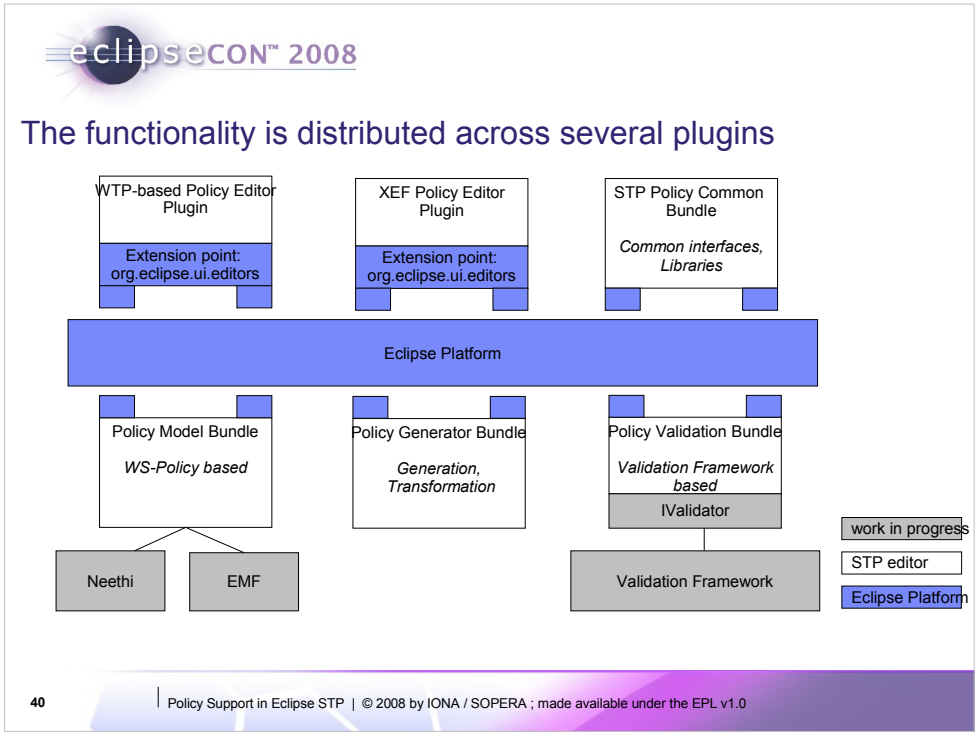
The STP policy editor combines two main contributions



The XEF-based editor was contributed by IONA



The WTP-based editor was contributed by SOPERA



1. WTP-based Policy Editor Plugin: WTP-based editor functionality; policy alternatives rendering; activation of XEF editor for assertions.
2. XEF Editor Plugin: representation of assertions based on XML schema; assertion editing and saving; communication with WTP-based editor via callback interface
3. Common STP Policy Bundle: contains common interfaces (IPolicyDetailEditorInput); common libraries
4. Policy Model Bundle: implementation of policy model, abstract interface for Policy Editor (support Neethy, EMF or internal model implementations). (Neethy, EMF – in progress)
5. Policy Generator Bundle: policy generation and transformation functionality
6. Policy Validation Bundle: ws-policy validator bundle based on Validation Framework (in progress)

Agenda

- What is a policy ?
- How can you work with the STP policy editor ?
  - Exercise 1 + 2
- What can you do with policies ?
- How can you extend the STP policy editor ?
  - Exercise 3

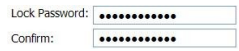
XEF Tutorial Part 3 – XEF extension points

Summary:

- Text filters for password fields
- Callback for populating value sets
- Custom field editors

XEF Tutorial Part 3 – Text Filters for Passwords

Password fields can use custom filters to process the value:



XSD Attribute Definition:

```
<xs:attribute name="lock_password" type="xs:string" use="required">
  <xs:annotation>
    <xs:appinfo>
      <xef:displayName>Lock Password</xef:displayName>
      <xef:filter>MyFilter</xef:filter>
      <xefgui:widget>password</xefgui:widget>
    </xs:appinfo>
  </xs:annotation>
</xs:attribute>
```

Plug in filters via Extension Point:

```
<extension
   point="org.eclipse.stp.xef.xefExtension">
   <filter class="org.example.MyFilter"
           filterId="MyFilter" />
</extension>
```

MyFilter (reverses pwd in document):

```
package org.example;

import org.eclipse.stp.ui.xef.editor.TextFilter;

public class MyFilter implements TextFilter {
    public String filter(String data) {
        return new StringBuilder(data).
                    reverse().toString();
    }
}
```

