WebSphere ESB

WebSphere Enterprise Service Bus V6.1 Update

Phil Coxhead
ISSW Worldwide Technology Practise
phil_coxhead@uk.ibm.com

4th March 2008

SOA on your terms and our expertise

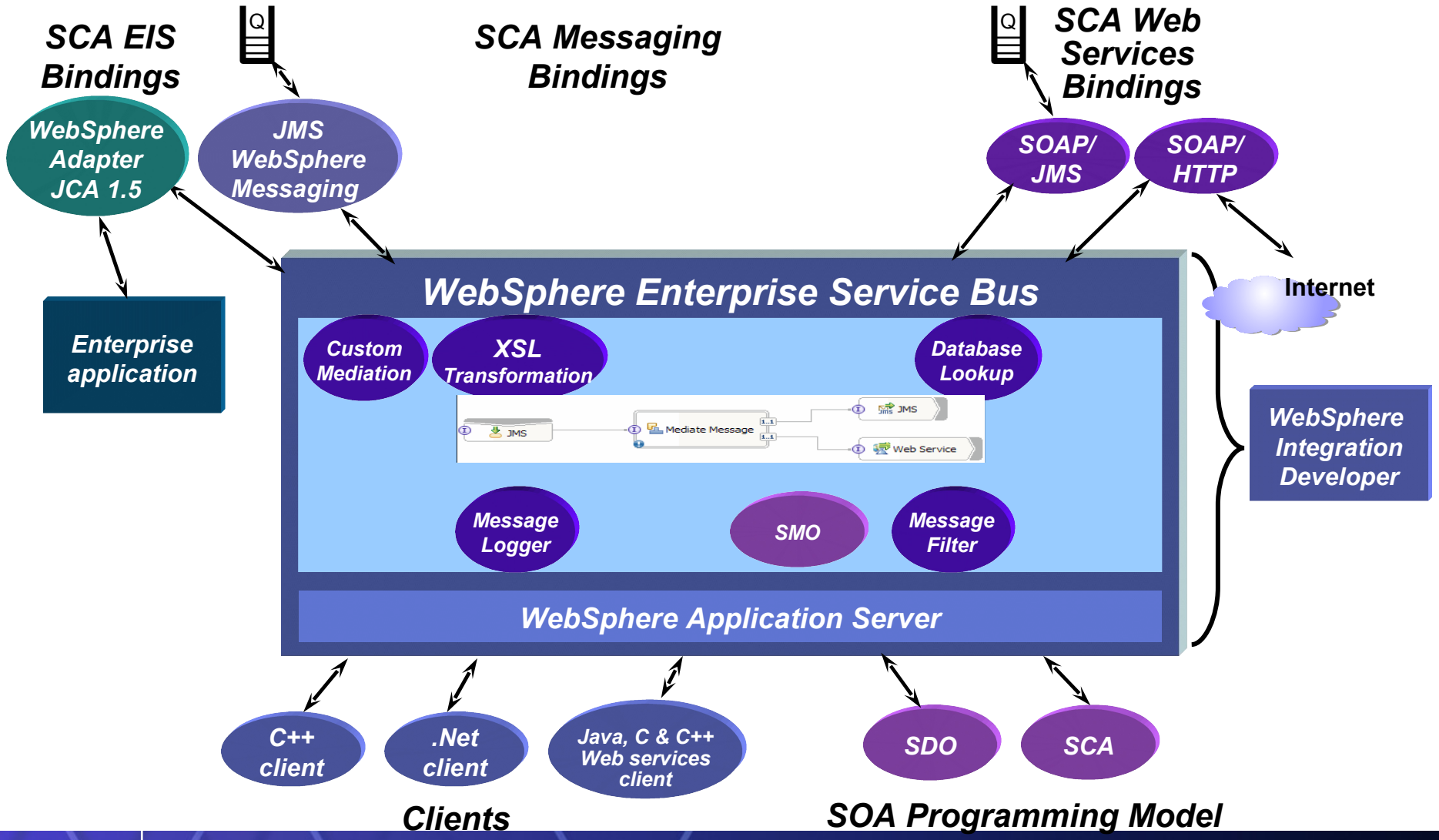
ON DEMAND BUSINESS™

Agenda

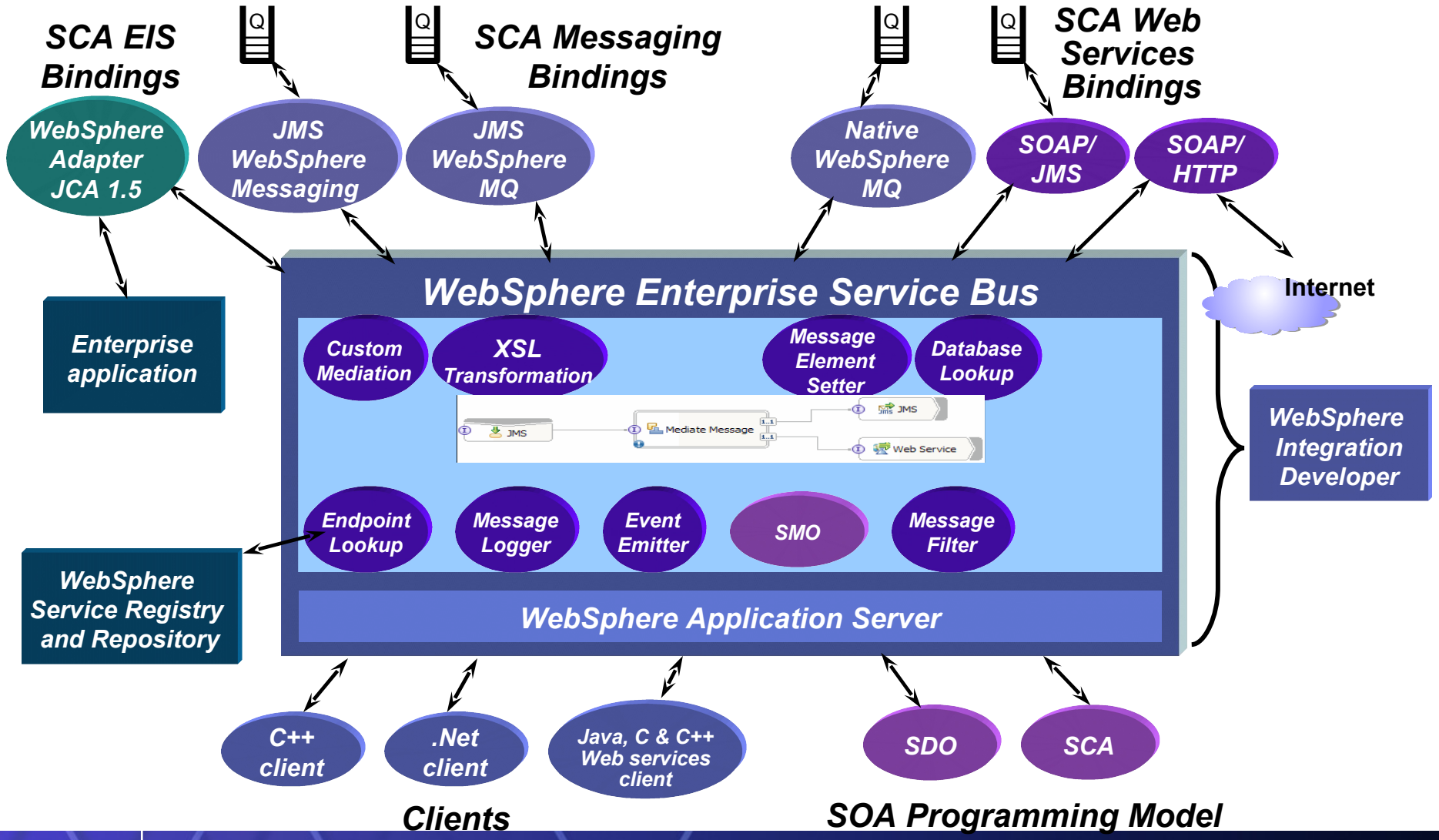
- Product Overview – a review
- Enhancements to existing features
- New features
- Enhancements to developer tooling

Review of V6.0.1 and V6.0.2 functionality

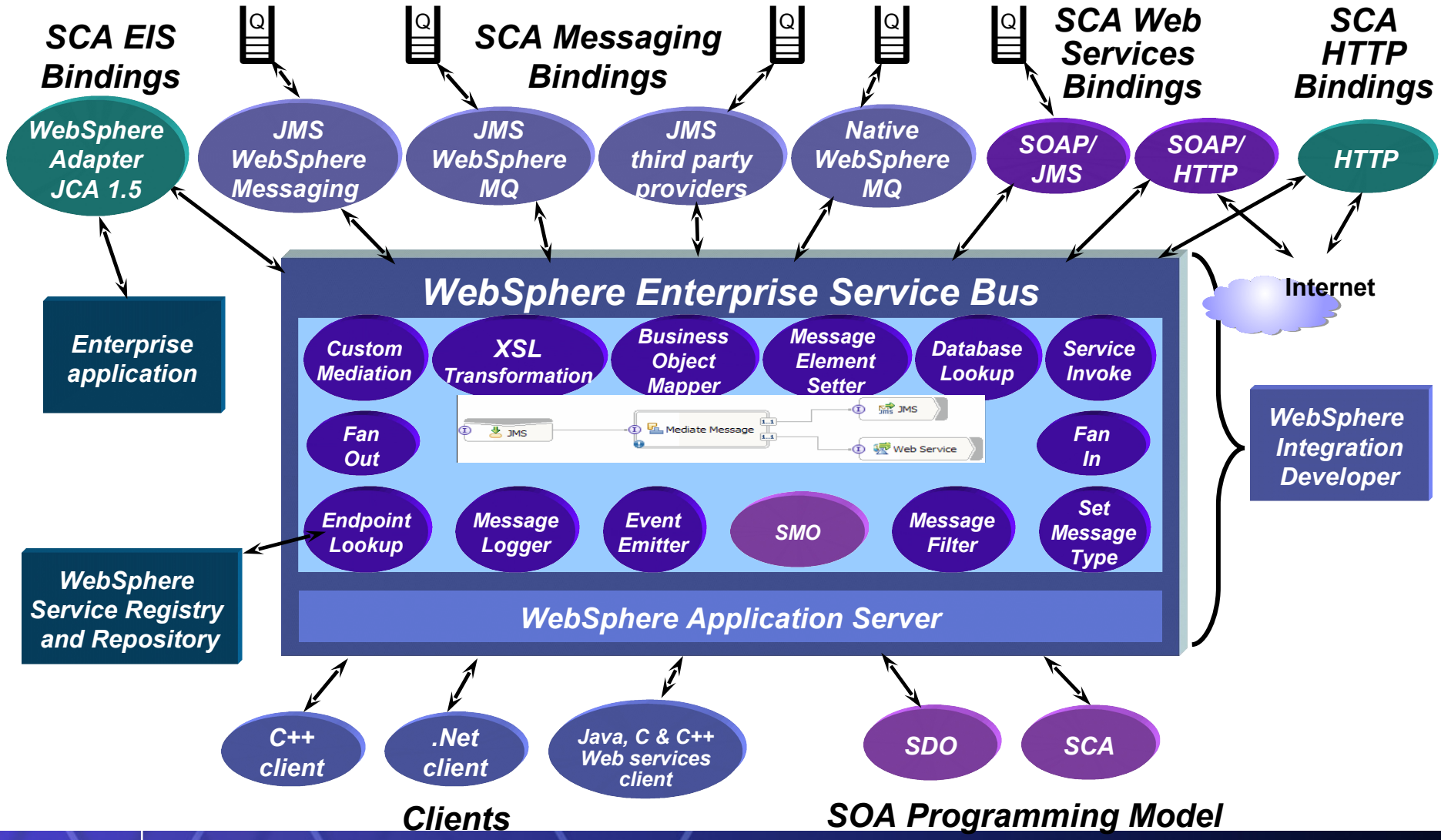
WebSphere ESB V6.0.1



WebSphere ESB V6.0.2



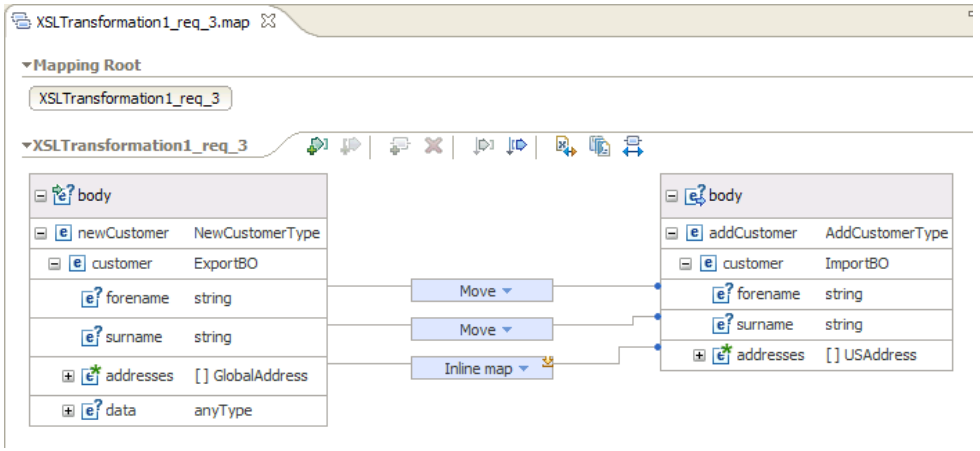
WebSphere ESB V6.1

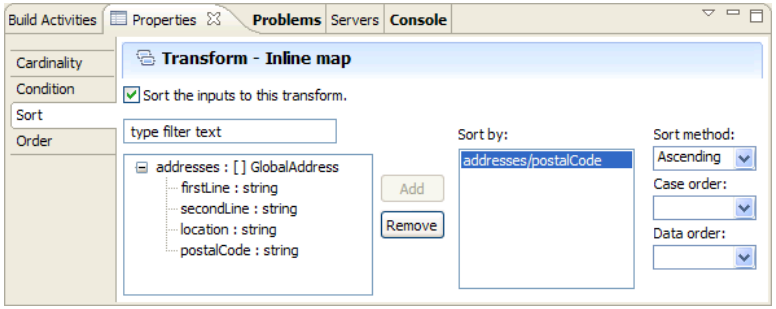


Enhancements to existing features

XSLT primitive enhancements

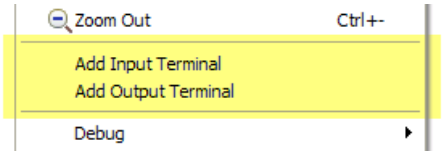
- Mapping editor based on RAD V7
- Creation of XML test input
- Reuse of maps
  - across primitives
  - as sub-maps for handling complex elements
- Other mappings/refinements
  - Move – copy primitive type
  - Concat – concatenate multiple strings
  - Custom
    - XPath
    - Java – call static methods
    - XSLT – call external template
  - Inline map – nested map only usable in current map file
  - Substring – returns substring based on delimiter and index
- Arrays can be sorted and either iterated over or specify cardinality
- Conditional transformation



