



Designing Service Collaborations: The Design of 'Wire'-Centric Integration

Mark Hapner and Gopalan Suresh Raj

Sun Microsystems, Inc.
<http://www.sun.com>

TS-8897



The 'wire' is the 'computer'

In the past, integration design was focused on middleware.

Today, the internet is the focus. 'Wire' design is global, non-proprietary, and platform-independent.

This session presents an overview of 'wire' centric design and describes several 'wire' design best-practices.

Agenda

Focus shifts to the 'wire'

What the 'wire' isn't

What the 'wire' is

'Wire' design

'Wire' design best practices

Agenda

Focus shifts to the 'wire'

What the 'wire' isn't

What the 'wire' is

'Wire' design

'Wire' design best practices

The 'Wire'

... or the 'what' beyond my code

Network context

- Code's view of its 'wire' consumers
- Code's view of its 'wire' suppliers

Network community

- Attracting and retaining consumers
- Leveraging suppliers
- Stability with constant change
- A 'wire' economy

'Wire' Supplier Concerns

Consumer is king

Evolve to survive

Consumer tail is the 'wire'

The 'wire' is the product

- Make it flexible

- Provide a full consumer view

'Wire' Consumer Concerns

Depending on suppliers is risky

It's useful to reverse proxy supplier 'wires'

Important 'wire' properties

- Visibility

- Stability

- Security

- Performance



eBusiness Is the Goal

'wire' infrastructure

'wire' collaboration

'wire' centric design

Agenda

Focus shifts to the 'wire'

What the 'wire' isn't

What the 'wire' is

'Wire' design

'Wire' design best practices



The 'Wire' Isn't a Program

It isn't a remote procedure

Don't use it to pass control

Code composition principles don't apply to it

The 'Wire' Isn't a Component Model

It doesn't have a constructor

It doesn't need to be configured

It isn't 'reused'

It isn't 'contained'



The 'Wire' Isn't Reliable

The network is unreliable

Suppliers and consumers

- Fail

- Have bugs

- Duplicate work



The 'Wire' Isn't Static

Evolution is the rule

Coordinating change is hard

The 'wire' has a tail

The 'Wire' Isn't an Architecture

Architectures are conceptual abstractions

The 'wire' is not an abstraction

- It doesn't exist until it's available for use

- An abstract supplier is useless

- The 'wire' can't be separated from the stack of internet standards that are its foundation

Agenda

Focus shifts to the 'wire'

What the 'wire' isn't

What the 'wire' is

'Wire' design

'Wire' design best practices

The 'Wire' Is Global

Global : no technical barriers to global access

A non-global 'wire' is a contradiction-in-terms

Global access is a fundamental 'wire' attribute

Global computing is the HTTP stack

Be suspicious of non-HTTP 'wires'

The 'Wire' Is Peer-to-Peer

Many collaborations require participants to be both suppliers and consumers

Client-server concepts are too limiting

- Session 'cookies' aren't useful

- Pseudo conversation modes aren't useful

Correlation is at the business layer not the protocol layer.

The 'Wire' Is Asynchronous

The 'wire' spans sessions

The 'wire' interleaves its work

The 'wire' is more ad hoc than conventional integration

The 'wire' requires correlation to function

The 'Wire' Is a Network Resource

The nature of network resources continues to evolve

Composition of network resources defines a new computing space

- It is fundamentally different than code composition

- In some ways it has become more important than code composition

- We have experience with the browser aspect of network composition but are just beginning to learn about 'wire' composition

Agenda

Focus shifts to the 'wire'

What the 'wire' isn't

What the 'wire' is

'Wire' design

'Wire' design best practices

Message Exchange Patterns (MEPs)

The units of 'wire' collaboration are MEPs

- HTTP Put, Post, Get, Delete

- WSDL 2.0 MEPs

- AS 2.0 MEPs

Transient message exchanges

- Within HTTP session

- Transport single message and quick
acknowledge/response

Design focus

- Select MEP

- Define the message/response it transports

Conversations

Correlated message exchanges

- Span HTTP sessions

- Related by correlation values exposed via message properties

- Peer-to-peer

Design Focus

- Define the roles

- Define the correlations

- Define the life cycle

- Define the shared state

Shared State

The semantics of collaboration

Conversation

The message exchanges used to move the shared state of a collaboration through its life cycle

