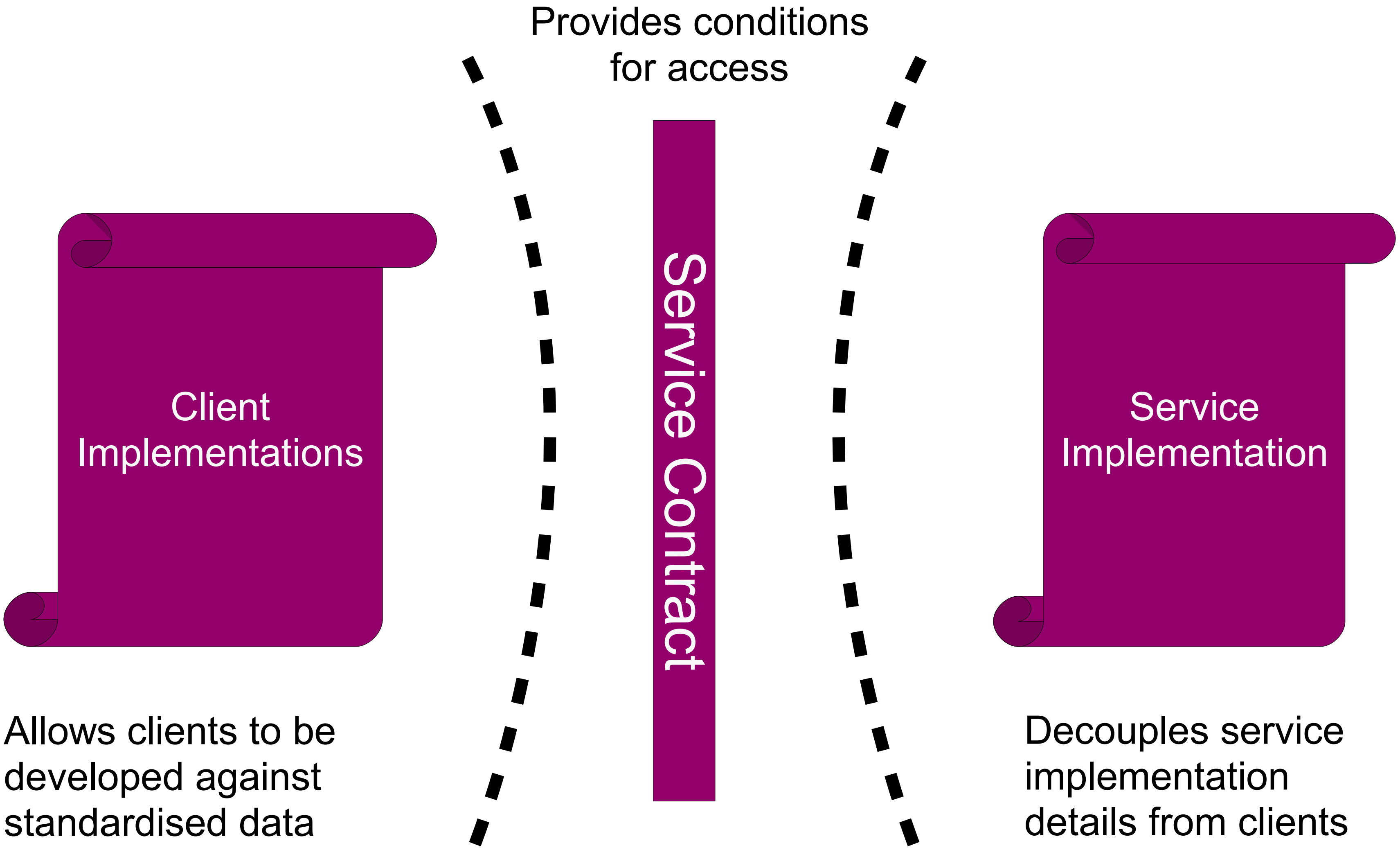
What is in a Good Contract?

Designing Interfaces for Distributed Systems

Schalk W. Cronjé
ysb33r @ gmail.com
@ysb33r

Service Contract



Provides conditions for access

Client Implementations

Service Contract

Service Implementation

Allows clients to be developed against standardised data models and message structures

Decouples service implementation details from clients

Contents

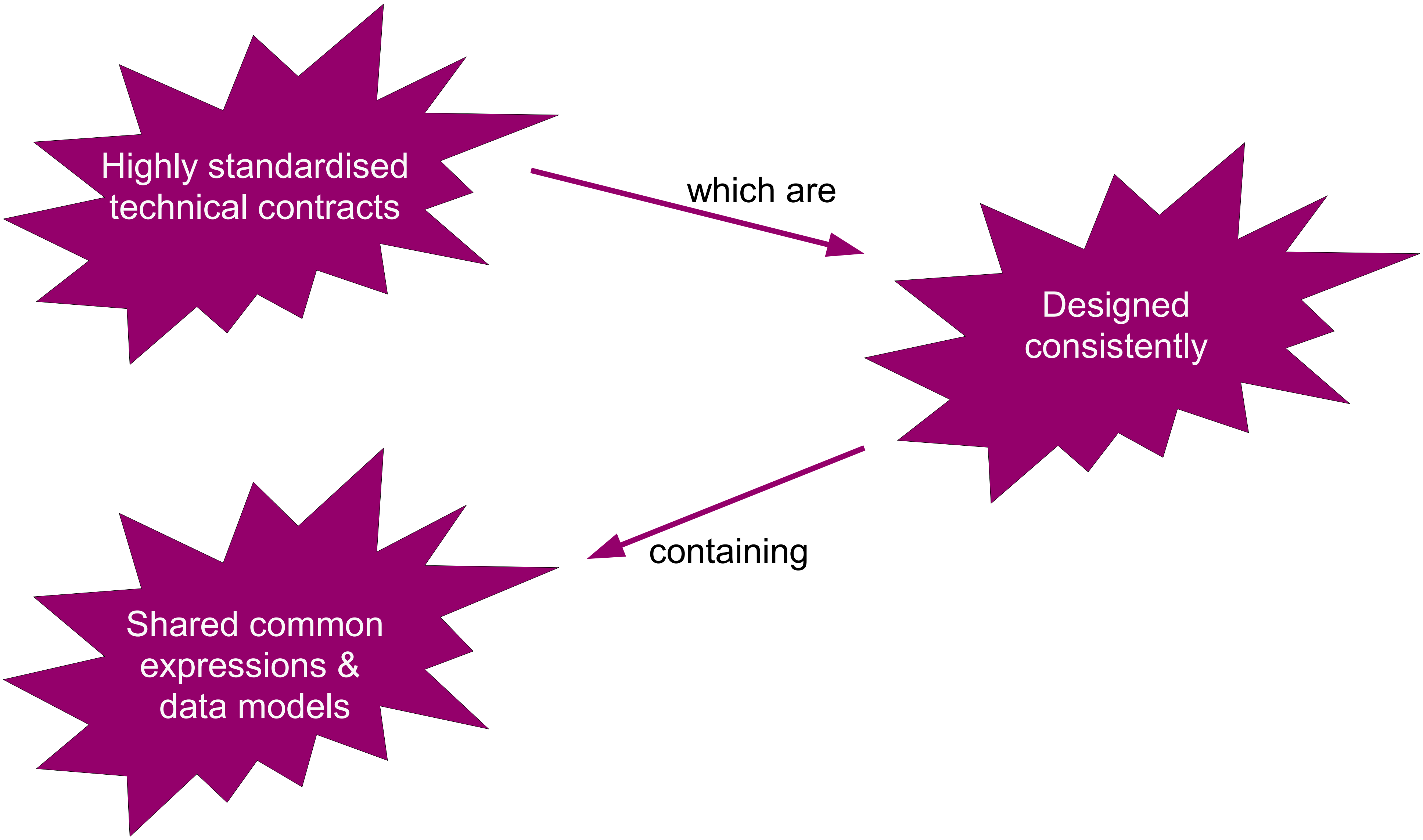
- Goals for service contract design
- XML services
- Evaluating example contracts
- Security policies
- TDD & Service Contracts
- What makes a good contract?

Goals for Service Contract Design

© Greco @ skiboardsonline.com

ACCU 2011
© Schalk W. Cronjé

Intrinsic Interoperability



Highly standardised technical contracts

which are

Designed consistently

containing

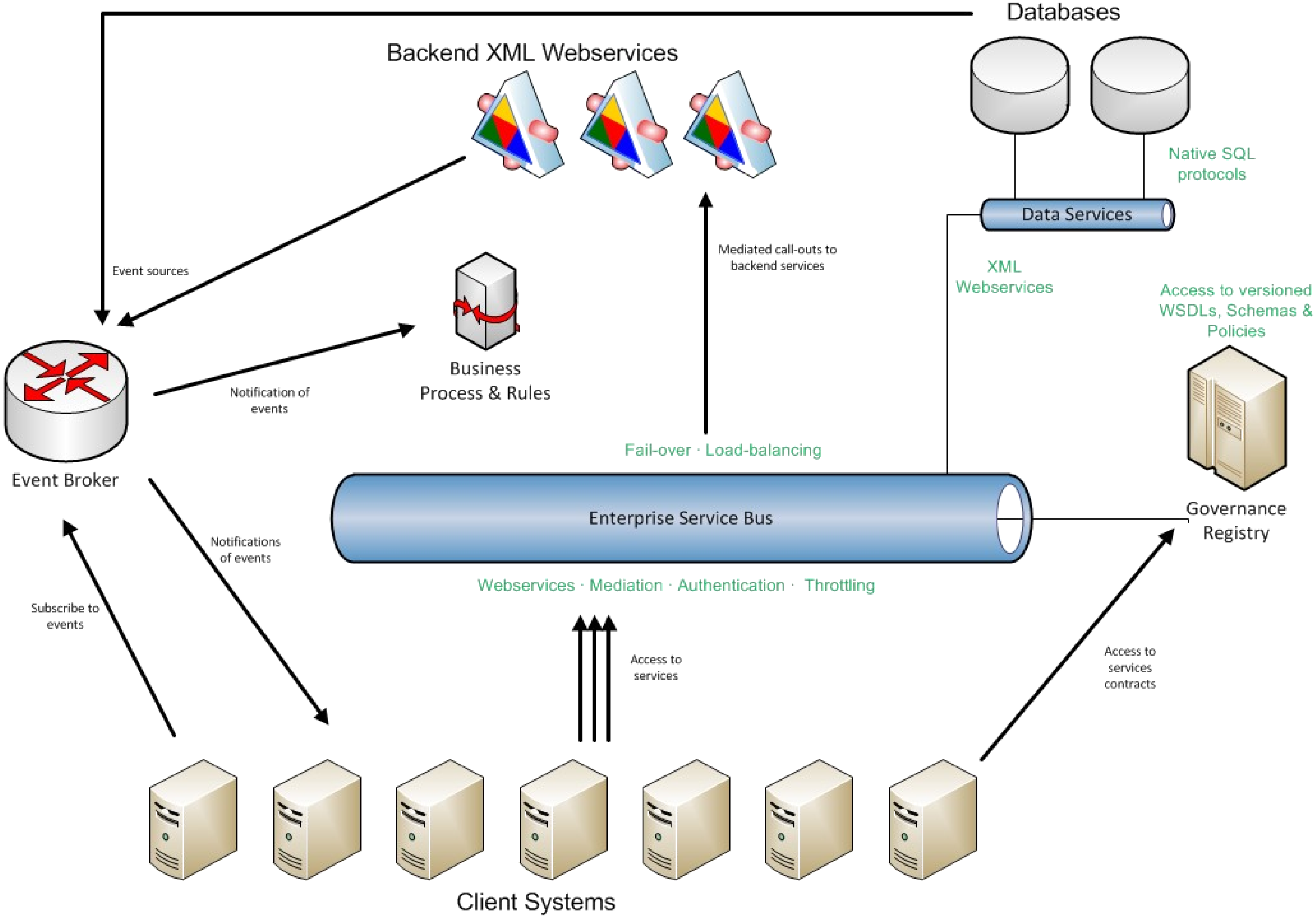
Shared common expressions & data models

Implementation Technology Agnostic

- Expression without vendor-specific details
- Avoid vendor lock-in within contract
- Avoid implementation leakage