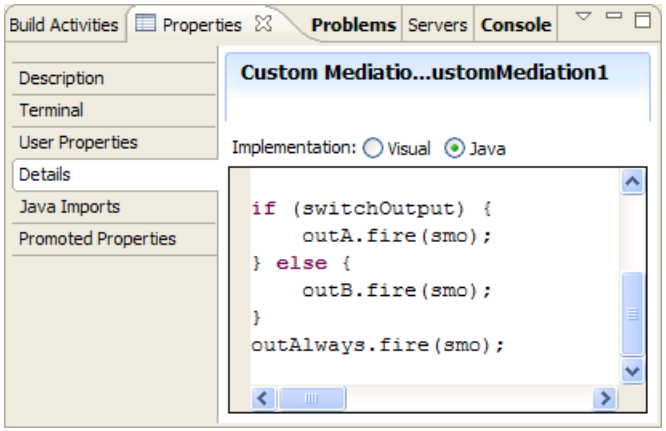


Custom mediation primitive enhancements

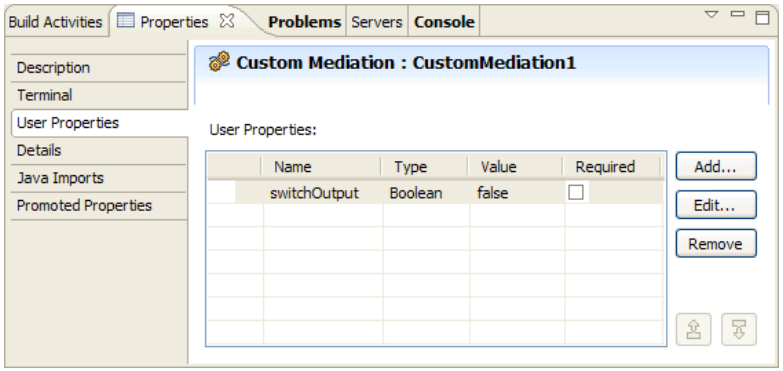
- Extends some of capabilities of user-defined primitive to custom mediation
  - One or more input terminals
  - Zero or more output terminals
  - Control over when output terminals fired
  - Ability to define (promotable) properties



- Separate panel for Java imports



- Ability to use SCA component as implementation removed – use new service invoke primitive
- Quick fix provided to migrate Java/Visual implementation to 6.1

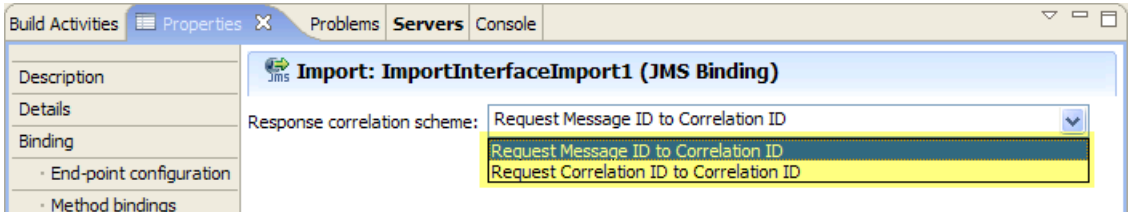


Message logger primitive enhancements

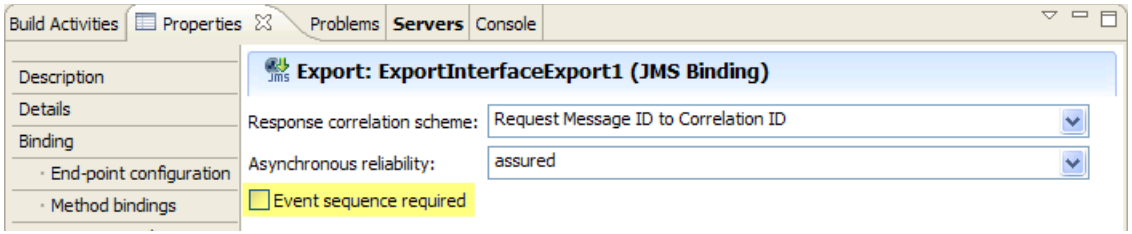
- Now logs to CommonDB by default for both single server and ND
- Script for generating resources now available for all supported databases
  - <WESB_HOME>/dbscripts/EsbLoggerMediation
- Schema name now overridden via ESB_MESSAGE_LOGGER_QUALIFIER environment variable

JMS binding enhancements

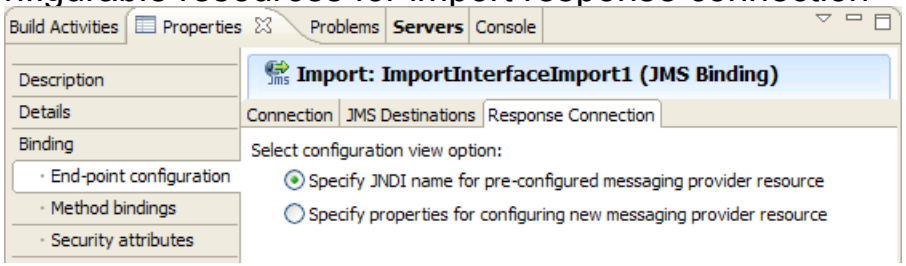
- JMS bindings brought in to line with WebSphere MQ JMS binding
  - Configurable correlation schemes for imports and exports



  - Event sequencing for exports

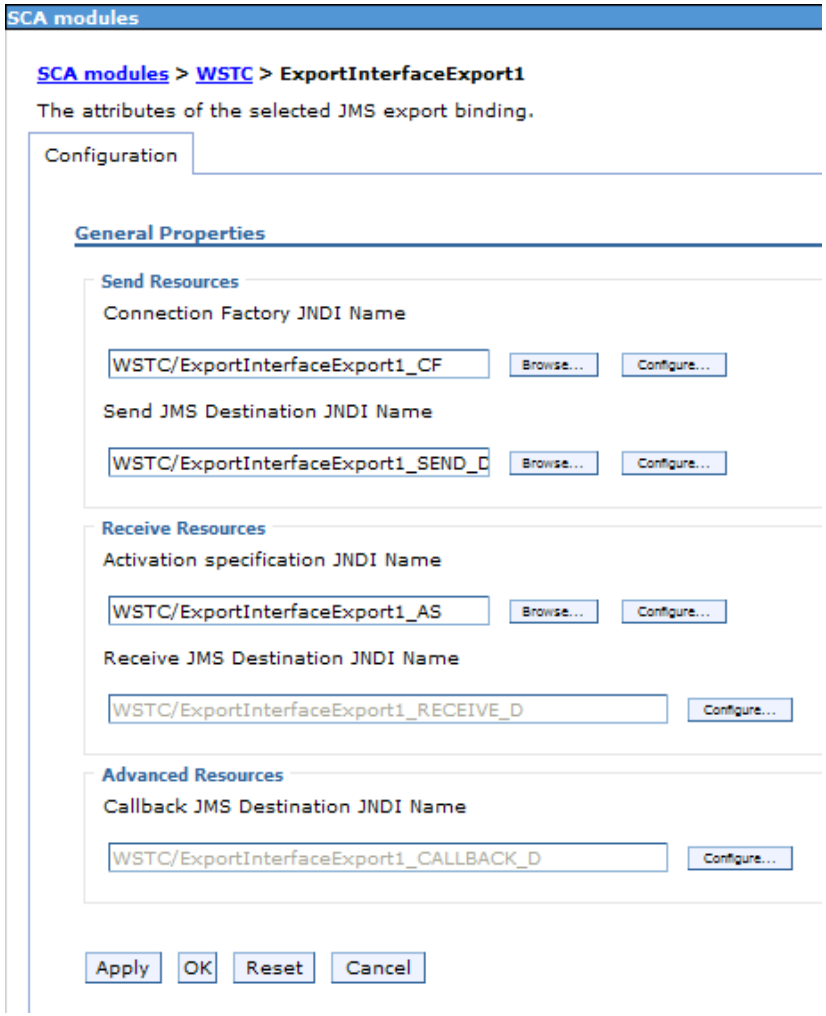


  - Configurable resources for import response connection



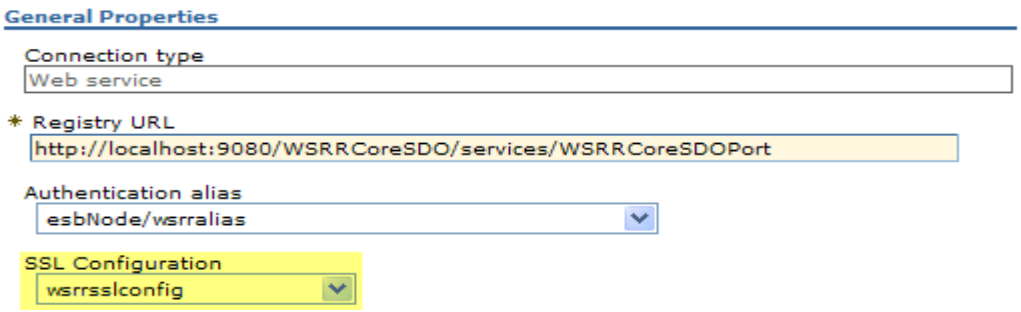
Administration of bindings

- Runtime administration previously only available for SCA and web service bindings
- Now also available for Generic JMS, JMS, MQ JMS and Native MQ (and HTTP) bindings
- Browse and configure destinations, connection factories, activation specs and listener ports
- AdminTask commands: (show|modify)SCA(Import|Export)(MQ|JMS)Binding

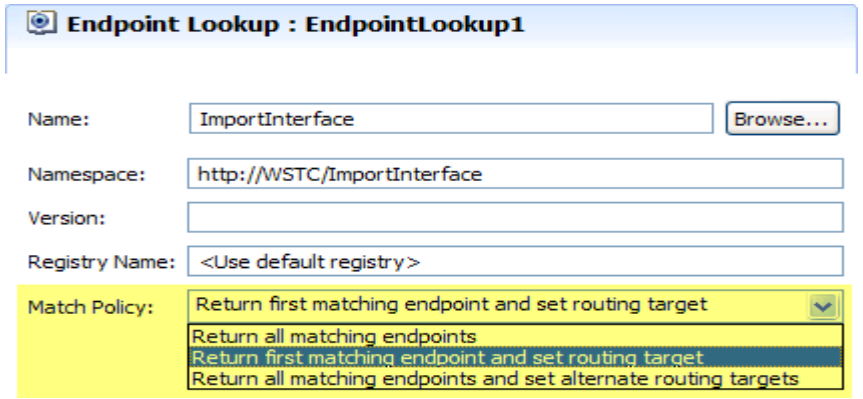


Endpoint lookup primitive

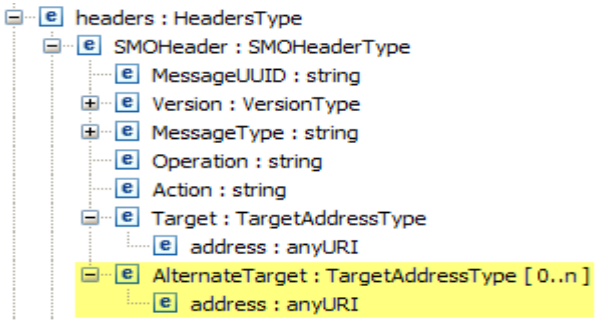
- SSL repertoire configured on WSRR definition



- New match policy:
  - *Return all matching endpoints and set alternate routing targets*

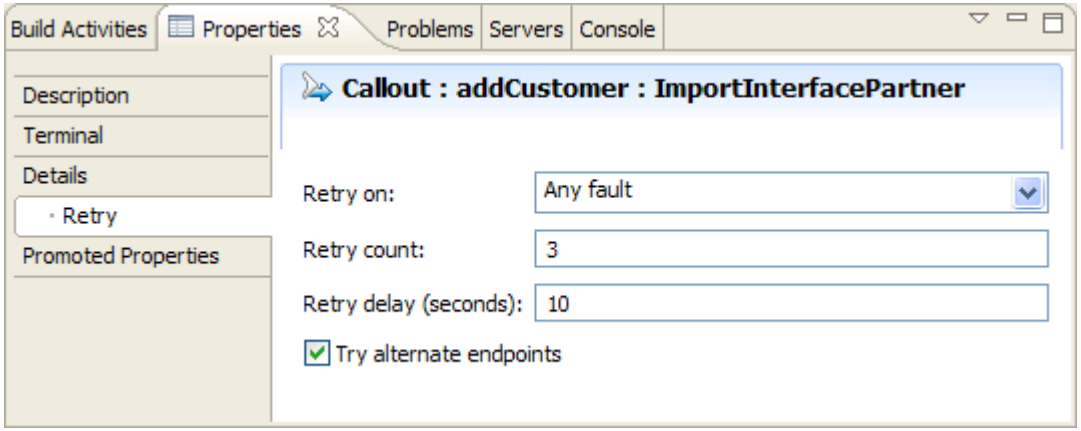


- Policy sets new *AlternateTarget* element in SMO Header



Retry on callout

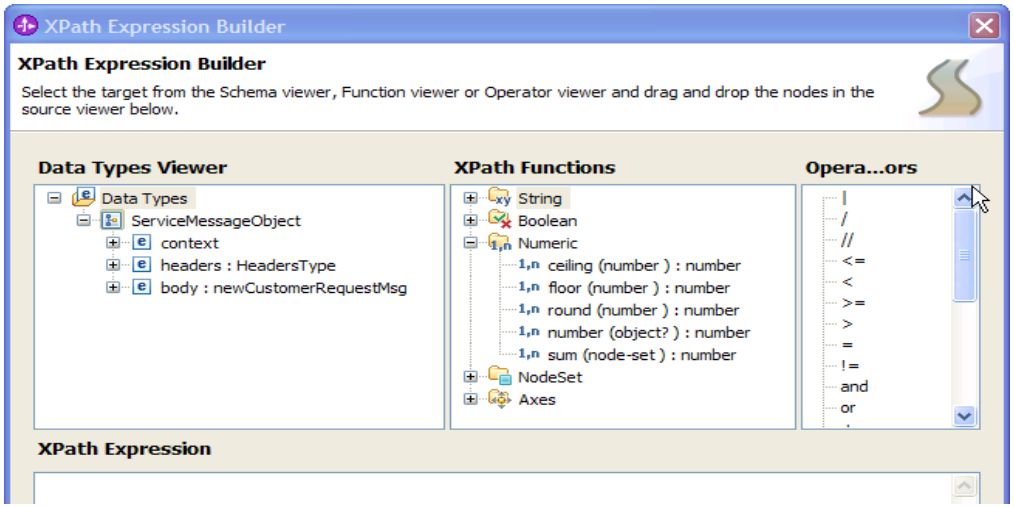
- Enables retry attempts on failure to invoke referenced service