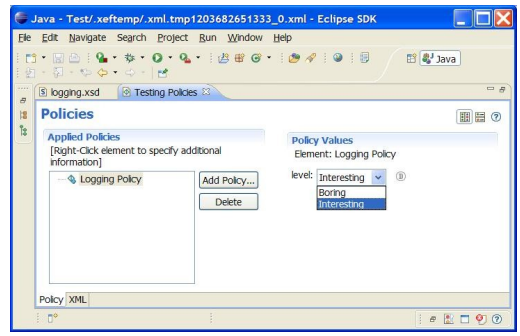
XEF Tutorial Part 3 – Value Proposal Callbacks

- You might want users to select from a prepopulated set of values

  Possible through XSD enumeration:

```
<xs:attribute name="level" default="Info">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Fatal"/>
      <xs:enumeration value="Error"/>
      <xs:enumeration value="Warning"/>
      <xs:enumeration value="Info"/>
      <xs:enumeration value="Debug"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

- Maybe you need a more dynamic approach, where possible values are fed from your application.



XEF Tutorial Part 3 – Value Proposal Callbacks (impl)

- A more dynamic approach is via a callback, declared in XSD:

```
<xs:attribute name="level" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <xefgui:context>
        <xefgui:values>loglevels</xefgui:values>
      </xefgui:context>
    </xs:appinfo>
  </xs:annotation>
</xs:attribute>
```

- Realized through an IContextProvider

```
IContextProvider myCtxProvider = new IContextProvider() {
    public Object getData(String ctxId) {
        return null;
    }
    public String[] getValues(String ctxId, String ctxFilter) {
        if ("loglevels".equals(ctxId)) {
            return new String [] {"Boring", "Interesting"};
        }
        return null;
    }
};
new XMLProviderEditorInput(settings, selectedFile.getProject(),
        new XPathXMLProvider(...), schemaProvider, myCtxProvider),
```

- Currently only supported via XMLProviderEditorInput

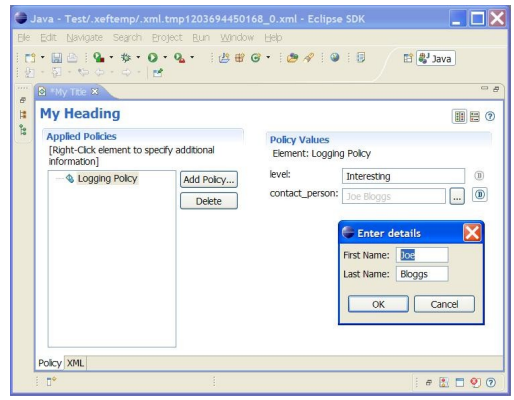
  Hopefully in IPolicyDetailEditorInput in Ganymede

There are hooks to re-evalutate the values based on the state of other fields, see reference guide.

XEF Tutorial Part 3 – Custom field editors

Some field may need their own complex editors

- These can be plugged in via an Extension point
- Example:



 |

XEF Tutorial Part 3 – Custom field editors

XSD Attribute Definition:

```
<xs:attribute name="contact_person" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <xef:fieldEditor>nameFieldEditor</xef:fieldEditor>
    </xs:appinfo>
  </xs:annotation>
</xs:attribute>
```

Plug in field editor via Extension Point:

```
<extension point="org.eclipse.stp.xef.xefExtension">
  <fieldEditor class="org.eclipse.stp.xef.test.MyFieldEditor"
               fieldEditorId="nameFieldEditor">
  </fieldEditor>
</extension>
```

XEF Tutorial Part 3 – Custom field editor impl

```
public class MyFieldEditor extends AbstractFieldEditor {
    private Text firstName;
    private Text lastName;
    private String result;

    public MyFieldEditor() {
        super(null);
    }
    protected Control createDialogArea(Composite parent) {
        Composite area = (Composite) super.createDialogArea(parent);
        final GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 2;
        gridLayout.makeColumnsEqualWidth = false;
        area.setLayout(gridLayout);

        new Label(area, SWT.NONE).setText("First Name: ");
        firstName = new Text(area, SWT.BORDER);
        new Label(area, SWT.NONE).setText("Last Name: ");
        lastName = new Text(area, SWT.BORDER);
        return area;
    }

    protected void okPressed() {
        result = firstName.getText() + " " + lastName.getText();
        super.okPressed();
    }

    public String getFieldText() {
        return result;
    }

    // Some details ommitted, look at the
    // org.eclipse.stp.ui.xef.editor.QNameFieldEditor for a full example
}
```

The end

Thank you for your attention

Any questions?

References

- Policy Editor Quick Start
  http://wiki.eclipse.org/STP/Policy_Component/Policy_editor_documentation
- XEF Reference Guide
  http://wiki.eclipse.org/STP/Policy_Component/XEF_Reference
- Latest info / getting the sources
  http://wiki.eclipse.org/STP
- Getting involved
  stp-dev@eclipse.org
- WS-Policy Standard
  http://www.w3.org/2002/ws/policy/
- Understanding WS-Policy processing
  http://www-128.ibm.com/developerworks/webservices/library/ws-policy.html