Federation

Consistent endpoints on technical service portfolio boundary



ACCU 2011
© Schalk W. Cronjé

Business & Technology Alignment

- Business-centric services
- Express business logic in close alignment with business analysis
- Production of conceptual versions before physical design

Abstraction

- Turn service into black box
- Contract is the only official published source
- Exposure of only essential information

Reusability

- Ability to re-use service for service composition and routing
- Forces decision on granularity of service
- Do one thing and do it well

Statelessness

- Minimise resource consumption
- Defer management of state information to a better suited backend
- Allows for easier load-balancing

Composability

- Be effective participant in a composed service irrespective of the size and complexity of the composition
- Possibilities include:
  - Direct exposure of one operation within another service
  - Routing of messages from one service to another
  - Single front-end, selected back-ends depending on operation

Maintainability & Supportability

- “Easy” to read contracts
- Ability to fix bugs in service without affecting the contract
- Operational message debugging
  - Understanding the message flow
  - Reading the message on the wire might be the only way of identifying an issue

Goals for Service Contract Design

- Intrinsic Interoperability
- Business and Technology Alignment
- Implementation Technology Agnostic
- Federation
- Abstraction
- Reusability
- Composability
- Maintainability
- Supportability

XML Services

SOAP vs REST

- WSDL as contract medium more mature than WADL
- SOAP-based XML Services not restricted to HTTP transports unlike REST
- SOAP-based XML Services has many standards
- Rest of content will concentrate on SOAP-based contracts

WS-I Profiles

- Defines requirements for interoperability based upon collections of specific web standards
- Contracts must be designed to conform to a specific profile
- Basic Profile 1.1 / 1.2
- http://www.ws-i.org

XML Service Guidelines

- Use XML namespaces to
  - separate data models
  - version contracts
- Prefer SOAP document-literal contracts to rpc-literal
  - Allows data model design to be completely decoupled from service contract in design
- DO NOT use rpc-encoded contracts

Four cornerstones

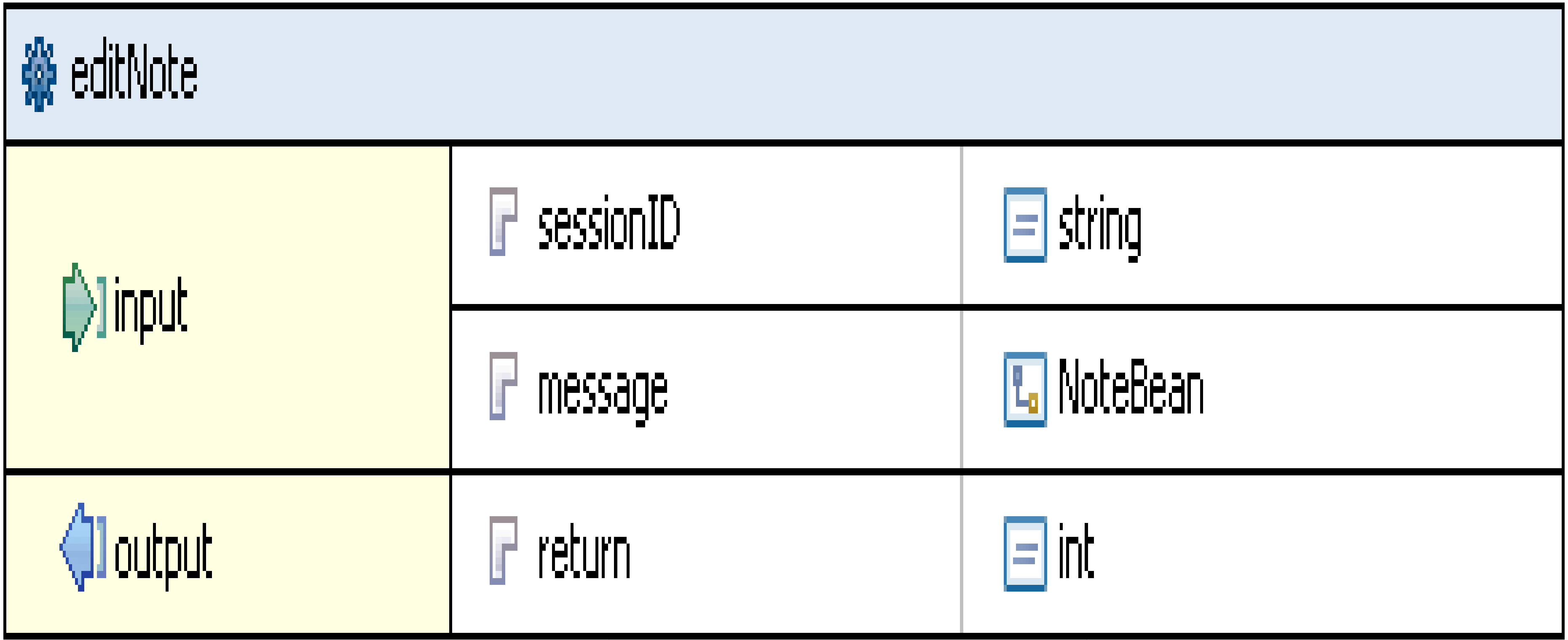
- Operations
  - What operations are supported?
- Data model
  - How is the data structured?
- Locations
  - Where can this service be found?
- Policies
  - What are the operational policies and contraints?

Non-functional contract aspects

- Non-functional aspects are attached to the message header
  - Embedding aspects in message body forces unnecessary coupling
- Security aspects added via WS-Security policies
- Message routing added via WS-Addressing
  - Required in SOAP 1.2 / Basic Profile 1.2
- Reliable messaging added via WS-ReliableMessaging

Exploring Contract Examples

Discussion: editNote operation



editNote		
input	sessionID	string
	message	NoteBean
output	return	int

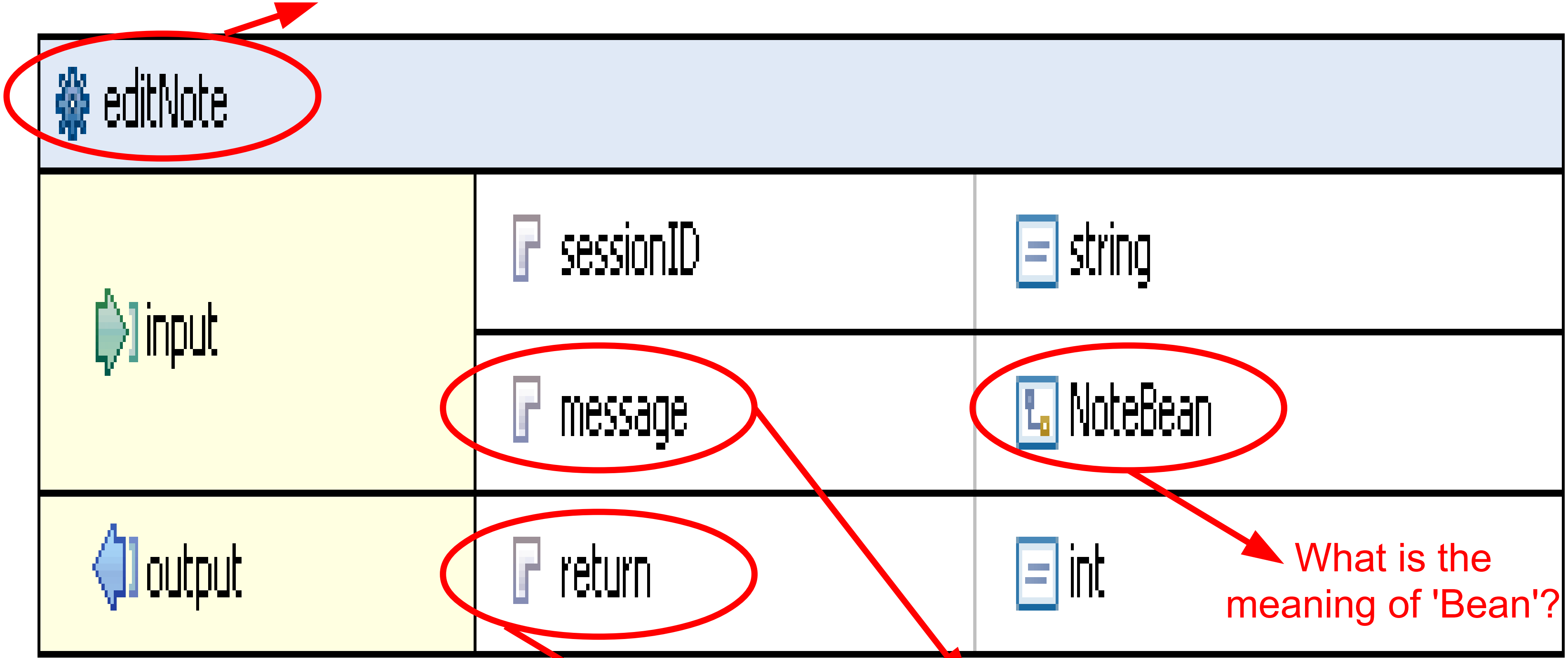
Part name

Data type

rpc-literal contract

Discussion: editNote operation

What is the semantical interpretation of 'edit'?



editNote		
input	sessionID	string
	message	NoteBean
output	return	int

What is the meaning of 'Bean'?

Why call a returned part 'return'?

Why call part of a message 'message'?