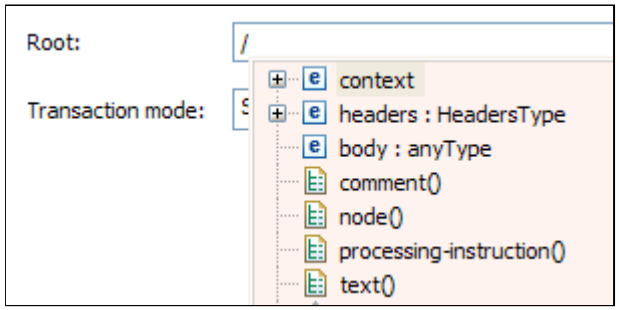
- ***Retry on***: ***Never*** | *Any fault* | *Modeled fault* | *Unmodeled fault*
- ***Retry count***: number of attempts after initial failure
- ***Retry delay***: delay in seconds between attempts (from timeout for asynchronous requests)
- ***Try alternate endpoints***: alternate endpoints from SMO header tried in order, wrapping around until retry count reached

XPath tooling enhancements

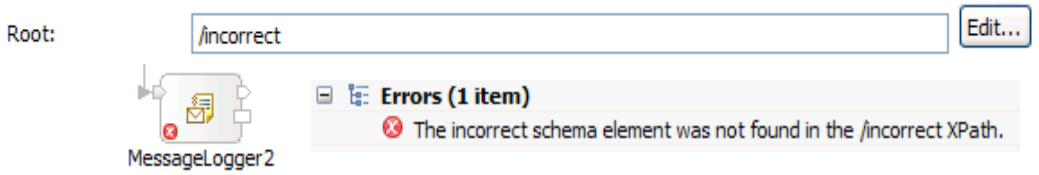
- Enhanced expression builder



- Content Assist



- Expression validation



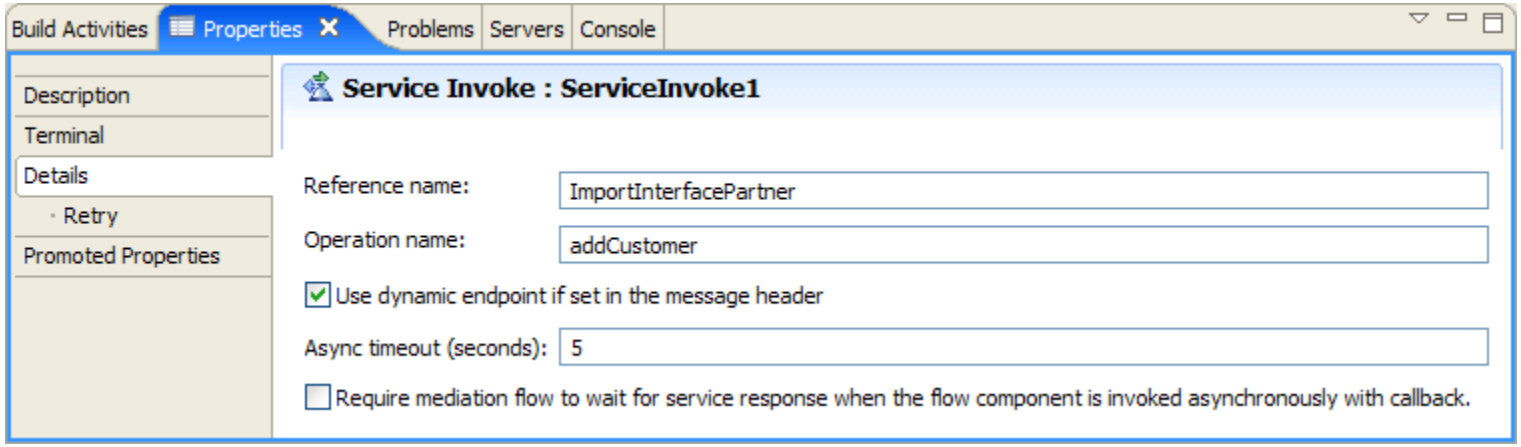
New runtime features

Deployment Environments

- Concept of deployment environments introduced to ease creation of ND topologies
- Three patterns supported out of the box:
  - Single cluster
  - Remote messaging
  - Remote messaging and remote support
- Clusters provide the following functions
  - Application deployment target
  - Messaging infrastructure
  - Support infrastructure
- Deployment environments can be created:
  - During installation
  - During profile creation
  - Via the administrative console
- Capability to view status and start and stop deployment environment

Service invoke primitive

- Invokes a service operation from within a request or response flow
- Request passed to *in* terminal; response flows from *out* or *fail*

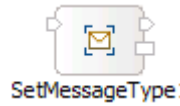


- *Reference name* and *Operation name*: determine operation to invoke – specified on creation
- ***Use dynamic endpoint if set in the message header***
- ***Async timeout*** : time to wait in seconds for an asynchronous response (or -1 for infinite wait) before firing *timeout* terminal; treated like unmodeled fault with respect to retry
- *Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback*
- Same **retry options** as callout

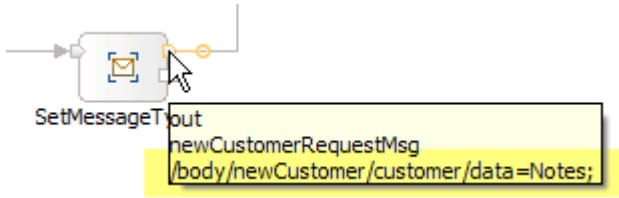
Service invoke – invocation style

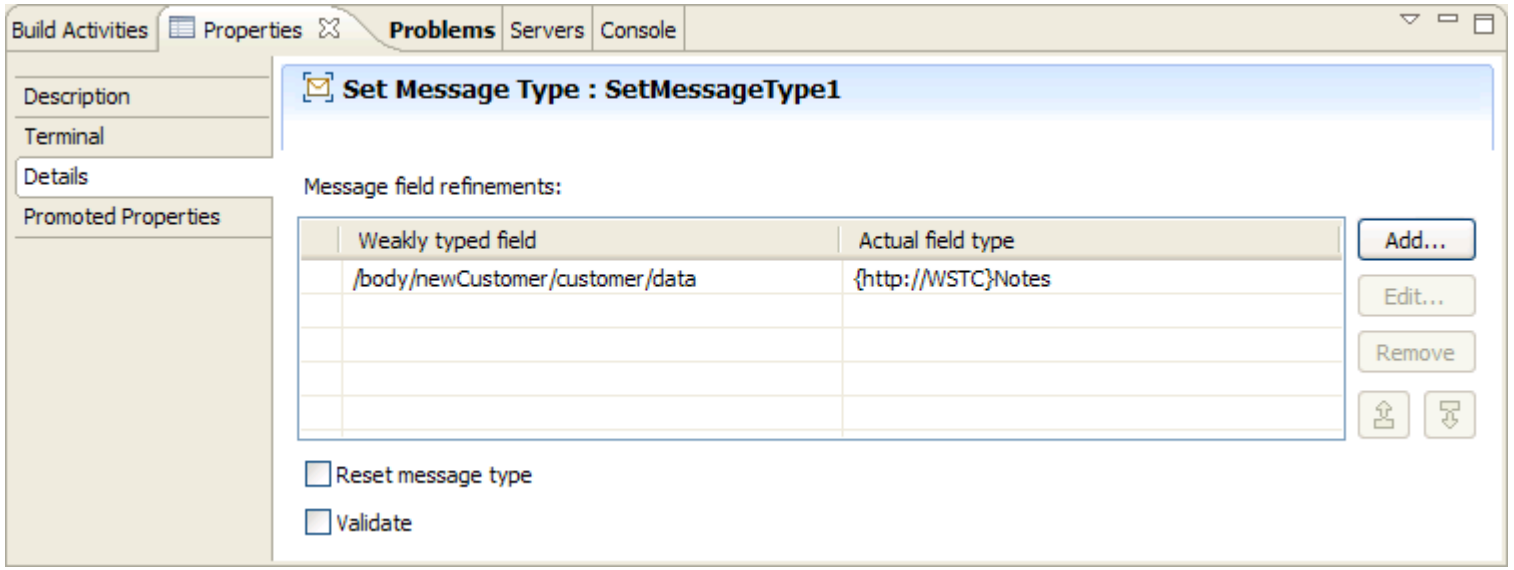
Mediation Flow Component Invocation Style	Target Preferred Interaction Style	One Way / Request-Response Operation	Force Sync	Invocation style used for Service Invoke
invoke (synchronous)	ANY	One Way	any	Async
		Request-Response	any	Sync
	SYNC	both	any	Sync
	ASYNC	both	any	Async
invokeAsync	ANY	One Way	any	Async
		Request-Response	any	Sync
	SYNC	both	any	Sync
	ASYNC	both	any	Async
invokeAsyncWithCallback	ANY	One Way	any	Async
		Request-Response	True	Async
			False	AsyncWithCallback
	SYNC	both	any	Sync
	ASYNC	One Way	any	Async
		Request-Response	True	Async
			False	AsyncWithCallback

Set message type primitive



- Indicates down cast for weakly typed fields (e.g. derived types or xsd:any) in SMO to enable manipulation by tooling
- Message type augmented with additional type information propagated from *out* terminal

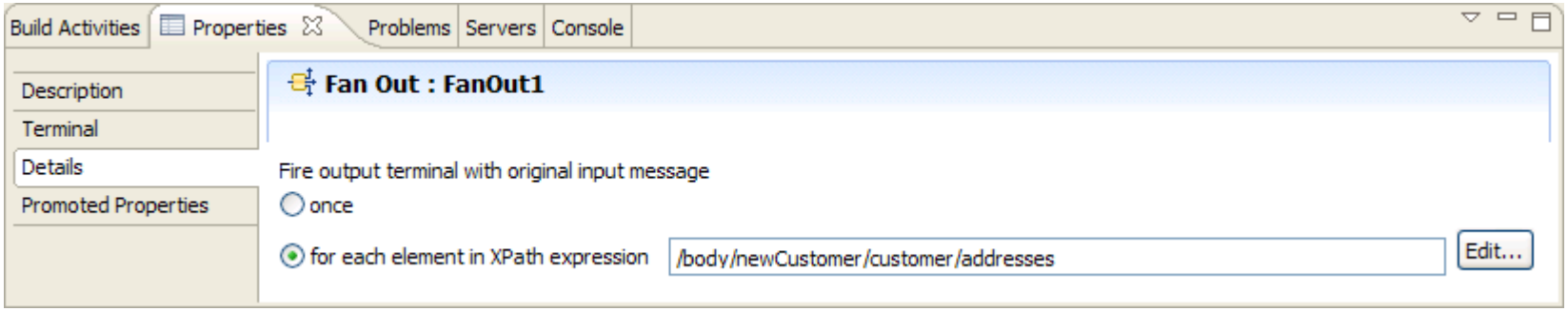




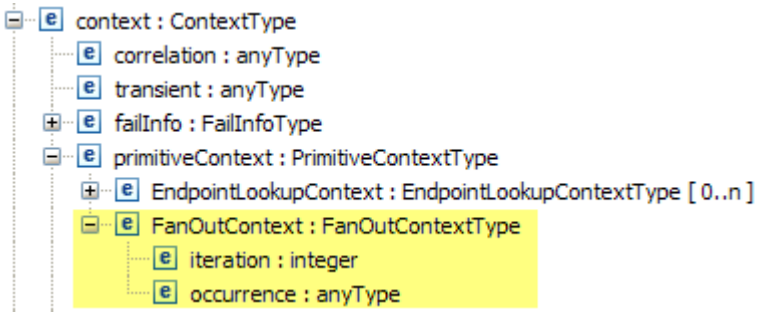
- *Reset message type*: removes any existing type augmentations
- ***Validate***: performs runtime validation to ensure field is of specified type

Fan out primitive

- Enables message to be processed by multiple paths or iterate over a repeated element
- No parallelism – synchronous processing of paths/elements