Shared state

The state that the roles in a collaboration semantically share as the representation of their common goal

Visibility of shared state

Shared state may be implied but not accessible

Correlation values are 'links' to shared state

The more visible its shared state is, the more stable a collaboration is

Collaboration Evolution

Collaborations aren't static

Participants don't evolve in lock-step

The tail of a collaboration must continue to function

While the XML messages evolve well, the XML Schema that describe them don't

- Don't assume that message evolution can be contained within a single XML Schema

- Don't over-complicate message schemas with 'extension points'

- Assume that new versions of messages may use new schemas

The 'wire' is the collaboration

The 'wire' is the technical definition of a collaboration

The simpler it is, the easier it is to collaborate

- Use the least complex, most universal 'wire' possible

- Decide what not to use

- Visibility is important for stability

'Wire' policies are collaboration policies

- Collaboration participants have internal policies just like they have internal state and semantics

Agenda

Focus shifts to the 'wire'

What the 'wire' isn't

What the 'wire' is

'Wire' design

'Wire' design best practices

#1: Contract First Development (top-down)

Also called Design By Contract

Create the data contract in XSD and the behavioral contract in WSDL upfront.

Use the XSD and WSDL editors provided by the NetBeans™ Software SOA Pack to do this.

This approach forces the designer to focus on messages and contracts as the key concepts in designing a service contract.

#2: Version Number your Message Exchange Patterns (MEP)

Explicitly add a version number attribute to the first element of both the request and the response message.

This will ensure the Message Exchange Pattern (MEP) and the schema of the exchanged messages evolve in tandem

You now have complete control over how to evolve the MEP

#2: Version Number your Message Exchange Patterns (MEP) ...

```
<!-- Version 1 message -->
<?xml version="1.0" encoding="UTF-8"?>
<person version="1">
  <name>John Doe</name>
</person>
```

```
<!-- Version 2 message -->
<?xml version="1.0" encoding="UTF-8"?>
<person version="2">
  <firstName>John</firstName>
  <lastName>Doe</lastName>
</person>
```

#3: Prefer use of document/literal rather than RPC

Prefer 'document' encoding and 'literal' use over other types for interoperability.

It is better-suited for coarse-grained interactions and better represents the data exchanged

It provides the ability to validate the XML data if the XML Schema is available.

Ability to Transform messages using XSLT easily

Provides better performance than other encoding/use styles

Since the service interface in the WSDL clearly defines the types of documents expected, it makes it easy for the consumer

#4: Introduce a unique top-level element name

Within a collaboration each Message Exchange pattern should deal with unique messages

To facilitate this, introduce a unique top-level element name in each message to identify the message in the schema