XML Message: editNote

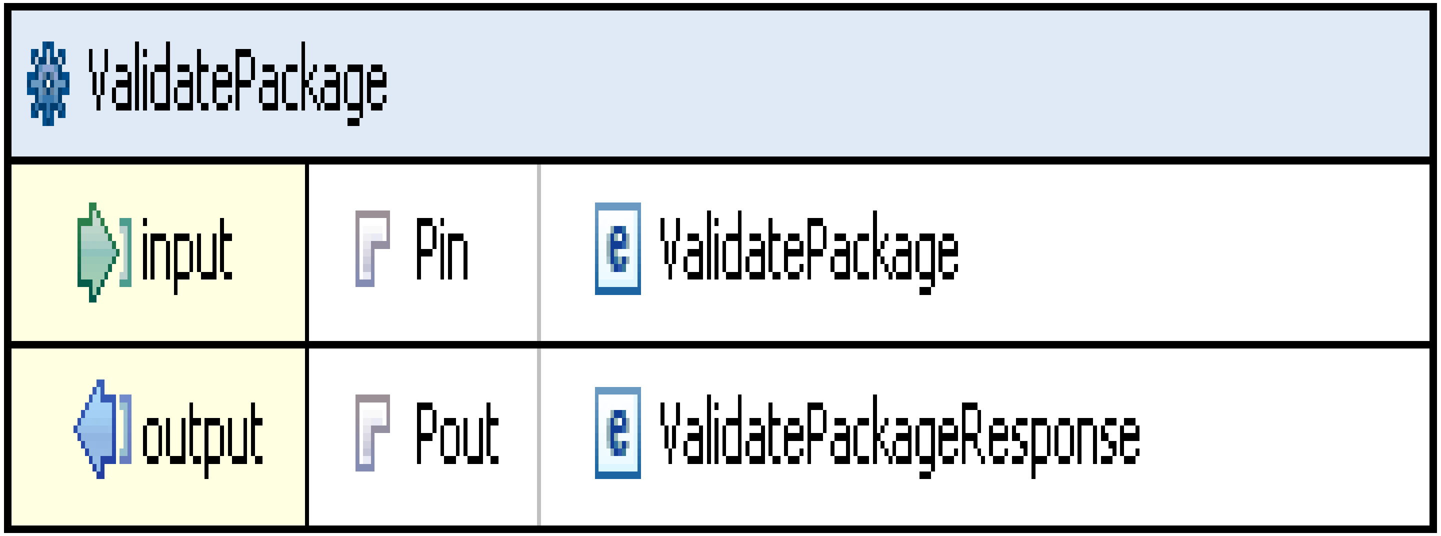
```
<soapenv:Envelope xmlns:soapenv="..."
  xmlns:api="http://example.com/api">
   <soapenv:Header/>
   <soapenv:Body>
      <api:editNote>
         <sessionID>?</sessionID>
         <message>
            <hasPassedSecurity>1</hasPassedSecurity>
            <ID>aNewUser</ID>
            <emailUsers>ysb33r@gmail.com</emailUsers>
            <entryDate>2010-11-01T00:03:05Z</entryDate>
            ...
```

XML Message: editNote

```
<soapenv:Envelope xmlns:soapenv="..."
  xmlns:api="http://example.com/api">
   <soapenv:Header/>
   <soapenv:Body>
      <api:editNote>
         <sessionID>?</sessionID>
         <message>
            <hasPassedSecurity>1</hasPassedSecurity>
            <ID>aNewUser</ID>
            <emailUsers>ysb33r@gmail.com</emailUsers>
            <entryDate>2010-11-01T00:03:05Z</entryDate>
            ...
```

Use of unnamed namespace

Discussion: ValidatePackage



ValidatePackage		
input	Pin	ValidatePackage
output	Pout	ValidatePackageResponse

Message root wrapper element

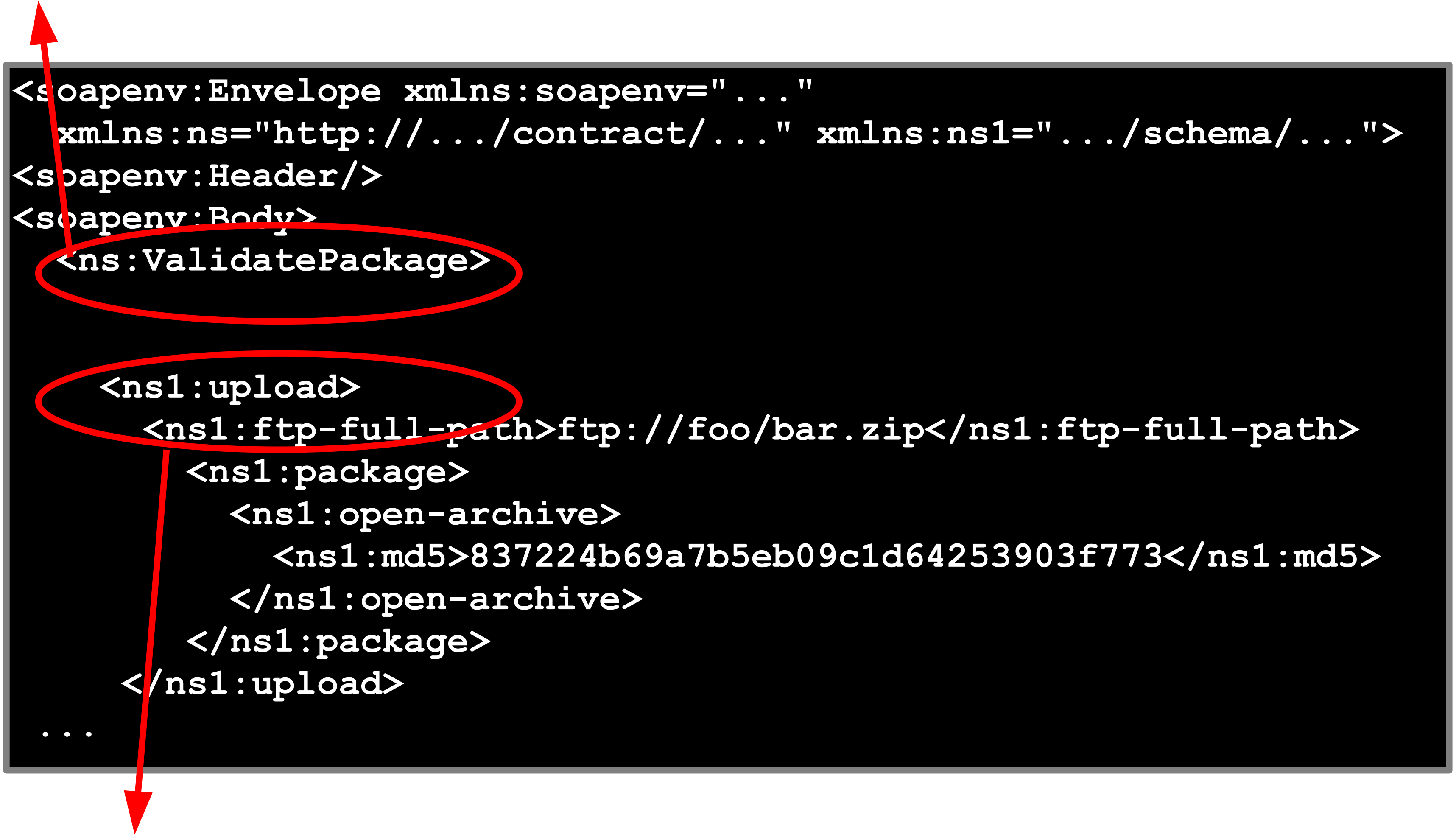
document-literal contract

XML Message: ValidatePackage

```
<soapenv:Envelope xmlns:soapenv="..."
  xmlns:ns="http://.../contract/..." xmlns:ns1=".../schema/...">
<soapenv:Header/>
<soapenv:Body>
  <ns:ValidatePackage>
    <ns1:upload>
      <ns1:ftp-full-path>ftp://foo/bar.zip</ns1:ftp-full-path>
        <ns1:package>
          <ns1:open-archive>
            <ns1:md5>837224b69a7b5eb09c1d64253903f773</ns1:md5>
          </ns1:open-archive>
        </ns1:package>
     </ns1:upload>
   </ns:ValidatePackage>
</soapenv:Body>
</soapenv:Envelope>
```

XML Message: ValidatePackage

Operation is clear; in namespace of contract



```
<soapenv:Envelope xmlns:soapenv="..."
  xmlns:ns="http://.../contract/..." xmlns:ns1=".../schema/...">
<soapenv:Header/>
<soapenv:Body>
  <ns:ValidatePackage>


    <ns1:upload>
      <ns1:ftp-full-path>ftp://foo/bar.zip</ns1:ftp-full-path>
        <ns1:package>
          <ns1:open-archive>
            <ns1:md5>837224b69a7b5eb09c1d64253903f773</ns1:md5>
          </ns1:open-archive>
        </ns1:package>
     </ns1:upload>
 ...
```

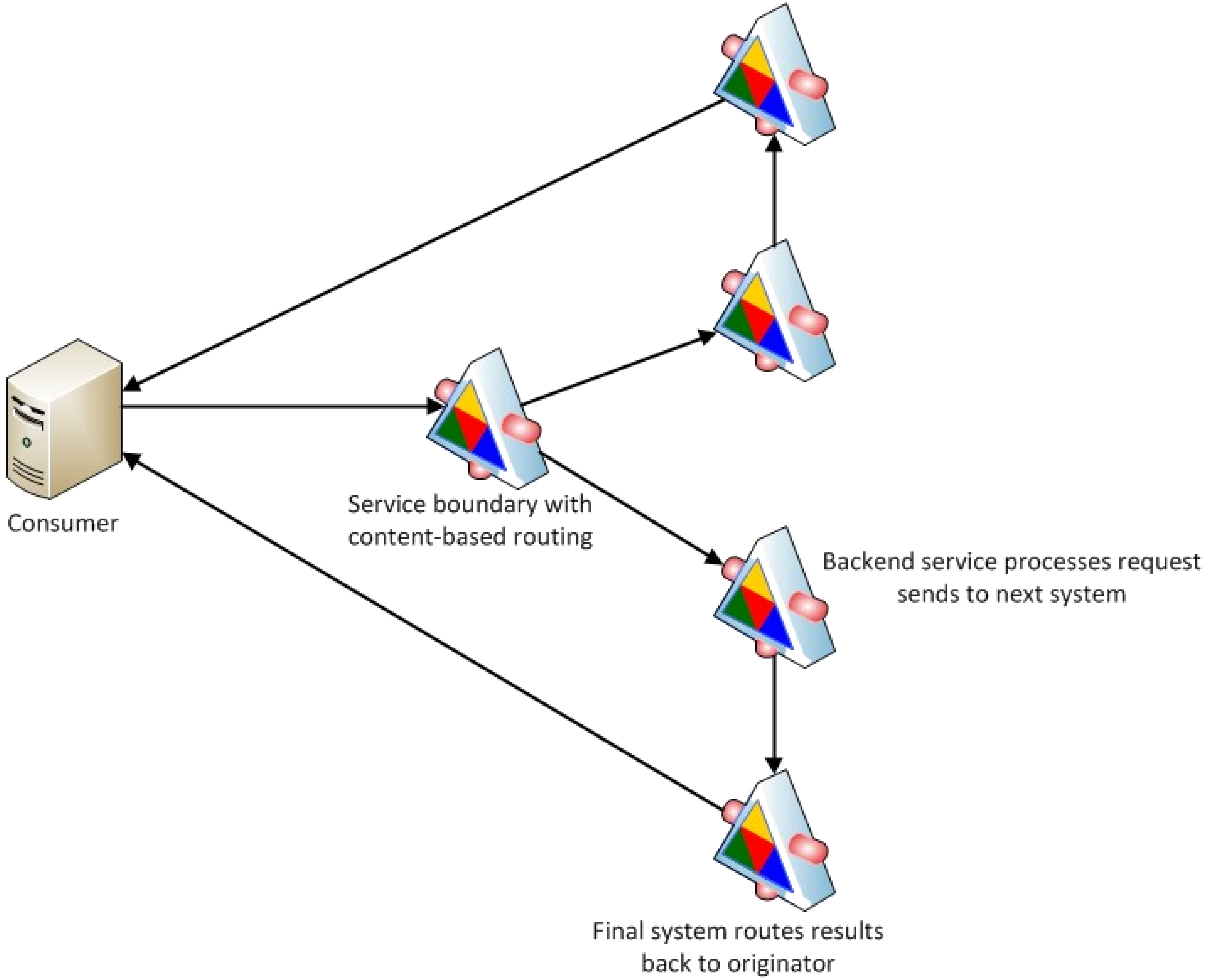
Re-used data model; in namespace of data model