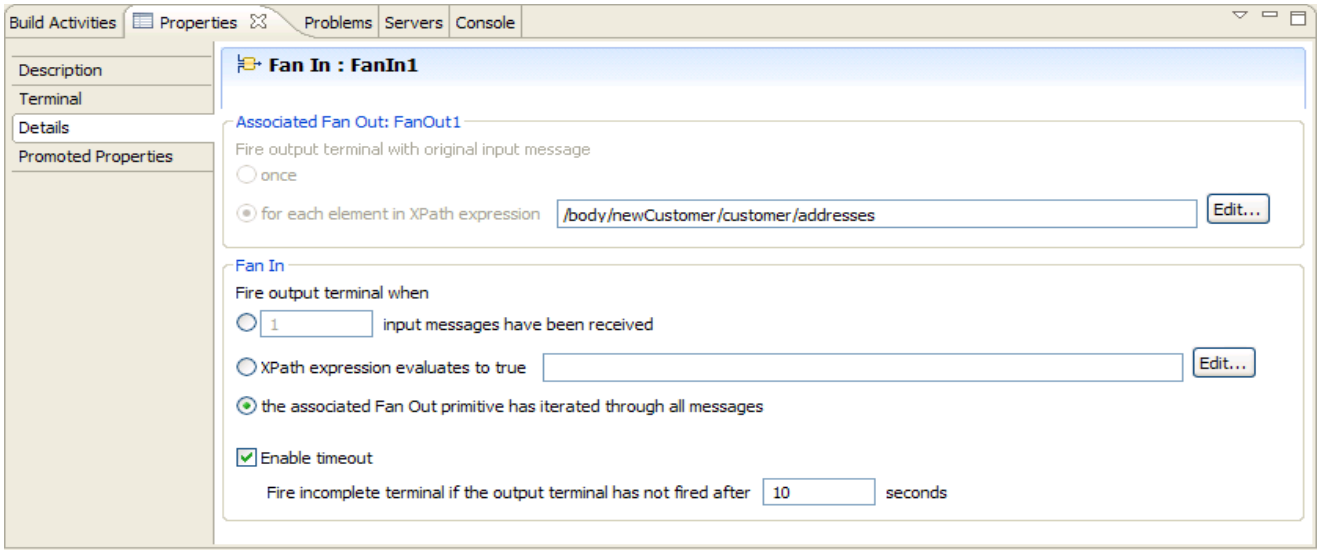


- *Fire output terminal with original input message*
  - *once*
  - *for each element in XPath expression*
    - FanOutContext contains *iteration* and *occurrence* (augmented type)
    - *noOccurrences* terminal fired if no elements found



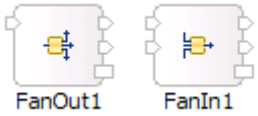
Fan in primitive

- Waits for messages from fan out until a completion criteria is met

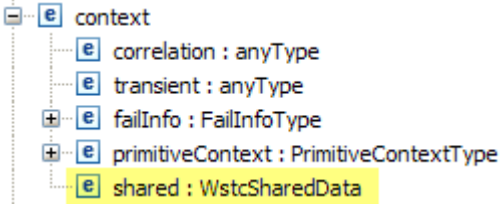


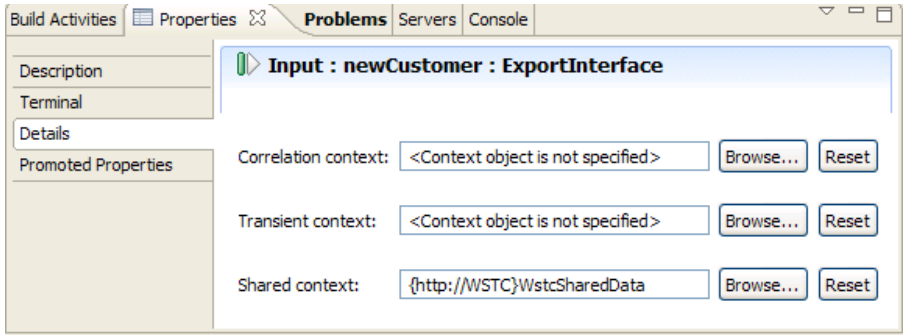
- *Fire output terminal when*
  - ***X** input messages have been received*
  - ***XPath** expression evaluates to true*
  - *the associated Fan Out primitive has iterated through all messages*
- *Enable timeout*
  - *Fire incomplete terminal if the output terminal has not fired after **Y** seconds*
- *stop* input terminal causes fan out/in to stop and fires *incomplete* terminal

Shared context

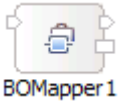


- Fan in does not in itself do any aggregation – the *out* terminal is just fired with the last message received
- Each path from the fan out receives a deep copy of the message e*xcept* for the shared context
- Shared context type defined on input node
- The path between the fan out and fan in should update this shared context
- The shared context is then propagated out from the fan in so can then be manipulated by subsequent primitives

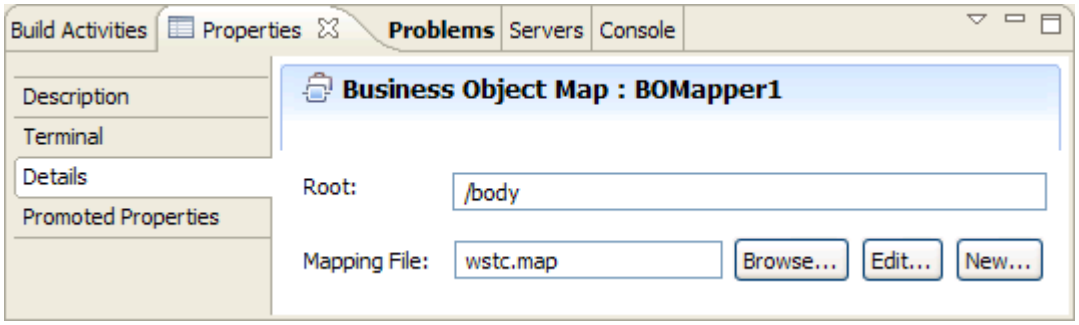




Business object map primitive



- Enables use of BO map for transformation of SMO

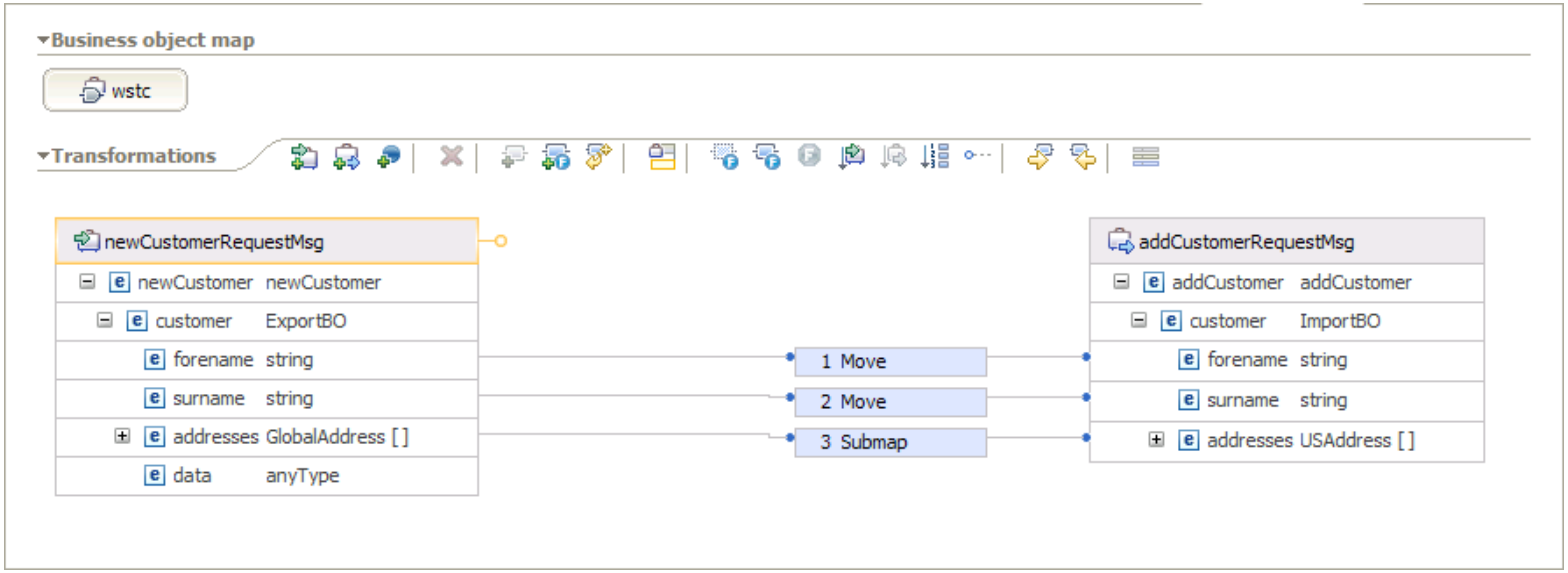


- *Root*: part of SMO to be transformed
  - /
  - /header
  - /context
  - /body

- Properties aren't editable – specified on creation of business object map

Business object map editor

- Transformations – processed in designated order
  - Move, extract, join, assign, submap, relationship, relationship lookup, custom, custom callout, custom assign
- Variables available to store temporary values during mapping
- Option to map similar fields where names don't match
- Cardinality enables position in array to be specified
- Graph or table view

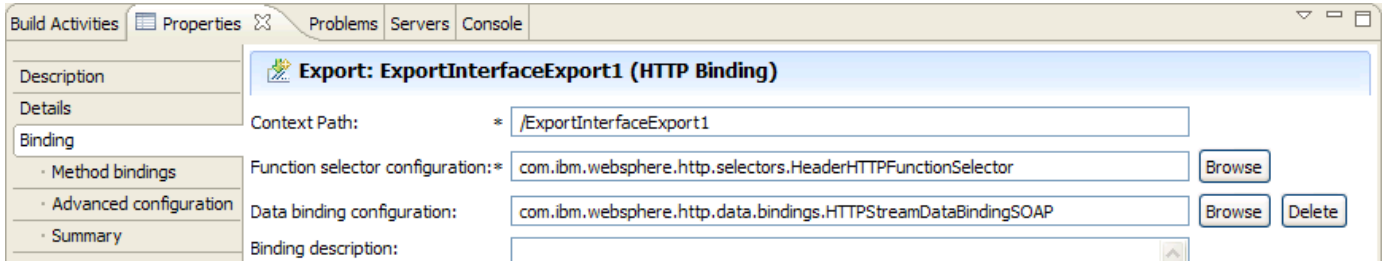


Business object map versus XSLT

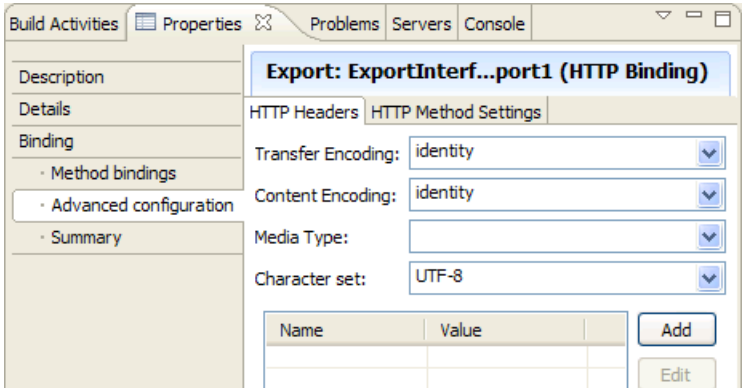
- Why use business object maps instead of XSLT?
  - Mapping requires maintaining a relationship
  - Change summary needs to be maintained in a business graph
  - Configure event settings to raise CEI events
  - Utilize existing investment in business object maps
  - Variables
  - Fuzzy mapping
- Performance
  - Business object map operates on service data object loosing benefits of lazy parsing of raw data
  - Actual transformation may be faster with a business object map

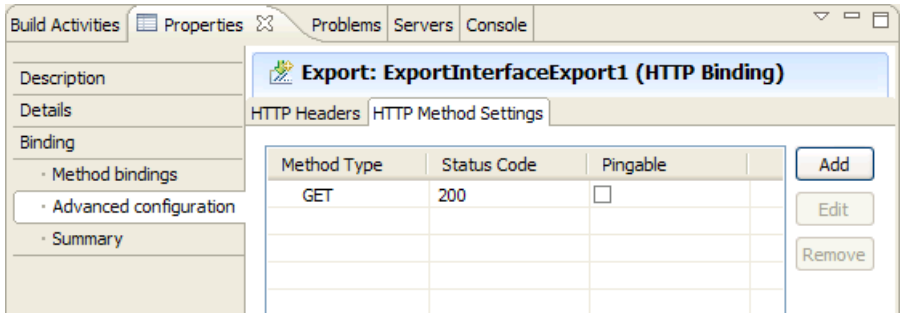
HTTP export binding

- Enables client connectivity via HTTP(S)
- Export context path, function selector and data binding specified on creation



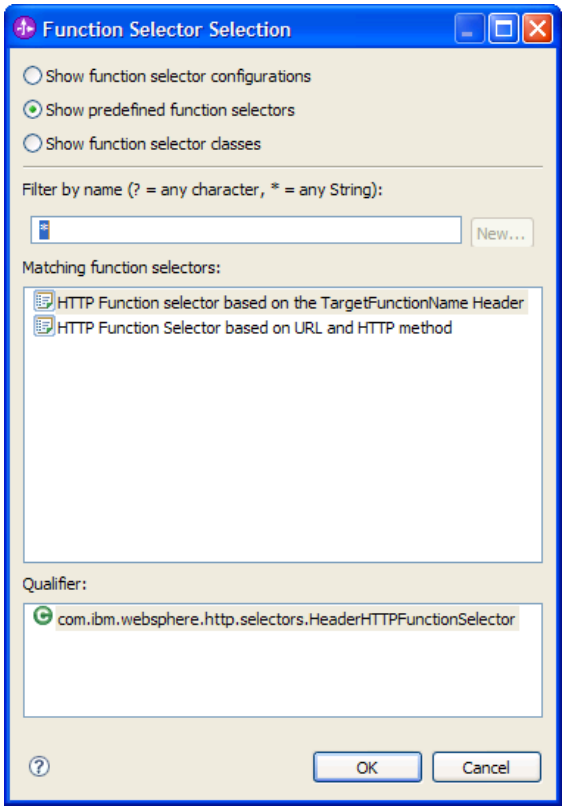
- Advanced configuration
  - HTTP headers for response
  - Supported HTTP methods and whether “pingable” (just status code returned)
- Methods bindings contain context path, native method(s) and enable other settings to be overridden on per operation basis