This will ensure that each message is self-defining

This name can also serve as a business context for what to do with the message

#5: Use message Wrapper to contain multiple business objects

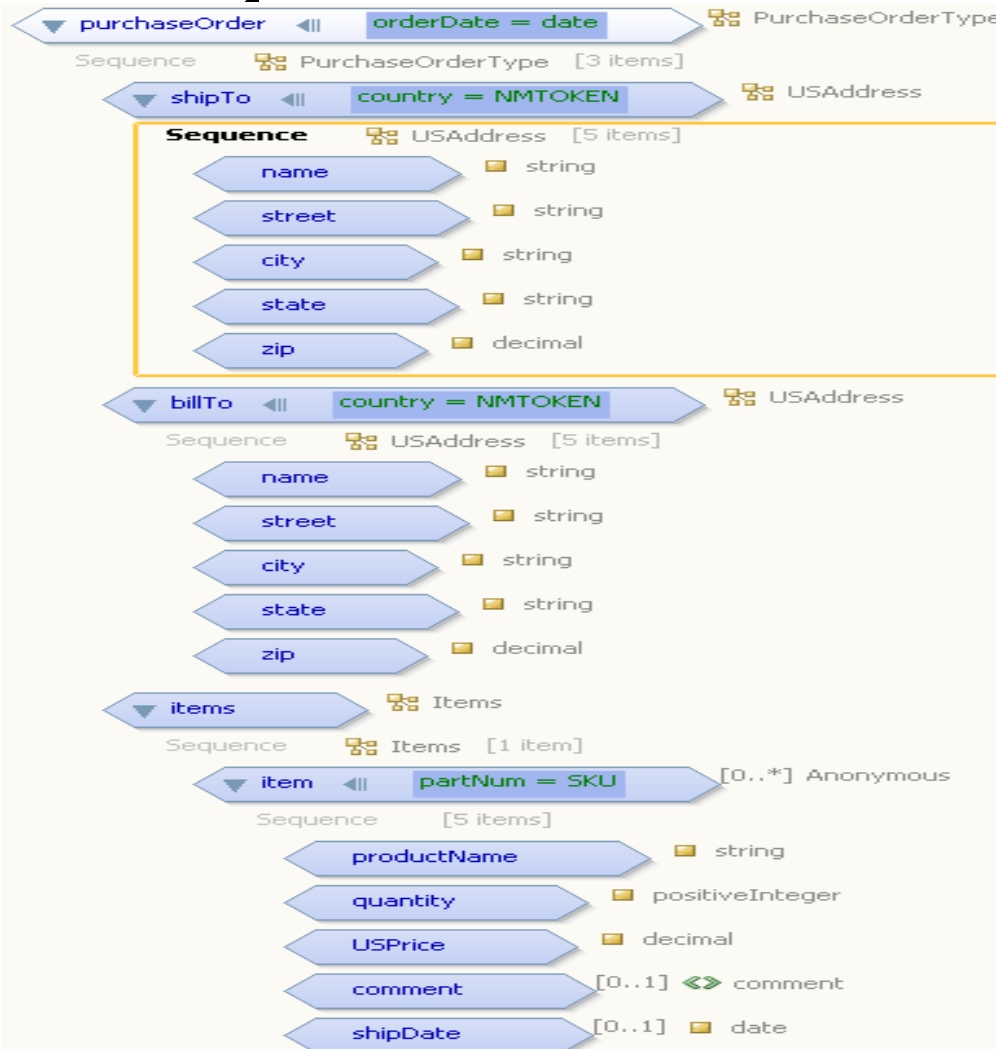
If you have a complicated interchange that has multiple business objects to it, aggregate these messages together by wrapping them

As mentioned in the previous slide, ensure that the container of the wrapped messages has a top-level element name

This wrapper can serve as an open-ended container that sets the business context for what to do with these messages

This container context ensures these messages are self-defining

#5: Use a Wrapper to contain multiple business objects ...



#6: For large messages use MTOM

If the exchanged Business Objects are large, use MTOM

Create a wrapper message that can contain multiple different types of files.

The wrapped message can then serve as a virtual message

Use multi-part MIME to physically carry the message so at any point, you don't have to parse the entire message at once

Caveat: Do not use SOAPAction to route messages

#7: Identify Shared Conversational State upfront

Conversations have an implied semantic state that participants can share and refer to the shared state in the message body to keep track of message relationships

Identify element(s) in the message that can serve as a shared state-holder, e.g., unique business specific identifiers that are customer specific, like SSNs, Claim Numbers, PO ids, etc.

Ensure that these values are easy to find in the message body by clearly defining XPath queries to extract the identifier(s) from within each message.

#8: Link messages with shared state using correlation values

Share conversational state on the wire using self-defined correlations in the message

This ensures that subsequent processes in the collaboration tail can correlate the response to a specific request

The identifier(s) placed in the Message Property can serve as correlation identifier(s) that tie together message instances with a particular conversation

#9: For asynchronous, peer-to-peer collaborations use multiple MEPs

MEPs are transient message exchange elements

If there is work that can be accomplished in a single, one-shot, stateless, self-contained collaboration, use a single MEP.