Readability: ValidatePackage

```
<soapenv:Envelope xmlns:soapenv="..."
  xmlns:ns="http://.../contract/...">
<soapenv:Header/>
<soapenv:Body>
  <ns:ValidatePackage>
    <upload xmlns=".../schema/...">
      <ftp-full-path>ftp://foo/bar.zip</ftp-full-path>
        <package>
          <open-archive>
            <md5>837224b69a7b5eb09c1d64253903f773</md5>
          </open-archive>
        </package>
     </upload>
   </ns:ValidatePackage>
</soapenv:Body>
</soapenv:Envelope>
```

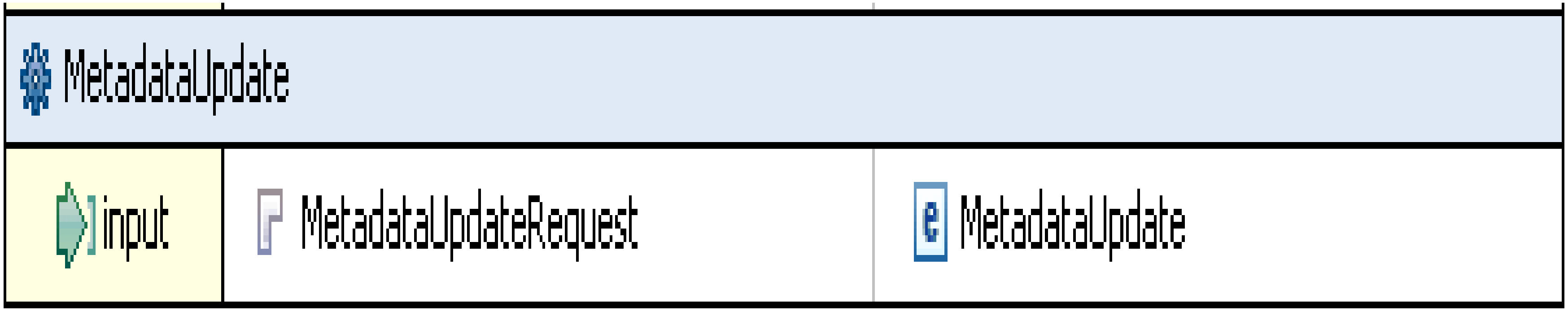
Correct usage of default namespace – should readability be a problem
(positioning of xmlns declarations are implementation-dependent)

Asynchronous contract



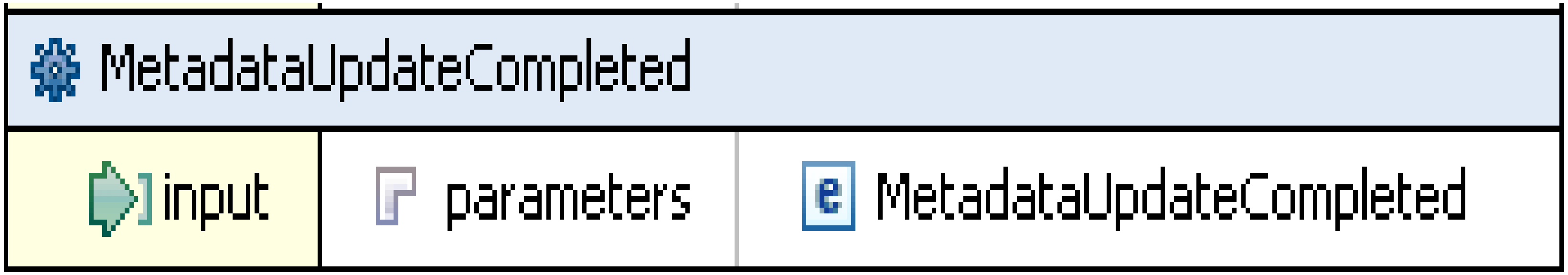
Consumer
Service boundary with content-based routing
Backend service processes request sends to next system
Final system routes results back to originator

Asynchronous Service Contract



MetadataUpdate		
input	MetadataUpdateRequest	MetadataUpdate

Interface implemented by service as per contract



MetadataUpdateCompleted		
input	parameters	MetadataUpdateCompleted

Interface to be implemented by consumer
in order to receive updates

(This is not SOA eventing)

Asynchronous Routing Header

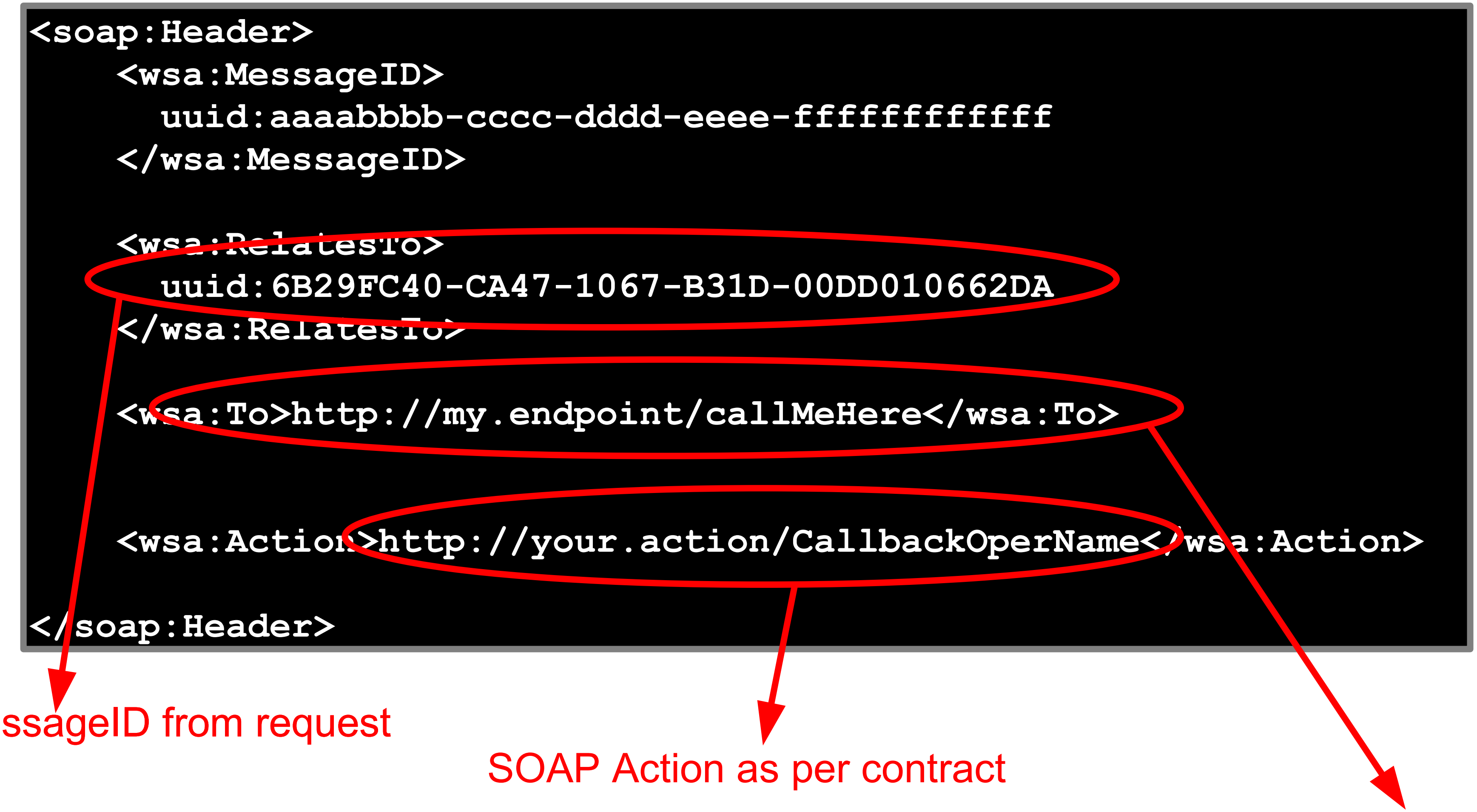
```
<soap:Header>
  <wsa:MessageID>
    uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
  </wsa:MessageID>
  <wsa:ReplyTo>
    <wsa:Address>http://my.endpoint/callMeHere</wsa:Address>
  </wsa:ReplyTo>
  <wsa:FaultTo>
    <wsa:Address>http://my.endpoint/faults</wsa:Address>
  </wsa:FaultTo>
  <wsa:To>http://your.endpoint/ServiceEndPoint</wsa:To>
  <wsa:Action>http://your.action/OperName</wsa:Action>
</soap:Header>
```

http://www.w3.org/Submission/ws-addressing/

Asynchronous Reply Header

```
<soap:Header>
    <wsa:MessageID>
      uuid:aaaabbbb-cccc-dddd-eeee-ffffffffffff
    </wsa:MessageID>

    <wsa:RelatesTo>
      uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
    </wsa:RelatesTo>

    <wsa:To>http://my.endpoint/callMeHere</wsa:To>


    <wsa:Action>http://your.action/CallbackOperName</wsa:Action>

</soap:Header>
```

Asynchronous Reply Header



```
<soap:Header>
    <wsa:MessageID>
      uuid:aaaabbbb-cccc-dddd-eeee-ffffffffff
    </wsa:MessageID>

    <wsa:RelatesTo>
      uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
    </wsa:RelatesTo>

    <wsa:To>http://my.endpoint/callMeHere</wsa:To>


    <wsa:Action>http://your.action/CallbackOperName</wsa:Action>

</soap:Header>
```