HTTP function selectors

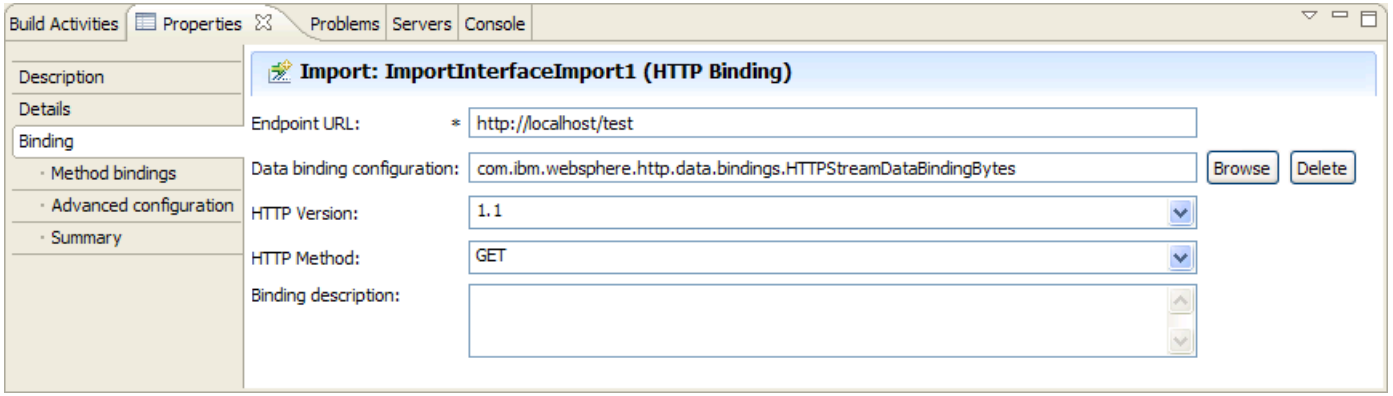
- Two predefined function selectors
  - *TargetFunctionName* header
  - URL and HTTP method
    - Native method name =
      HTTP Export Context Path
      + Operation Context Path
      + "@"
      + HTTP method
    - For example:
      /ExInterfaceExport1/newCustomer@GET



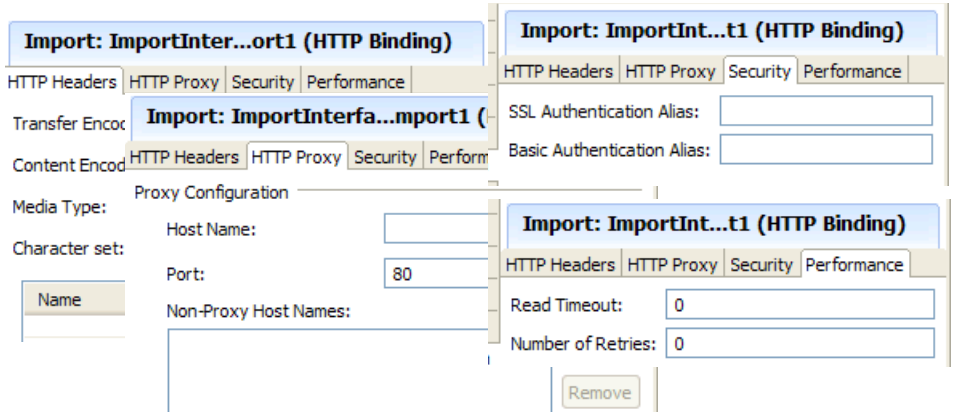
- Custom function selector
  - Extend *com.ibm.websphere.http.selectors.HTTPFunctionSelector*
  - Implement *String generateEISFunctionName(HTTPControl, HTTPHeaders, HTTPInputStream)*

HTTP import binding

- Enables invocation of service via HTTP(S)
- Endpoint URL and data binding specified on creation - option to generate TargetFunctionName

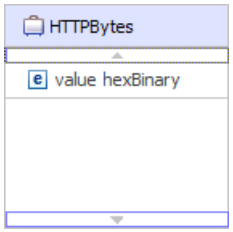


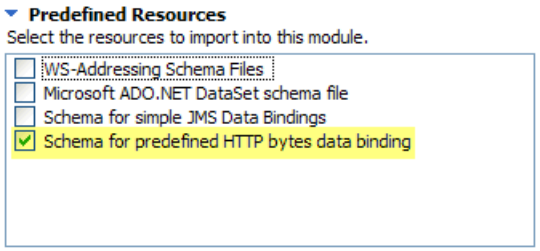
- HTTP version and method specified
- Advanced configuration
  - HTTP headers, proxy and SSL settings, and retry/timeout properties
- Method bindings allow all of above to be overriden on per operation basis

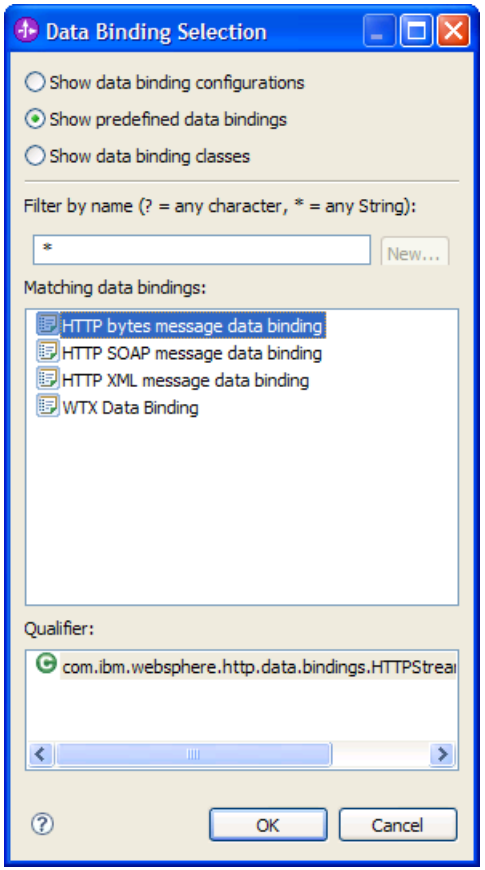


HTTP data bindings

- Supplied HTTP data bindings
  - HTTP bytes message
    - Request represented as XSD hexbinary
    - Add dependency for HTTPBytes business object
    - Operations must take/return HTTPBytes BO
  - HTTP SOAP message
  - HTTP XML message



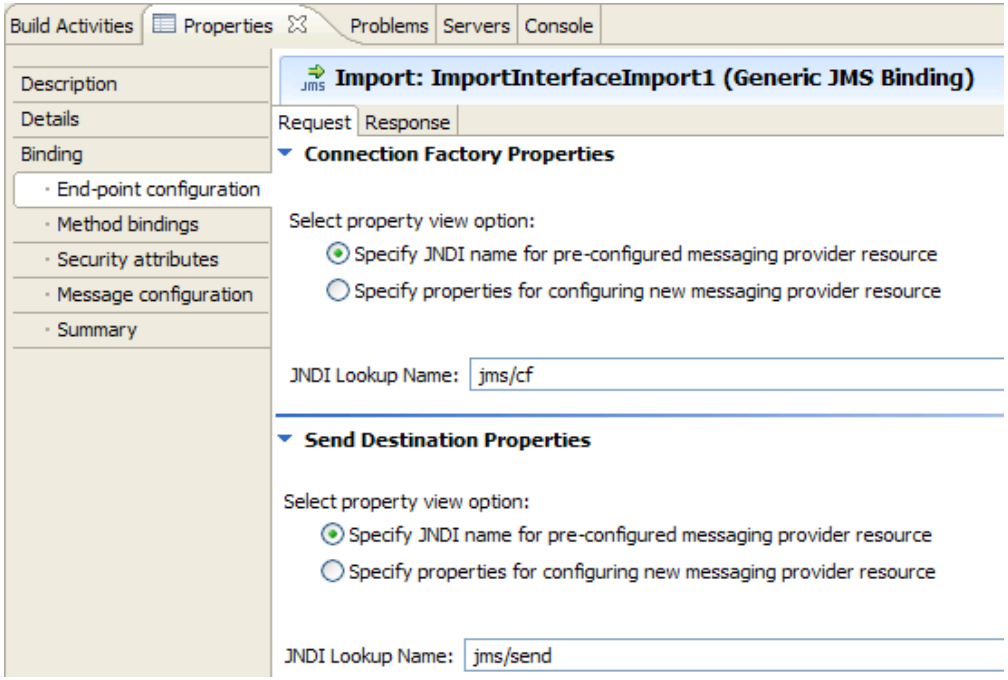




- Custom data binding
  - Implement com.ibm.websphere.http.data.bindings.HTTPStreamDataBinding
    - get/setDataObject
    - setBusinessException/isBusinessException
    - get/setControlParameters
    - get/setHeaders
    - convertFromNativeData(HTTPInputStream)
    - convertToNativeData()
    - write(HTTPOutputStream)

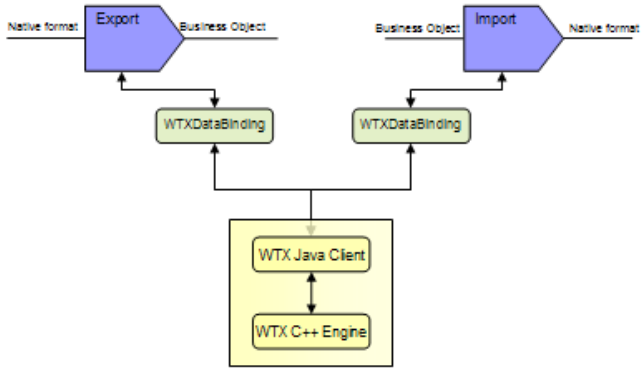
Generic JMS binding

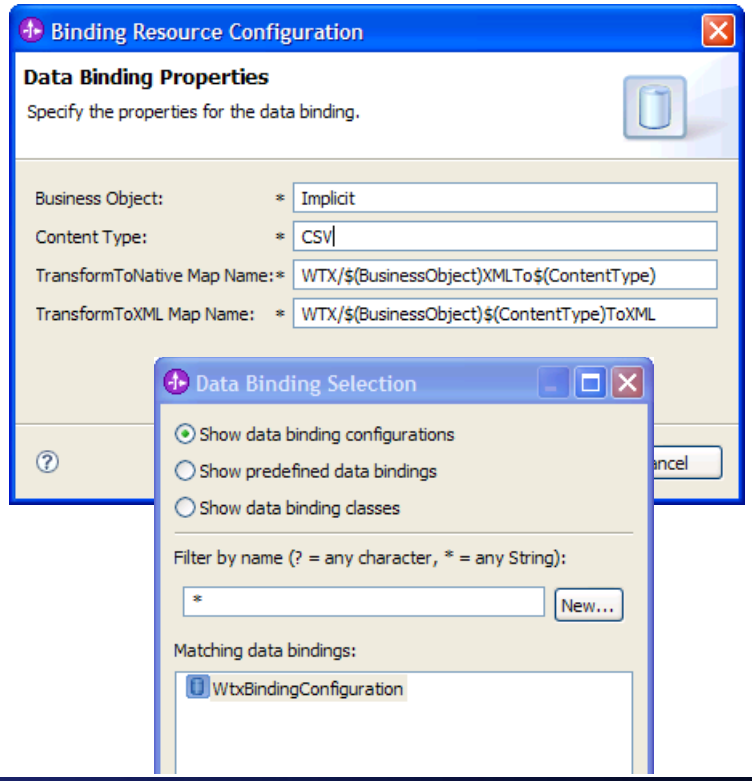
- Provides same capabilities for generic JMS providers as WebSphere MQ JMS and JMS bindings
- Provider be JMS 1.1 compliant and implement Application Server Facilities
- Resources can be preconfigured via application server's Generic JMS provider support or generated (name provider must already exist)



WTX data binding

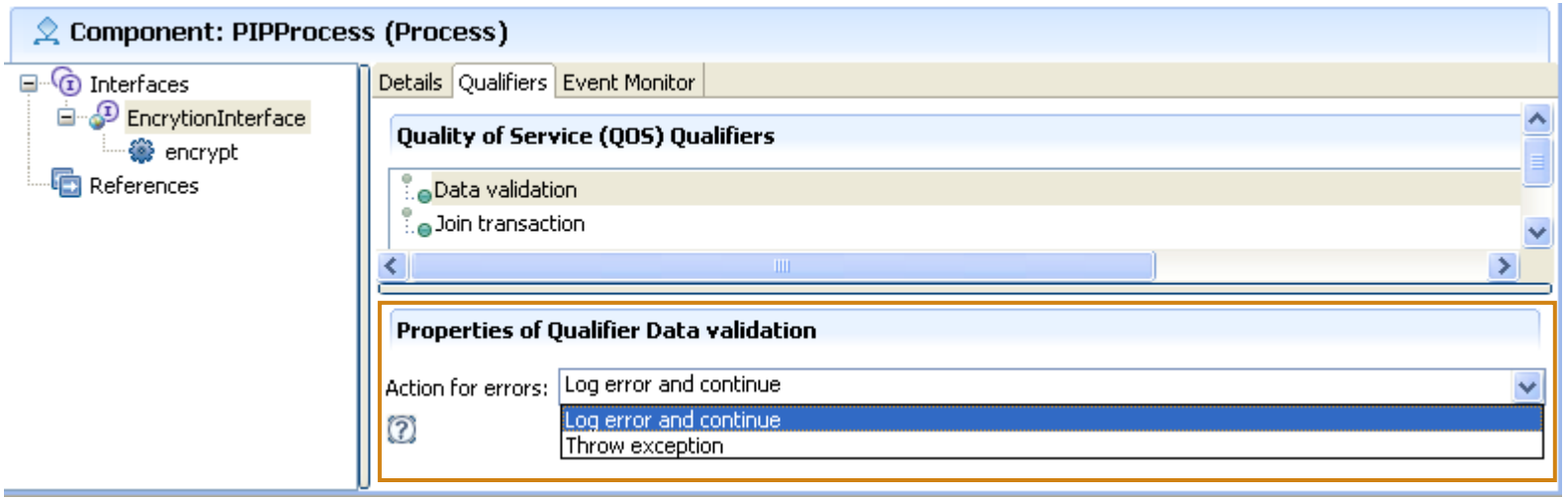
- WebSphere Transformation Extender provides universal data transformation and validation
- WTX data binding applicable to imports and exports with messaging, EIS and HTTP transport bindings
- Uses JNI to invoke WTX (8.2) transformation engine running in-process
- Data binding configuration defined via *New* > *Binding Resource Configuration* and then selected on creation of import/export binding
- Default is to find WTX map file in SCA module based on business object name and configured content type e.g. WTX/CustomerCSVToXML.mmc