For long-running, conversational, peer-to-peer collaborations, where there is an asynchronous lag between a request and a response with shared state use multiple MEPs with correlations.

You can't have an asynchronous collaboration without having peer-to-peer message exchanges

#10: Place the business acknowledgment in a separate MEP

When dealing with stateful interactions, mandating a business response as part of a single request/response MEP overly restricts the asynchronous collaboration that is required

Often the business response is not available quick enough to place the acknowledgment into the response

The only way the business response can fully be decoupled from messages that produce them is to place the acknowledgment in a separate MEP

#11: Always go Forward never back

Compensation is an application level function that defines the semantics for resolving issues that come up with in-flight instances. It is better handled by providing application level 'cancellation' functions

Think of the Collaboration as an entity that relentlessly pushes forward - it may change but it never 'goes back' to a previous state

If things get hopelessly stuck, then cancel and take whatever business hit cancellation costs (such as canceling a nonrefundable flight reservation).

#12: Don't expose unnecessary details in the service contract

When exposing your existing applications as services:

- Analyze your existing architecture

- Define a service contract

- Define a logical architecture that you want to migrate to

- Refactor your existing architecture by exposing appropriate facades to match the service contract

- Migrate your implementation to conform to the logical architecture

In service-orientation you are exposing services not objects.

#13: The Collaboration exists even if the implementations don't

The Collaboration design is a complete design element that captures the full semantic content of the collaboration and stands apart from the applications

Throw away the implementation of the roles and still be left with a complete collaboration design that documents well-defined semantics

A third-party should be able to follow everything that's going on, about all the messages that flow through the system by just looking at the design

#14: The Message Body has to stand alone – that's the business data

A message has a header and a body

However when you want to persist the data in a database for further processing, you only persist the message body since that contains the business data

Therefore the Message body has to stand alone – do not place information that you will need to reuse to process a collaboration in the Message header

#15: Don't use WS-Addressing cookies for acknowledgments

WS-Addressing requires you to use cookies in the header to perform callbacks

However, in an asynchronous, peer-to-peer exchange scenario, where the response is not immediately available, the data may have to be persisted to database until the business response is available

In such a case, the message body that has the business data and the correlation values will be persisted to the database and the header with the WS-Addressing cookie data will be thrown away

#16 Maintain secure conversation with users often communicating with you

Efficient strategy is to maintain a secure conversation with those users who are often communicating with each other in a fairly high bandwidth way

Based on amount of communication between the provider and a consumer, consider setting up a secure conversation with partners that you are frequently communicating with



JavaOne

Project Open Enterprise Service Bus (Open ESB)

A True Open SOA Community

Open ESB 2.0 Beta 2 – Available Now!

JB1 based SOA Integration platform

Open Standard, Open Source, Interoperable

Build composite applications leveraging existing applications and webservice

An extensible platform with pluggable architecture

Integrated runtime with GlassFish™ V2

Integrated Tooling through NetBeans release 6.0

Available in Java Platform, Enterprise Edition (Java EE platform) SDK

Community-based JBI component development

JB1 – An open standard for SOA based integration platform

Rich set of Service Engines including BPEL, IEP, XSLT, Java EE platform, Aspects, WLM, Data Mashups, Encoder

Exhaustive list of Binding components including, Http, Java DataBase Connectivity (JDBC™), Java Message Service (JMS), MQ, SAP, Email, CICS, IMS, SAP and many more

free to download, and deploy

<http://open-esb.org>

<http://blogs.sun.com/gopalan>

JB1 = Java Business Integration





Q&A

Mark Hapner and Gopalan Suresh Raj
Sun Microsystems, Inc.
<http://www.sun.com>



Designing Service Collaborations: The Design of 'Wire'-Centric Integration

Mark Hapner and Gopalan Suresh Raj

Sun Microsystems, Inc.
<http://www.sun.com>

TS-8897