MessageID from request

SOAP Action as per contract

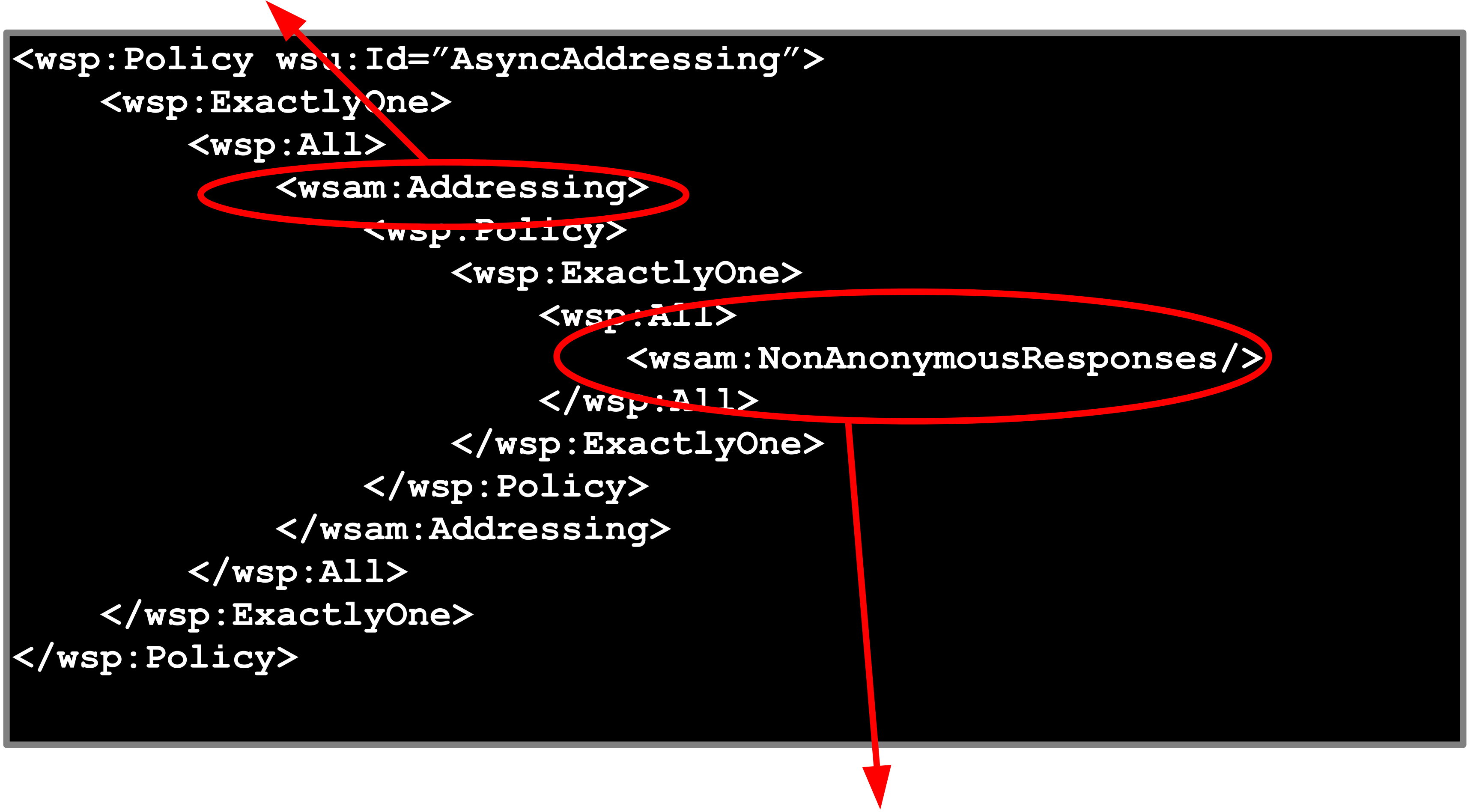
Endpoint from ReplyTo

Adding Policy to Service Contract

```
<wsp:Policy wsu:Id=”AsyncAddressing”>
    <wsp:ExactlyOne>
        <wsp:All>
            <wsam:Addressing>
                <wsp:Policy>
                    <wsp:ExactlyOne>
                        <wsp:All>
                            <wsam:NonAnonymousResponses/>
                        </wsp:All>
                    </wsp:ExactlyOne>
                </wsp:Policy>
            </wsam:Addressing>
        </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>
```

Adding Policy to Service Contract

WS-Addressing policy



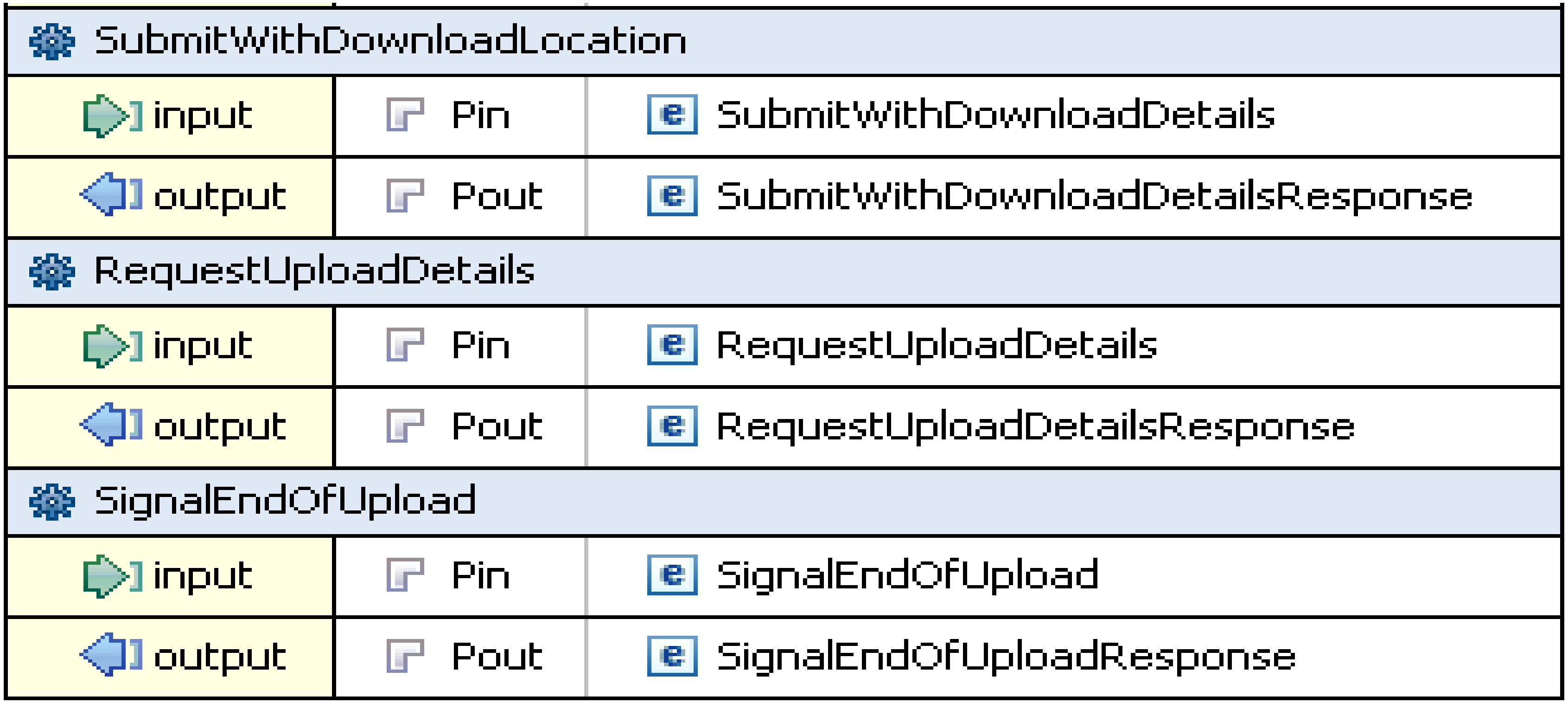
```
<wsp:Policy wsu:Id=”AsyncAddressing”>
    <wsp:ExactlyOne>
        <wsp:All>
            <wsam:Addressing>
                <wsp:Policy>
                    <wsp:ExactlyOne>
                        <wsp:All>
                            <wsam:NonAnonymousResponses/>
                        </wsp:All>
                    </wsp:ExactlyOne>
                </wsp:Policy>
            </wsam:Addressing>
        </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>
```

Locks down operation only to use one-way channels
(anonymous responses across same HTTP channel not allowed)

Handling large files

- XML is not optimised for bulk binary data
- Embedding binary data in XML can lead to unnecessary processing overhead in systems.
- Large files should be transferred out-of-band or as attachments.
- MTOM is primary means of adding attachments.

Out-of-band Transfer Operations



SubmitWithDownloadLocation		
input	Pin	SubmitWithDownloadDetails
output	Pout	SubmitWithDownloadDetailsResponse
RequestUploadDetails		
input	Pin	RequestUploadDetails
output	Pout	RequestUploadDetailsResponse
SignalEndOfUpload		
input	Pin	SignalEndOfUpload
output	Pout	SignalEndOfUploadResponse

- Request location & credentials from service, upload, notify on complete
- Provide location & credentials to service

MTOM in a Nutshell

- “Message Transfer Optimisation Mechanism”
  - Used in conjunction with “XML-binary Optimised Packaging” (XOP)
  - http://www.w3.org/TR/soap12-mtom/
- Most modern platforms are MTOM-aware
- Attachments are sent as MIME within same transport channel
- Reference identifier within message body links to attachment

MTOM Type Declaration

```
<xsd:element

  xmlns:xsd=”http://www.w3.org/2001/XMLSchema”
  xmlns:xmime=”http://www.w3.org/2005/05/xmlmime”

  name="attachmentForAccu2011"

  xmime:expectedContentTypes="application/octet-stream"

  type="xmime:base64Binary"
/>
```

MTOM on the wire

```
POST http://SomeUrl HTTP/1.1
Content-Type: multipart/related; type="application/xop+xml";
  start="<rootpart@soapui.org>"; start-info="text/xml";
  boundary="----=_Part_1_17914961.1302946204187"
MIME-Version: 1.0
Content-Length: …

------=_Part_1_17914961.1302946204187"
Content-Type: application/xop+xml; charset=UTF-8; type="text/xml""
Content-ID: <rootpart@soapui.org>
...
<attachmentForAccu2011>
  <xmime:Include href="cid:309040910934"
     xmlns:xmime="http://www.w3.org/2004/08/xop/include"/>
</attachmentForAccu2011>
...
------=_Part_1_17914961.1302946204187"
Content-Type: application/octet-stream; name=WhatIsInAGoodContract.bin"
Content-ID: <309040910934>"
...
(Binary data follows)
```

Security Aspects in Contracts

Security Policies

- Security is a difficult concept for many to grasp
- Creating the infrastructure is not easy
- Start easy with Username Tokens and not encryption + signing
- Move to SAML Tokens when the above is understood
- Add encryption + signing when STS infrastructure is in place
- Read WS-I Basic Security Profile
  - http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html

WS-Security Policy

```
<wsp:Policy wsu:Id="UTOverTransport">
<wsp:ExactlyOne>
<wsp:All>

  <sp:TransportBinding> ... </sp:TransportBinding>

  <sp:SignedSupportingTokens> ... </sp:SignedSupportingTokens>

</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

Security policy is attached to
contract at an appropriate level

```
<wsdl:binding name="MyContractBindingName"
  type="tns:MyContractPortType">

  <wsp:PolicyReference URI="#UTOverTransport"/>

  ...
```

WS-Security Policy

```
<sp:TransportBinding>
  <wsp:Policy>
    <sp:TransportToken>
      <wsp:Policy>

        <sp:HttpsToken RequireClientCertificate="false"/>

      </wsp:Policy>
    </sp:TransportToken>
    <sp:AlgorithmSuite>

      <wsp:Policy> <sp:Basic256/> </wsp:Policy>

    </sp:AlgorithmSuite>
    <sp:Layout>

      <wsp:Policy> <sp:Lax/> </wsp:Policy>

    </sp:Layout>
    <sp:IncludeTimestamp/>
  </wsp:Policy>
 </sp:TransportBinding>
```

WS-Security Policy

```
<sp:TransportBinding>
  <wsp:Policy>
    <sp:TransportToken>
      <wsp:Policy>

        <sp:HttpsToken RequireClientCertificate="false"/>

      </wsp:Policy>
    </sp:TransportToken>
    <sp:AlgorithmSuite>

      <wsp:Policy> <sp:Basic256/> </wsp:Policy>

    </sp:AlgorithmSuite>
    <sp:Layout>

      <wsp:Policy> <sp:Lax/> </wsp:Policy>

    </sp:Layout>
    <sp:IncludeTimestamp/>
  </wsp:Policy>
 </sp:TransportBinding>
```

HTTPS
required

WS-Security Policy