Runtime Business Object Validation

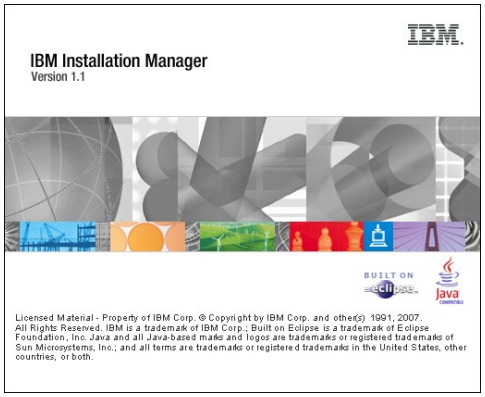
- Execution-time validation of request and response data against schema
  - Applies to all operations when enabled
  - Specified as an SCA Qualifier on an interface
- Reporting options
  - Validation errors are logged and running application continues
  - Exceptions thrown, potentially application dies

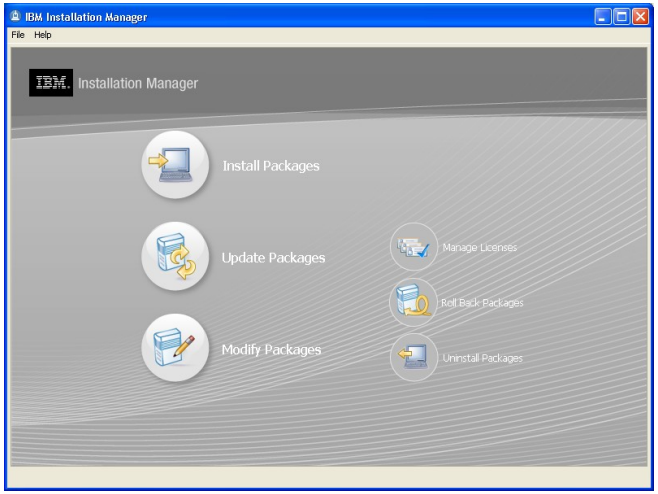


Enhancements to developer tooling

Installation Manager

- **Install the whole product stack simultaneously** when using multiple IM-based products (e.g., RAD, WID, MB toolkit, Monitor toolkit)
- Install product updates during product installation
- Go from product install to latest level and skip the updates in between (upgrade from 1.0->1.5 without installing the interim levels)
- **Combine products into different configurations** and still share plugins if the same versions are installed (e.g., one configuration with WID/RAD, another with just WID)
- **Service both IDE and UTE through the same mechanism**
- Uninstall older versions of plugins when updates are applied (avoid disk space bloat over time)



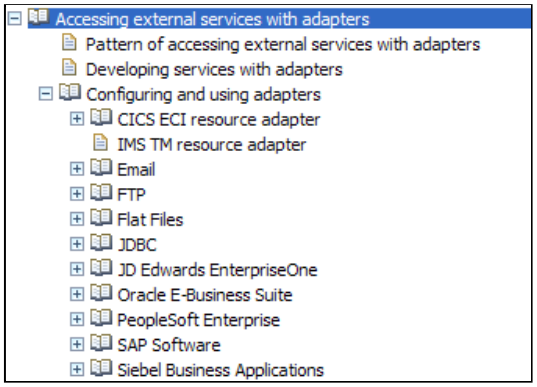


Major Categories of Improvements

- Full (tools and runtime) support for additional XSD/WSDL constructs
  - xsd:choice, xsd:any family, xsd:union
- Support for popular industrial schemas such as HL7 (Health Care), ACORD (Insurance), OAGIS (B2B), and others
  - Some documented fixes requires, schemas have bugs too!
- Numerous fixes
  - i.e. Problem: xsd:attribute cannot have the same name as xsd:element
- Validation
  - Consistent WSDL and XSD validation in WID and WPS
  - Faster and better WSDL validator from RAD
  - New runtime message validation
  - Validation for: xs:ID, xs:IDRef, xs:NMTOKEN, xs:Key, xs:KeyRef

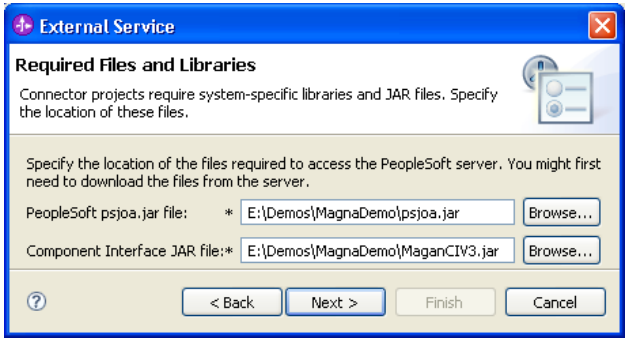
J2C Adapters – Key New Features

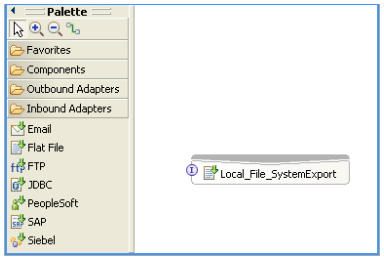
- Documentation included in Info Center



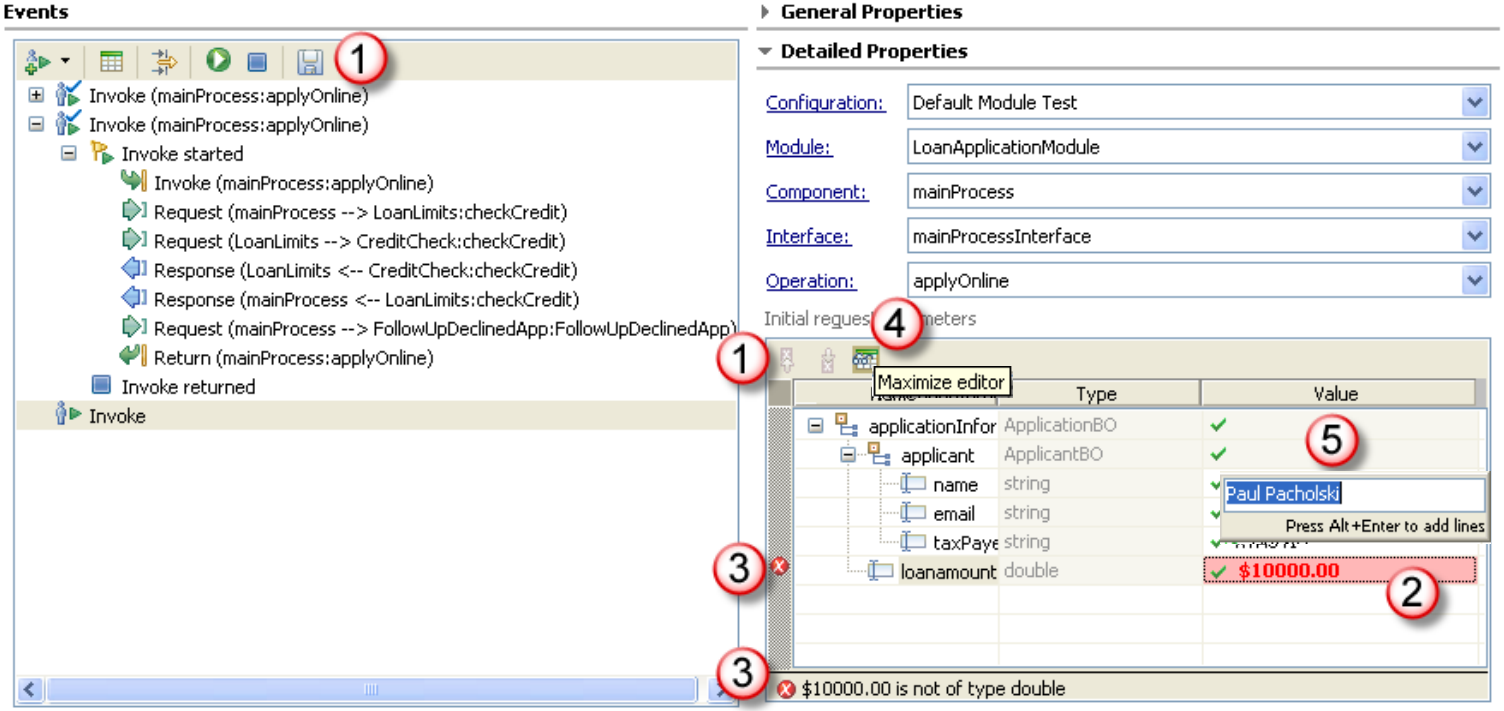
- Redesigned Adapter Wizards – greatly improved usability
- Drag and drop adapter from the Assembly Editor palate
  - Adapter RAR file is installed automatically!
  - Adapter Wizard is launched

- Adapters Wizards asks for the adapter specific jars adds them to your adapter projects (i.e. People Soft jars)





Integrated Test Client Enhancements

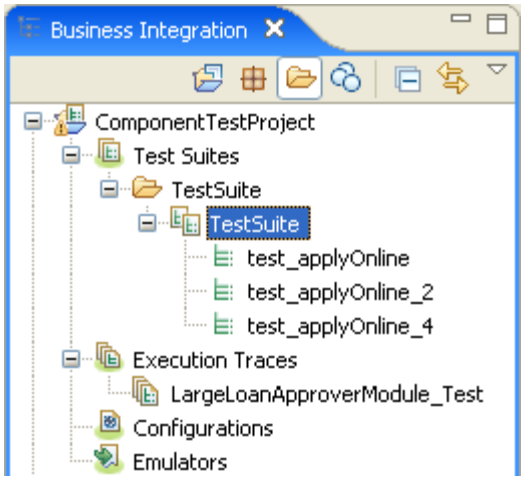


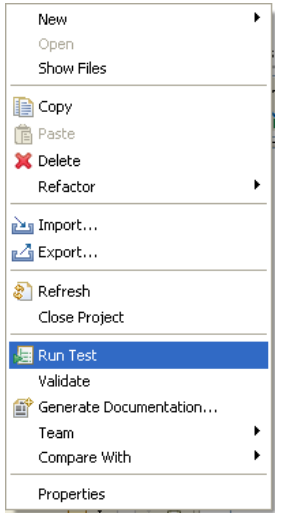
1. Buttons repositioned to where they are used
2. New input UI
   - Icons denoting structure types and input state
   - Parameter validation

1. Status area and error columns simplify error identification
2. Maximize Editor button for easier data entry
3. Multi-line data entry

Support for Application Testing

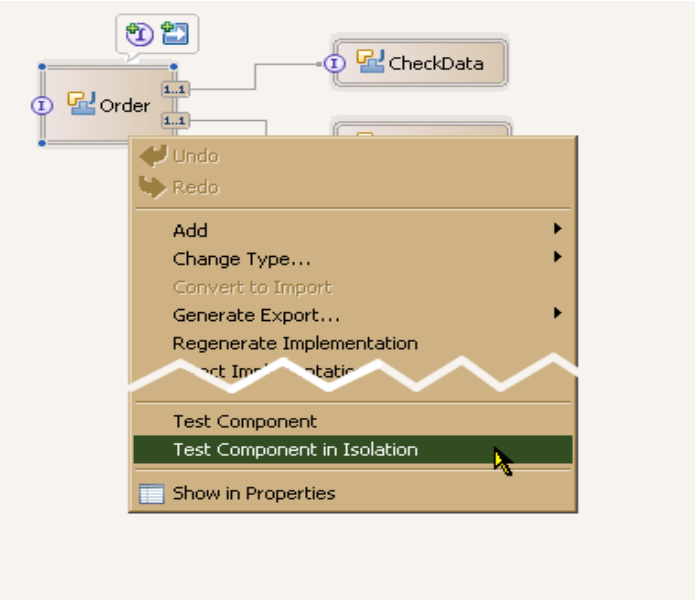
- Users can automate testing of WID applications
- New Component Test Projects
  - Test Projects contain Test Suites
  - Test Suites contain Test Cases
  - Test Cases can be authored:
    - Manually
    - From Test Client invocation trace
- Test execution is triggered from a Component Test Project
  - Test are deployed to server and executed
  - Results appear within the Integrated Test Client
  - Can be invoked from an ant script
- Test Variations
- Test Scenarios





Test component in isolation

- 6.0.x: Test Component meant test selected component, emulate everything else
  - Not what user expects
- 6.1: Test Component means test module, but start with selected component
  - What users expect
- 6.1: Test Component in Isolation: test selected component, emulate everything else
  - Test Component behavior in 6.0.x was useful in some cases, so now it is named for what it does



Choice support

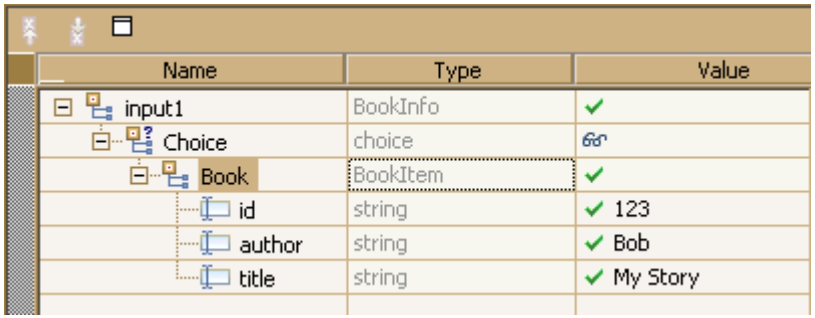
```
<xsd:complexType name="BookInfo">
    <xsd:choice>
        <xsd:element name="ISBN" type="xsd:string" />
        <xsd:element name="Book" type="bons0:BookItem" />
    </xsd:choice>
</xsd:complexType>
```

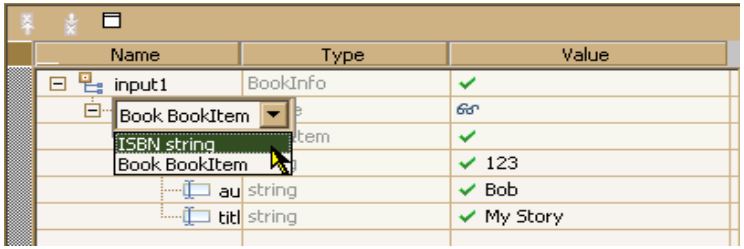
- 6.0.x: <xs:choice> was treated as an <xs:sequence>
  - Required user to unset all but one element
  - User needed to be familiar with the schema

Name	Type	Value
input1	BookInfo	
ISBN	string	<unset>
Book	BookItem	
id	string	123
author	string	Bob
title	string	My Story

Choice support

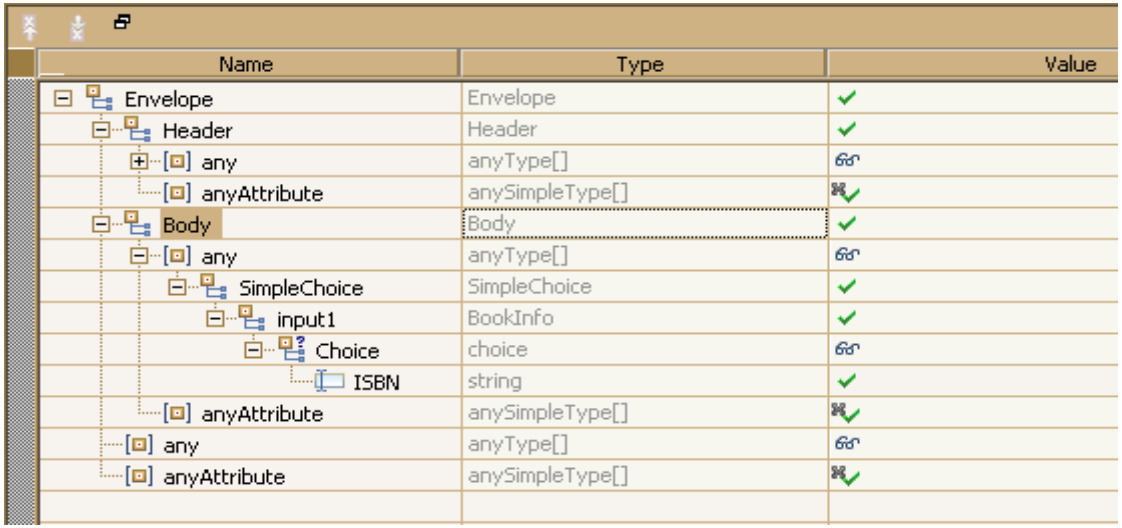
- 6.1: Select a choice element
  - Unselected elements are unset
  - Click the Choice cell to change selected element
- xs:union treated similarly





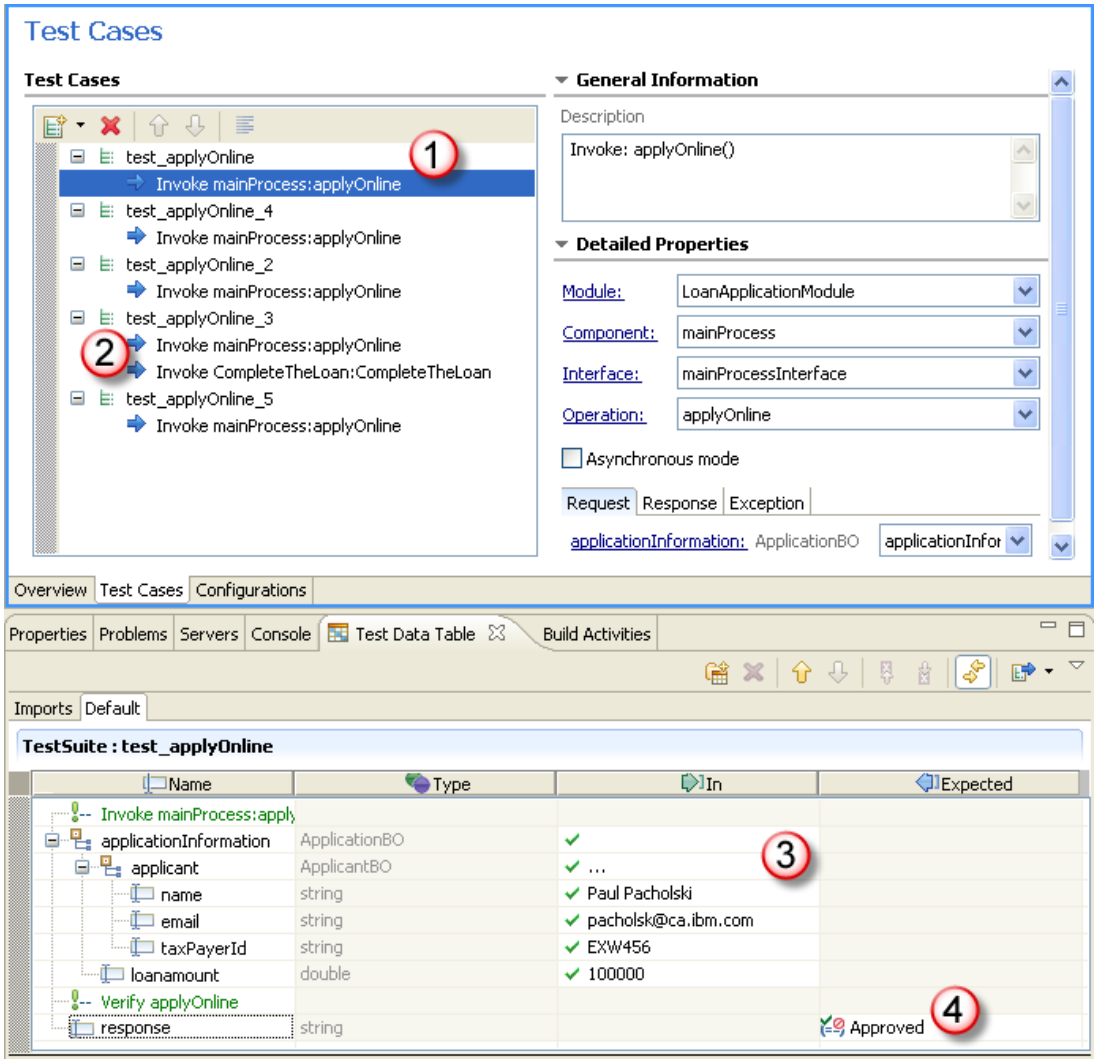
Testing thru the edges

- 6.0.x: Testing export == testing component wired to export
- 6.1: Test client looks at export binding
  - SCA: Equivalent to calling from an import in another module
  - Web Service: Client sends soap message to the address in the WSDL binding

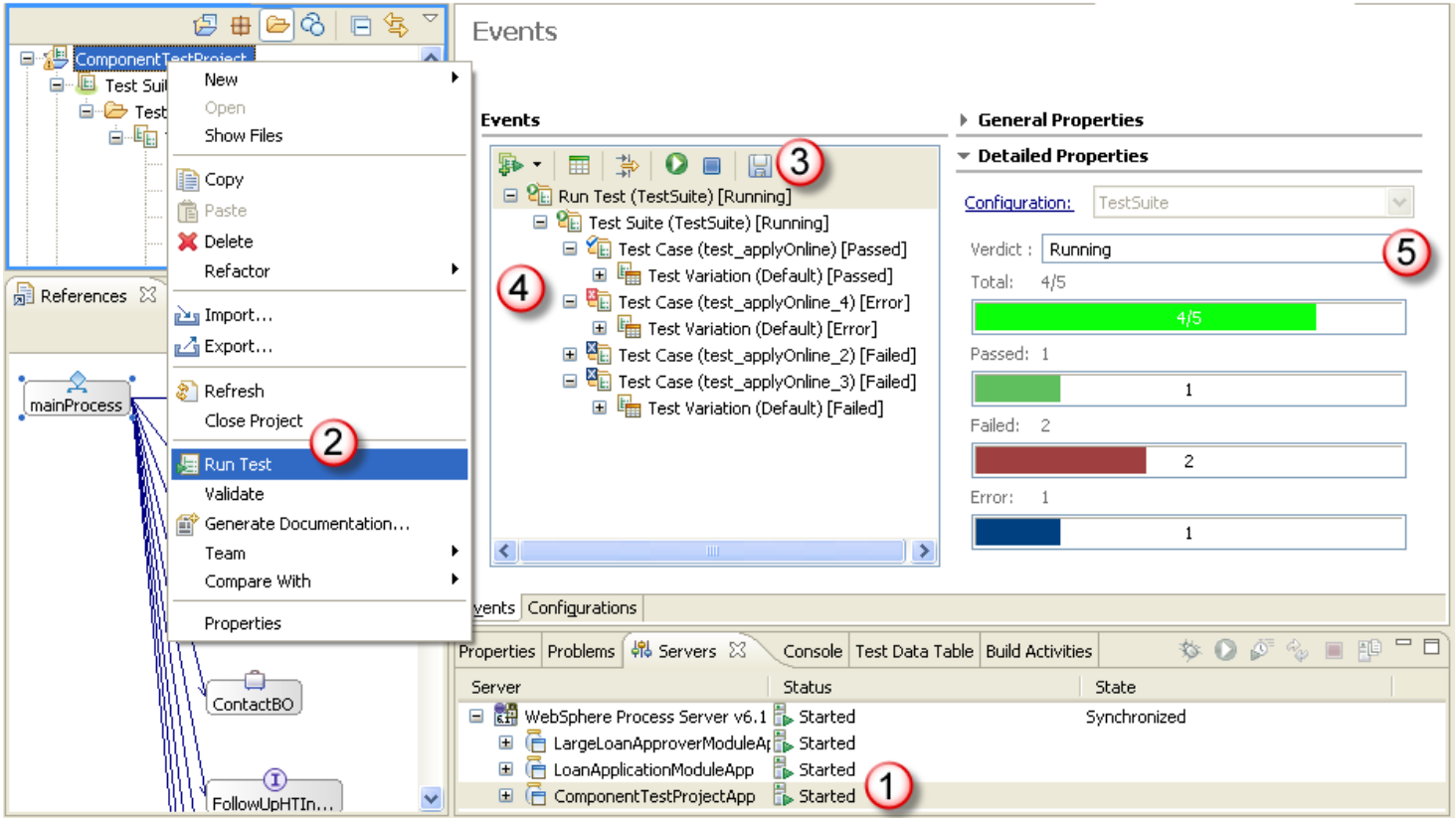


Authoring Test Cases

1. Collection of tests in a test case
2. A test case may include a series of operations
   - i.e. An interaction with state machine
3. Each test has separate input data
4. Each test has expected return value
   - Used to determine if the test failed or succeeded
   - Has expression builder



Executing Test Cases



1. Component Test Project is automatically deployed to the server together with the module under test
2. Launching the Test opens Test Client

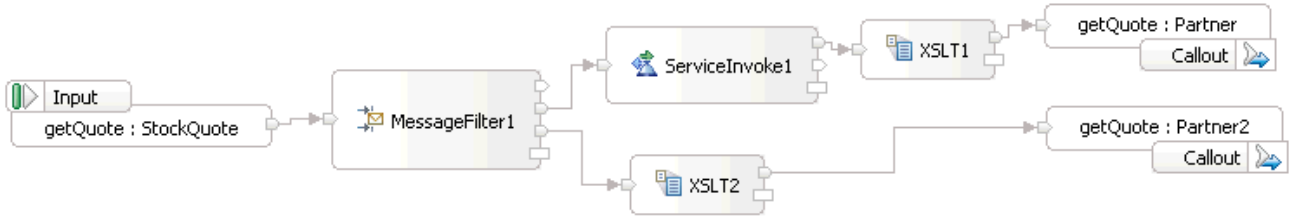
1. Toolbar to launch the test
2. Detailed invocation trace is optionally available
3. Test execution totals are shown for a Test Suite

Summary

- Product Overview – a review
- Enhancements to existing features
- New runtime features
- Enhancements to developer tooling

Backup

Service Invoke – Parallel Threads Example



At run time the flow on thread one might be as follows:

1. The first matching output terminal of MessageFilter1 is fired.
2. The input terminal of ServiceInvoke1 receives the input message, and ServiceInvoke1 performs an `asyncWithCallback` invocation. The call does not wait for a response and returns immediately.
3. The thread traces back up the wiring path to the MessageFilter1 mediation primitive.
4. The second matching output terminal of MessageFilter1 is fired.
5. The input terminal of XSLT2 is reached, and the transformation occurs.
6. The response to the invocation made by ServiceInvocation1 is received. The flow engine handles this and resumes the processing in ServiceInvoke1 on a new thread.

From now on, the threads can process in parallel.