```
<sp:SignedSupportingTokens>
  <wsp:Policy>

    <sp:UsernameToken
       sp:IncludeToken="http://...securitypolicy..."/>

  </wsp:Policy>
</sp:SignedSupportingTokens>
```

WS-Security Policy

```
<sp:SignedSupportingTokens>
  <wsp:Policy>

    <sp:UsernameToken
       sp:IncludeToken="http://...securitypolicy..."/>

  </wsp:Policy>
</sp:SignedSupportingTokens>
```

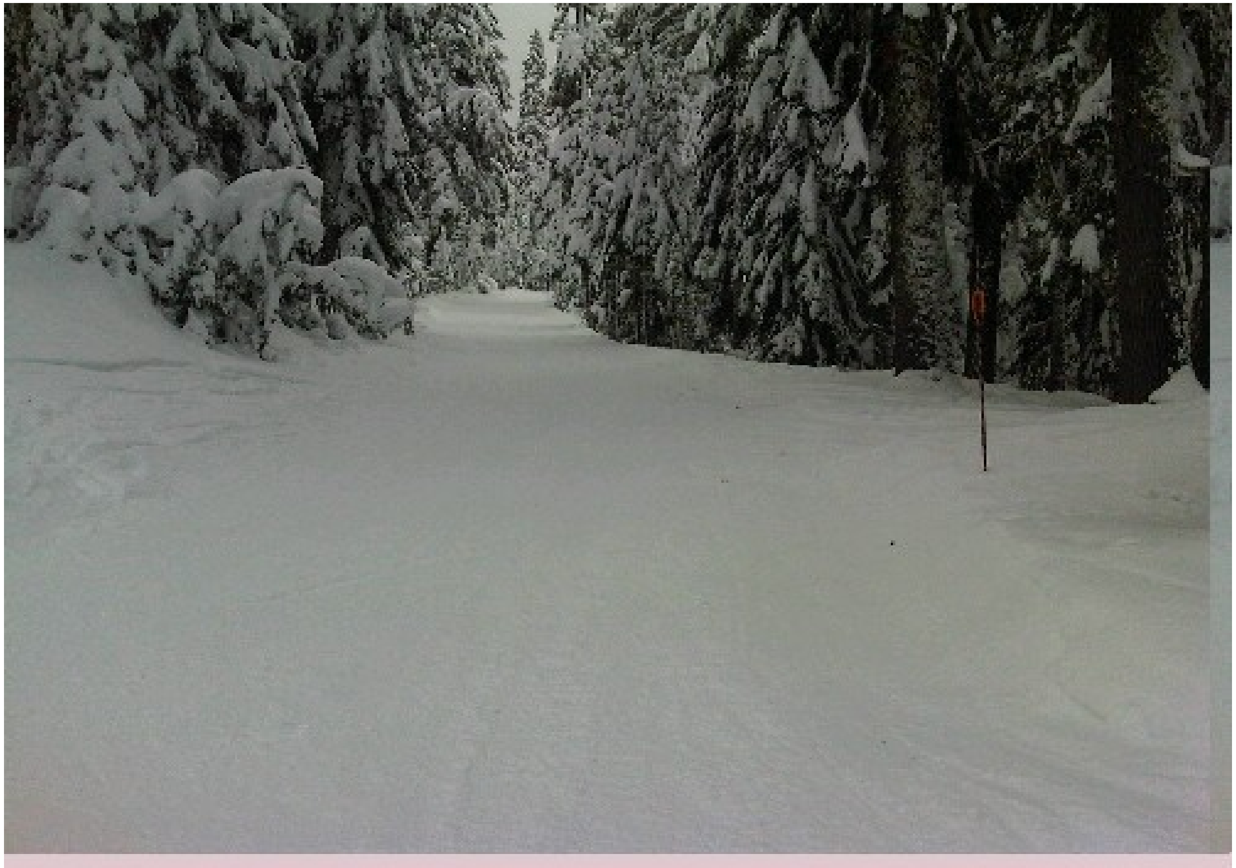
Authentication is username+password

(limited security, SAML tokens are better)

WS-Security Policy

```
<soap:Header>
  <wsse:Security>

    <wsse:UsernameToken wsu:Id="UsernameToken-8">
      <wsse:Username>${USERNAME}</wsse:Username>
      <wsse:Password Type="http://docs.../#PasswordText">
      ${PASSWORD}
      </wsse:Password>
    </wsse:UsernameToken>

    <wsu:Timestamp wsu:Id="Timestamp-7">
      <wsu:Created>2011-04-15T13:36:10.357Z</wsu:Created>
      <wsu:Expires>2011-04-15T16:22:50.357Z</wsu:Expires>
    </wsu:Timestamp>

  </wsse:Security>
</soap:Header>
```

TDD & Service Contracts

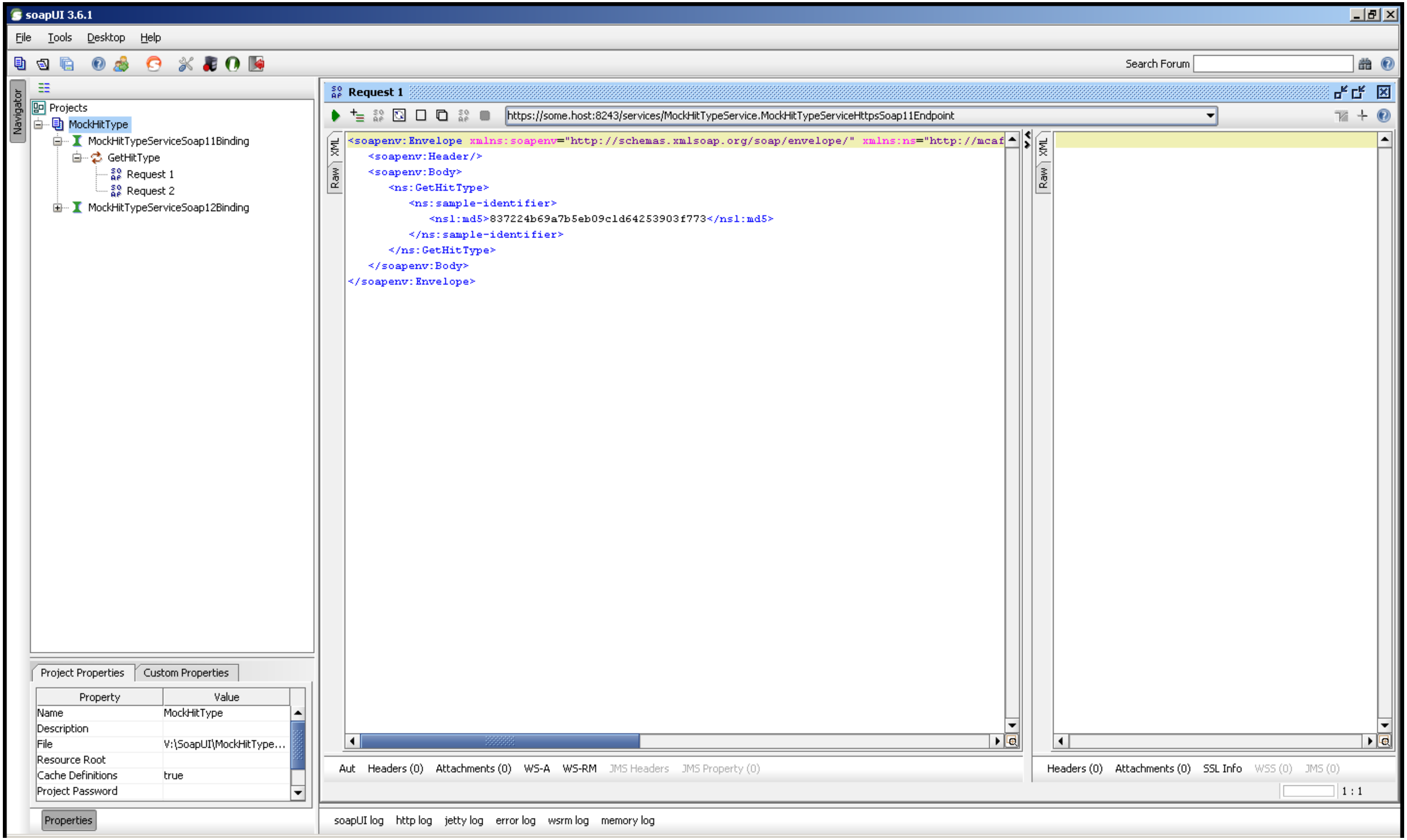


How to add testing to a contract without implementing the service

Using TDD

- Conventional test-first is very hard and impractical
- Iterative process of designing operation then generating test code
- SoapUI is an efficient tool for TDD of contracts
- Initial tests can be extended to become first set of integration tests
- Initial tests become a living specification
- Helps with documenting the contract

Test Request Message



https://some.host:8243/services/MockHitTypeService.MockHitTypeServiceHttpsSoap11Endpoint

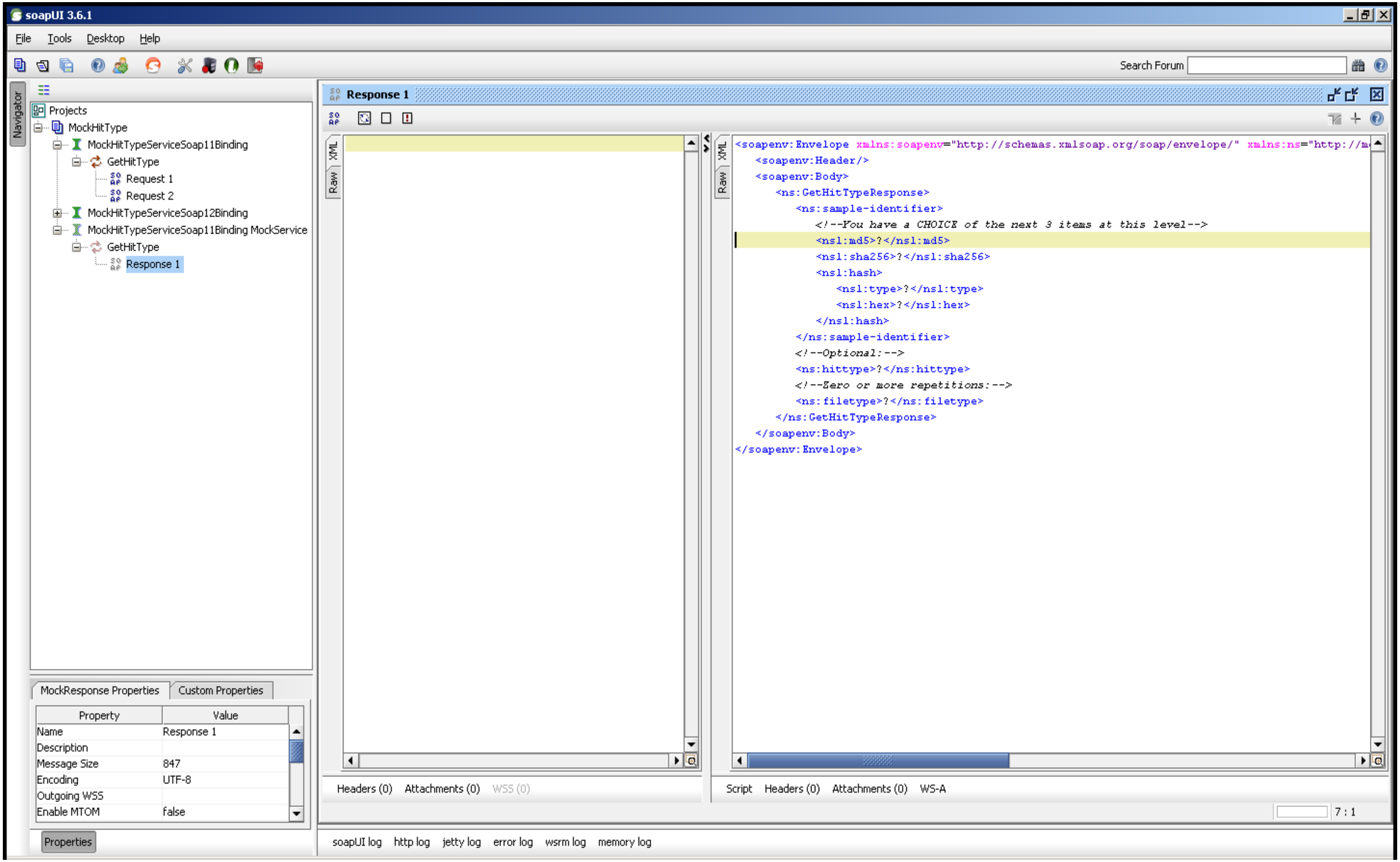
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns="http://mcaf
   <soapenv:Header/>
   <soapenv:Body>
      <ns:GetHitType>
         <ns:sample-identifier>
            <nsl:md5>837224b69a7b5eb09c1d64253903f773</nsl:md5>
         </ns:sample-identifier>
      </ns:GetHitType>
   </soapenv:Body>
</soapenv:Envelope>
```