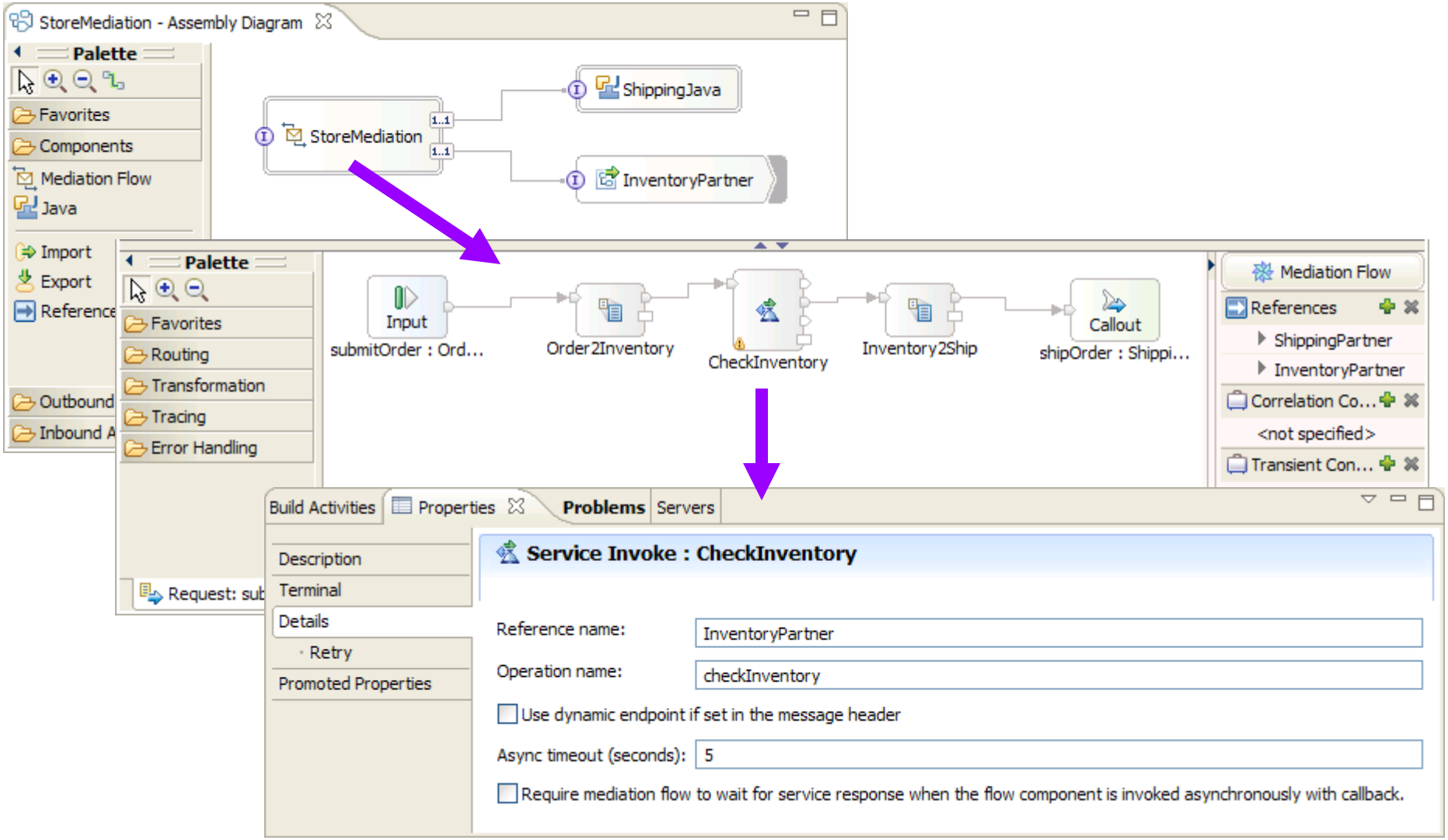
The following actions occur on thread one:

1. The output terminal of XSLT2 is fired.
2. The Partner2 callout is invoked. This results in the invocation of the reference, and whatever the reference is wired to. For example, the sending of a SOAP message from a Web services import.
3. Because this is the end of the wiring path, the thread tracks back up the wire to the MessageFilter1 primitive.
4. There are no more output terminals to fire. Therefore, MessageFilter1 has completed and the thread tracks the wire back out of the input terminal to the previous mediation primitive, which in this case is the Input node.
5. At this point, thread one has completed.

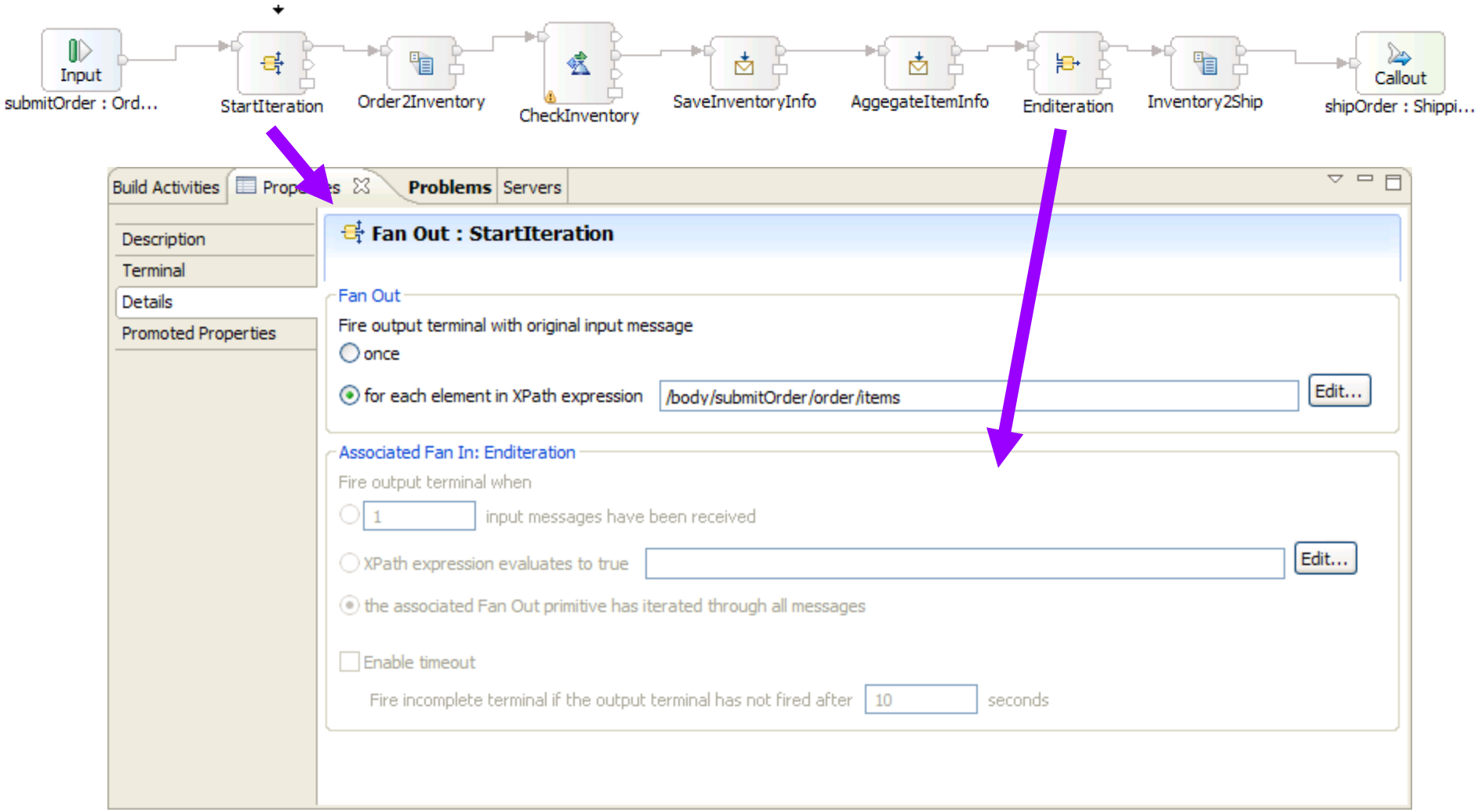
The following actions occur on thread two:

1. The input terminal of XSLT1 is reached, and the transformation occurs.
2. The output terminal of XSLT1 is fired.
3. The Partner callout is invoked. This results in the invocation of the reference, and whatever the reference is wired to. For example, the sending of a SOAP message from a Web services import.
4. Because this is the end of the wiring path, the thread tracks back up the wire to the ServiceInvoke1 primitive.
5. At this point, thread two has completed.

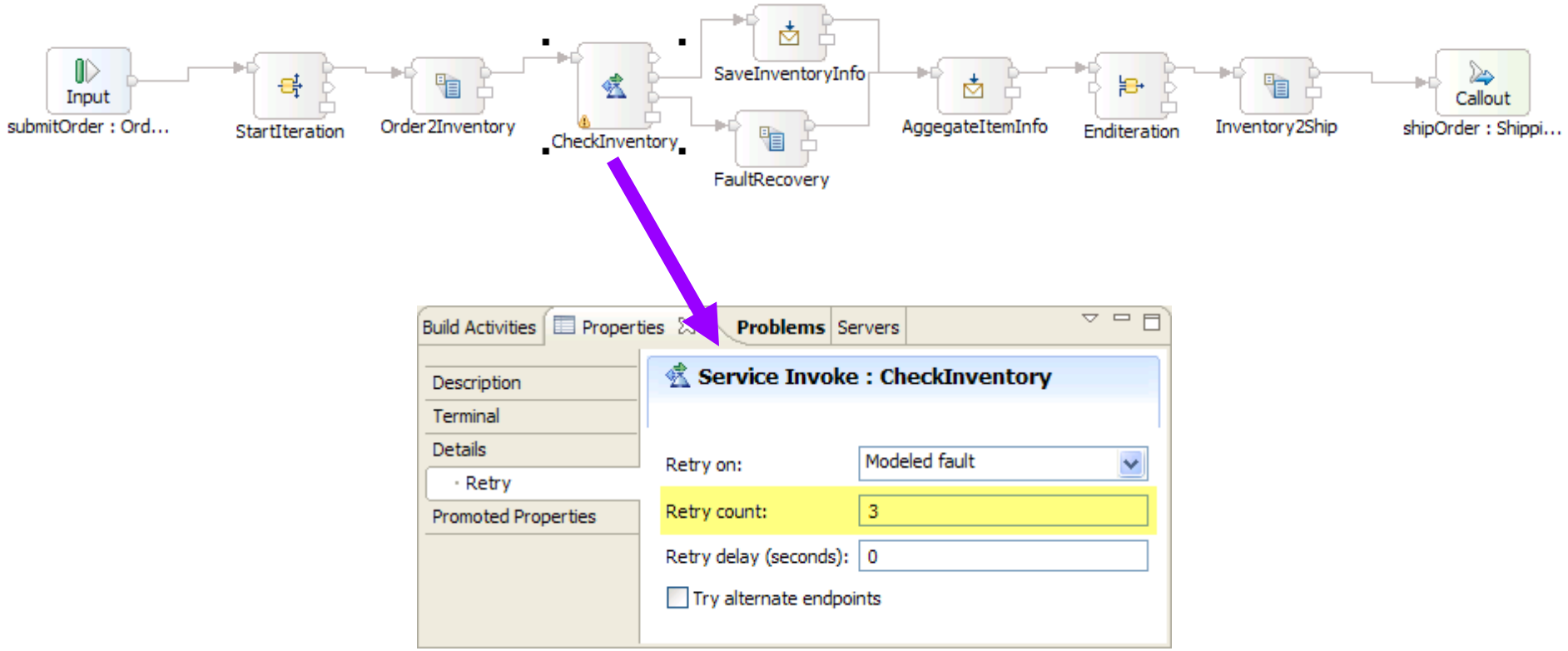
Service invoke for message augmentation



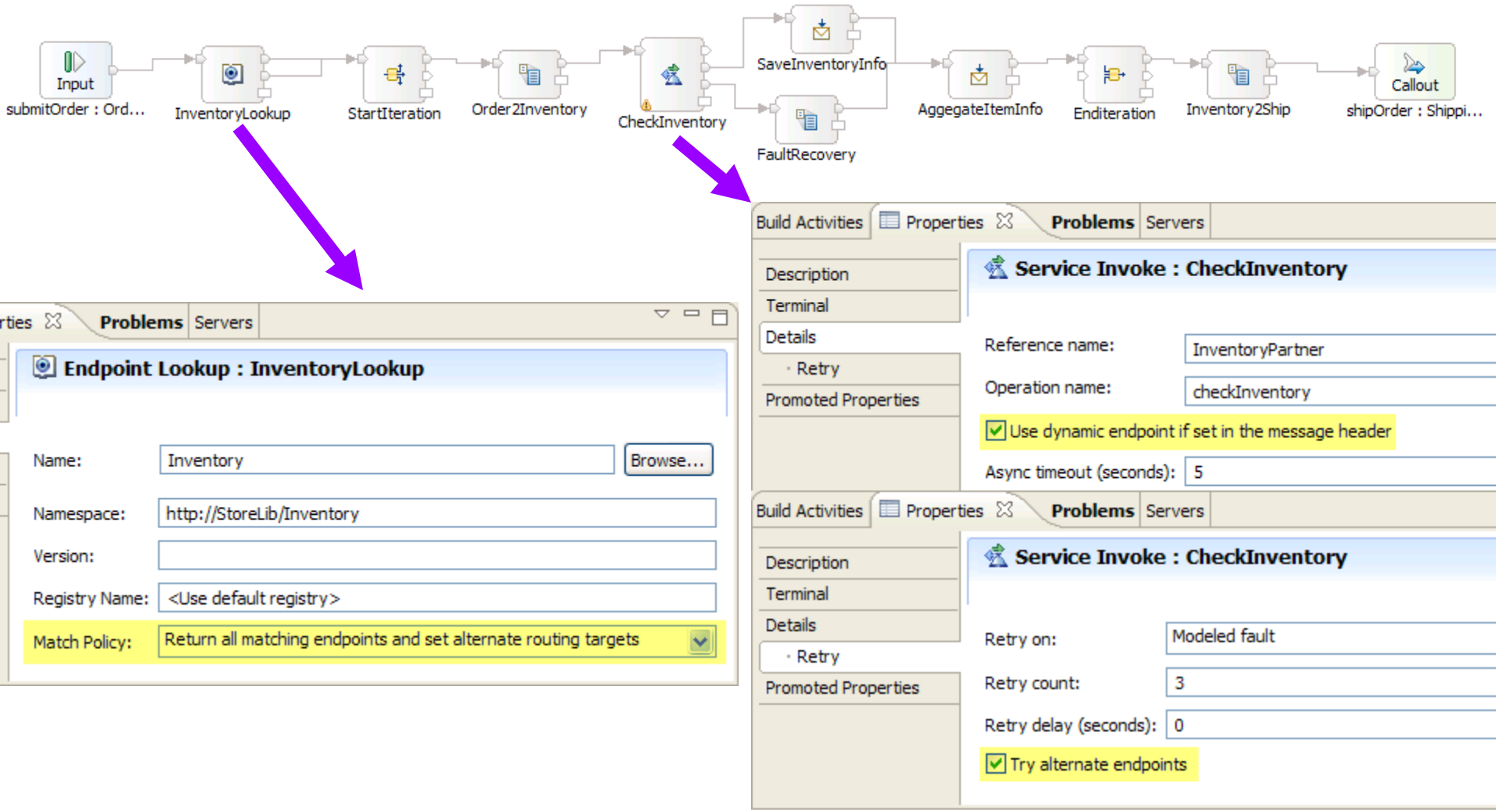
Fan-out/fan-in for iteration



Retry for failure handling



Alternate endpoints for failure handling



Custom mediation for fault handling