Project Properties | Custom Properties

Property	Value
Name	MockHitType
Description	
File	V:\SoapUI\MockHitType...
Resource Root	
Cache Definitions	true
Project Password	

ACCU 2011
© Schalk W. Cronjé

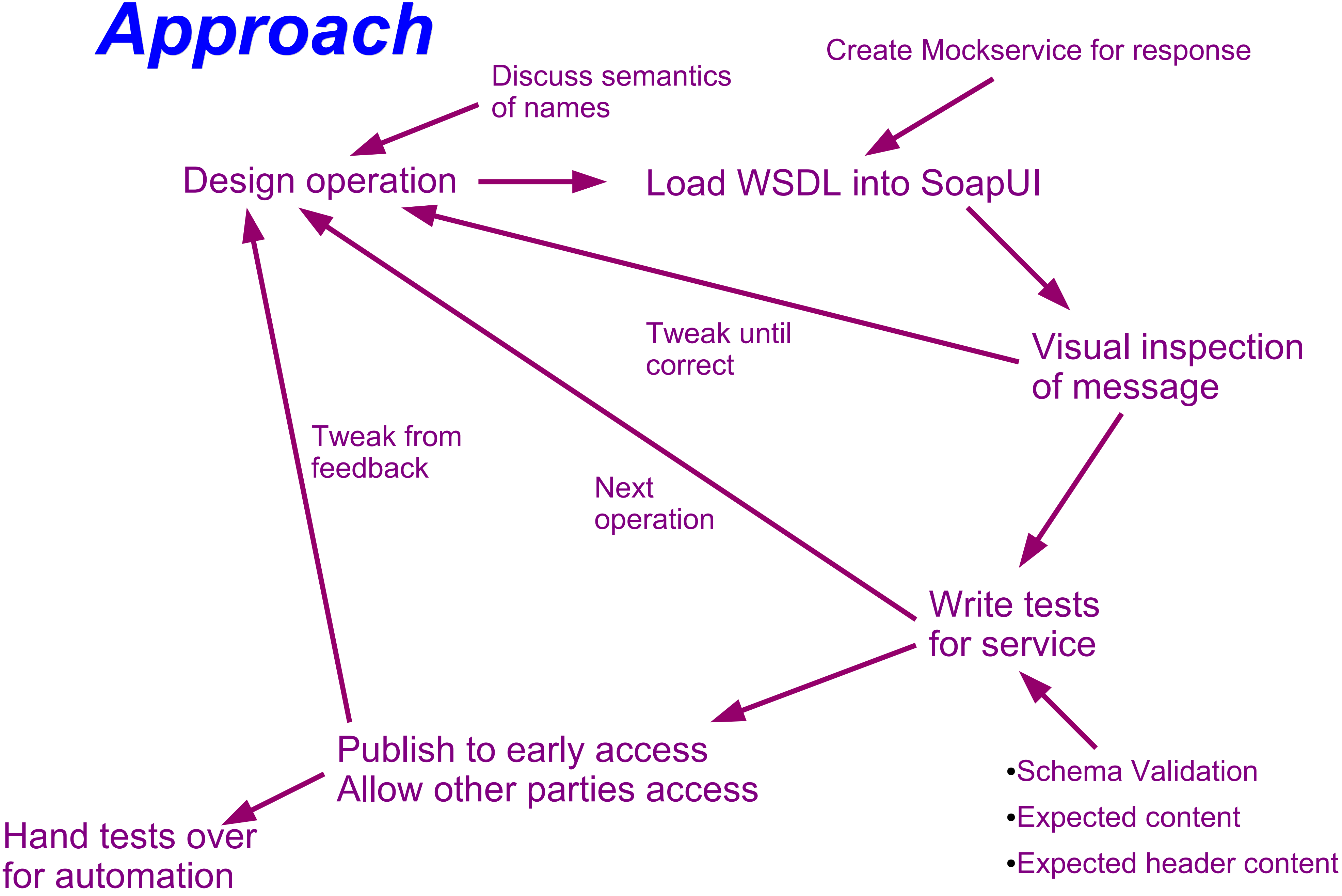
Add Mockservice to Test Response



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns="http://m
   <soapenv:Header/>
   <soapenv:Body>
      <ns:GetHitTypeResponse>
         <ns:sample-identifier>
            <!--You have a CHOICE of the next 3 items at this level-->
            <nsl:md5>?</nsl:md5>
            <nsl:sha256>?</nsl:sha256>
            <nsl:hash>
               <nsl:type>?</nsl:type>
               <nsl:hex>?</nsl:hex>
            </nsl:hash>
         </ns:sample-identifier>
         <!--Optional:-->
         <ns:hittype>?</ns:hittype>
         <!--Zero or more repetitions:-->
         <ns:filetype>?</ns:filetype>
      </ns:GetHitTypeResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Property	Value
Name	Response 1
Description	
Message Size	847
Encoding	UTF-8
Outgoing WSS	
Enable MTOM	false

Approach



Discuss semantics of names

Create Mockservice for response

Design operation

Load WSDL into SoapUI

Visual inspection of message

Tweak until correct

Tweak from feedback

Next operation

Write tests for service

- Schema Validation
- Expected content
- Expected header content

Publish to early access
Allow other parties access

Hand tests over for automation

What makes a Good Contract?

No implementation technology exposed

Implementation Technology

- Can cause unnecessary vendor lock-in for the service implementer
  - Cannot change the back-end without changing the contract
- Can convey the message that contract has been designed unprofessionally

Contract-first design

Contract-First Design

- Leads to cleaner interfaces
- Maps better to business requirements
  - Consumer involvement makes the learning and writing process easier.
- Achieves intrinsic interoperability
- Avoids exposing bugs in the implementation technology and code generators

Intuitive, business-centric names

Naming

- Self-explanatory operation names
  - GetCustomerNameByEmailAddress
- Names in data models must reflect the domain models
- Names should be as simple as possible, but no simpler (“ReplyTo” vs “rt”)
- Names should make it easier to understand the contract

Element form default is qualified

Qualified Elements

- Unambiguous data models
- Unnamed namespaces do not cause issues during message aggregation / splitting

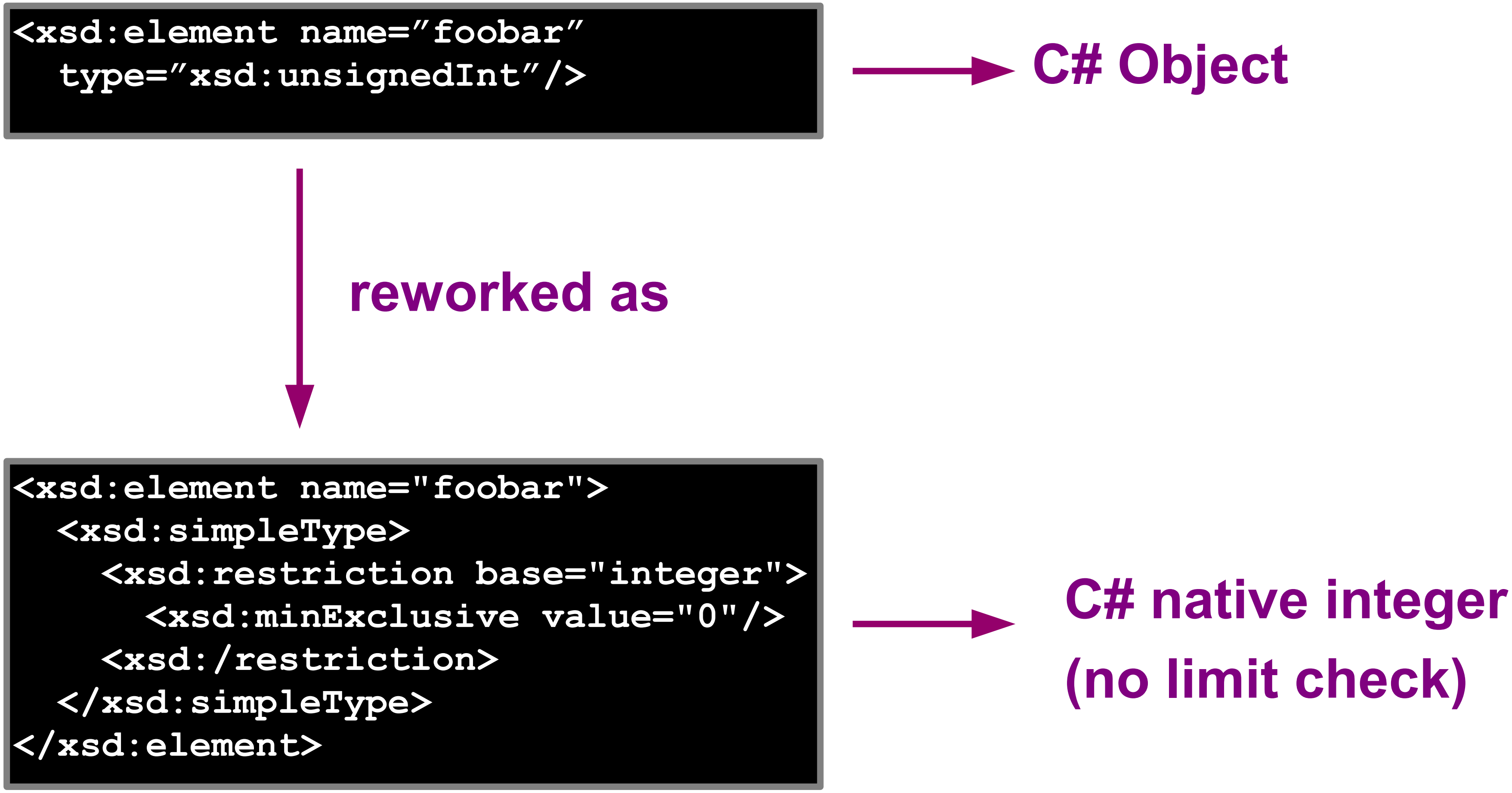
```
<xsd:schema
  attributeFormDefault="unqualified"
  elementFormDefault="unqualified"
  targetNamespace="http://.../contract/.../2.0">
```

Accommodates known interoperability issues

Platform Idiosyncrasy

- Some technologies have well-known issues
- If known this can be addressed in the contract in a portable manner
- Compromises should affect the contract in a negative way.
- Try out code generators from various platforms and study the artefacts.
- If in doubt, the standards (W3C/Oasis/WS-I) are the law.

svcutil.exe idiosyncrasy



```
<xsd:element name=”foobar”
  type=”xsd:unsignedInt”/>
```

C# Object

reworked as

```
<xsd:element name="foobar">
  <xsd:simpleType>
    <xsd:restriction base="integer">
      <xsd:minExclusive value="0"/>
    <xsd:/restriction>
  </xsd:simpleType>
</xsd:element>
```

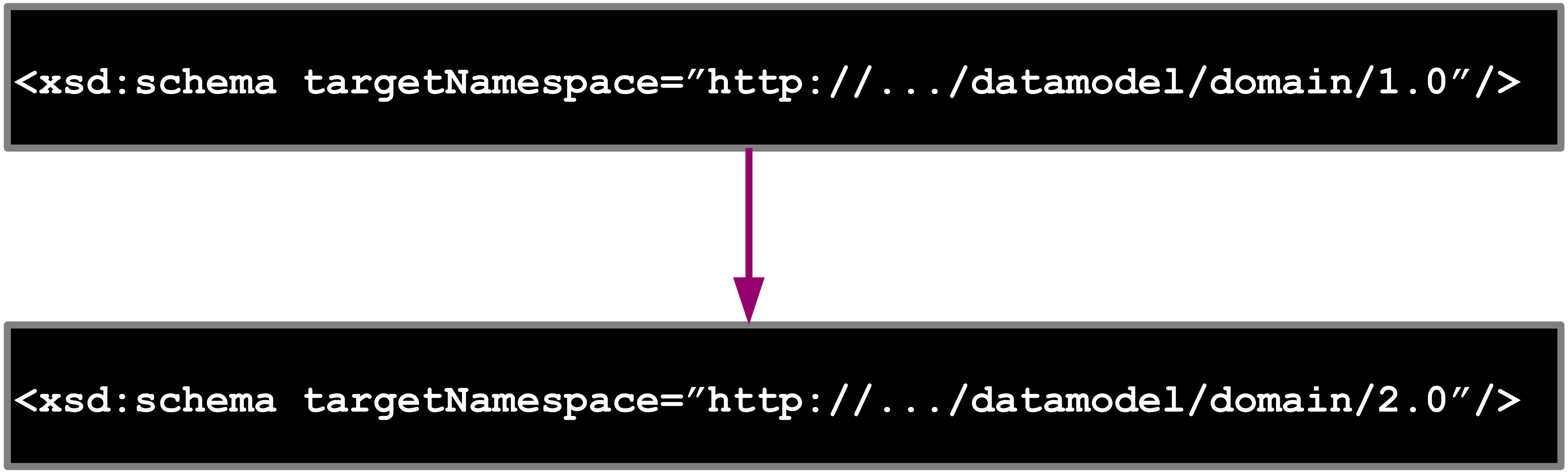
C# native integer

(no limit check)

Namespace versioned

Namespace versioning

- XML world has two approaches
  - Attribute “version” on root element
  - Namespace versioning
- XML Service world prefers namespace versioning



```
<xsd:schema targetNamespace=”http://.../datamodel/domain/1.0”/>
```

```
<xsd:schema targetNamespace=”http://.../datamodel/domain/2.0”/>
```

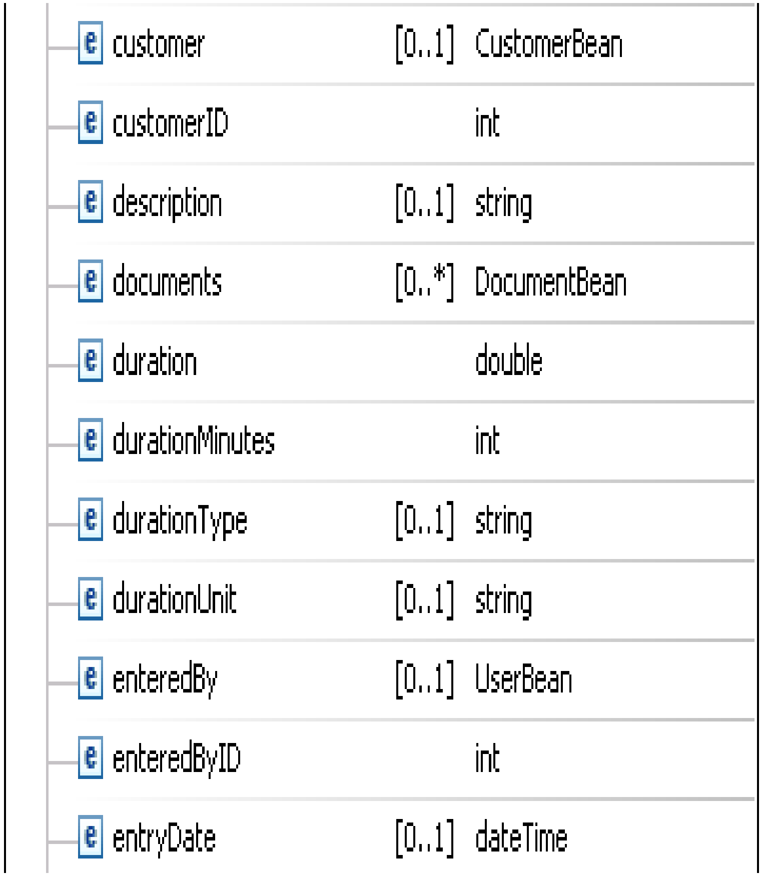
Namespace versioning

- Allows for breaking original structure, but keeping the core of the domain model intact
- Allows mixing content from both versions in a document, but maintaining clear boundaries
- Parsers can be maintained independently

No data model surprises

Consistency

- Specified multiplicity must map to the data model
- If technology does not directly support constraints, implement it in the code
- Ensure test cases cover data model validation



e	customer	[0..1]	CustomerBean
e	customerID		int
e	description	[0..1]	string
e	documents	[0..*]	DocumentBean
e	duration		double
e	durationMinutes		int
e	durationType	[0..1]	string
e	durationUnit	[0..1]	string
e	enteredBy	[0..1]	UserBean
e	enteredByID		int
e	entryDate	[0..1]	dateTime

Non-functional aspects decoupled

Non-functional aspects

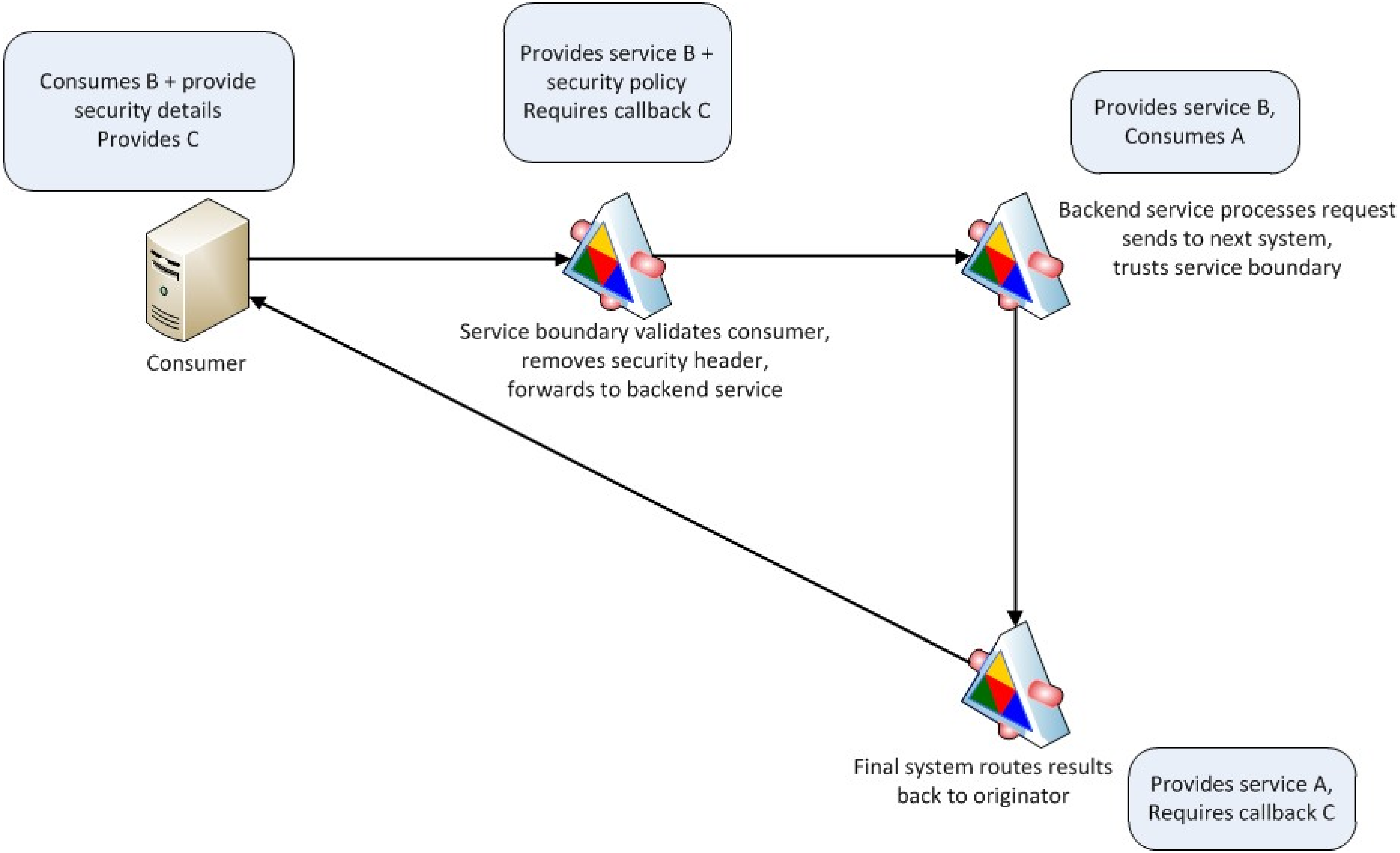
- Aspects such as security, routing, eventing and notification decoupled from the data model
- Contained in own schemas
- Attached as policies to the contract
- Allows for various groups to focus on specific dimensions in contract design without being inter-dependant

Characteristics of Good Contracts

- No implementation technology exposed
- Contract-first design
- Intuitive, business-centric names
- Qualified elements
- Interoperability issues accommodated
- Namespace versioned
- No data model surprises
- Non-functional aspects decoupled from data model

Wrapping up

Composability can be easy



Consumes B + provide security details Provides C

Provides service B + security policy Requires callback C

Provides service B, Consumes A

Backend service processes request sends to next system, trusts service boundary

Consumer

Service boundary validates consumer, removes security header, forwards to backend service

Final system routes results back to originator

Provides service A, Requires callback C

Understand the costs

- Service contracts are not an afterthought
- Use contract-first design
- Time need to be invested up-front prior to any develop to publish and agree on a suitable interface
- Some tweaking might be required during initial development, but should be clearly communicated to all involved parties via early access program

Governance

- Once the contract is published to production it is not allowed to be modifed, only extended
- Modifications require (namespace) versioning, effectively becoming a new contract

Testing

- Testers need to understand the domain + the requirements of the various policies
- Must be able to test service direct and interpret XML messages
- Must be able to automate tests that validate the contract using both positive and negative tests

Thank you

Schalk W. Cronjé
ysb33r @ gmail.com
@ysb33r