TRANS-ENTERPRISE

Asynchronous Web Services Protocol

Introduction by examples

Version 1.0 April 7, 2002 Jeffrey Ricker

Abstract

This document provides an introduction by example to Asynchronous Web Services Protocol (AWSP). We have created an electronic commerce scenario to demonstrate the key capabilities of AWSP. With this scenario, we will trace the exchange of specific documents between applications. It is intended that this document will provide you the level of understanding to begin experimenting and implementing an AWSP solution.

This document assumes an understanding of XML, Java and HTTP.

Table of contents

1 Introduction

1.1 Scenario explained

Big Aircraft Corp has begun work on the new stretch 838-800 aircraft. There are new requirements for bulkhead pass through connectors. The manufacturer is Mendix. However, the standard socket plugs supplied are inadequate to meet specification. Midwest Aircraft Supply makes the proper plugs required. Also required are new wire harnesses to accommodate the additional airframe length and bulkhead separation. Big Aircraft Seattle will have the harnesses constructed at the Big Aircraft Wichita Plant.

Trans-enterprise connects the Seattle based 838-800 manufacturing team with Mendix in Smallville, Kansas, Midwest Aircraft Supply in Rockford, Illinois and Big Aircraft Wichita in Wichita, Kansas in the same way.

- Big Aircraft Wichita will receive the order in their Inventory Request System, Procurement, Contracts, Engineering and Product Management via their automated system. Currently, this process is handled by e-mail and hand entered into the automated system. Big Aircraft Wichita was, until ten years ago, McDougal-Duffy Corporation. Their internal systems are completely different than Big Aircraft Seattle.

- Mendix has a mix of automation and manual operations that will link the different organizations needed to complete the fulfillment of the connector order.

- Midwest Aircraft Supply relies on Dave Stephenson, the inventory clerk, to see that the order is filled properly.

These systems are all linked seamlessly to the Big Aircraft Seattle automated system and they see all three interfaces as tied into that system even though they are in different phases of automation themselves.

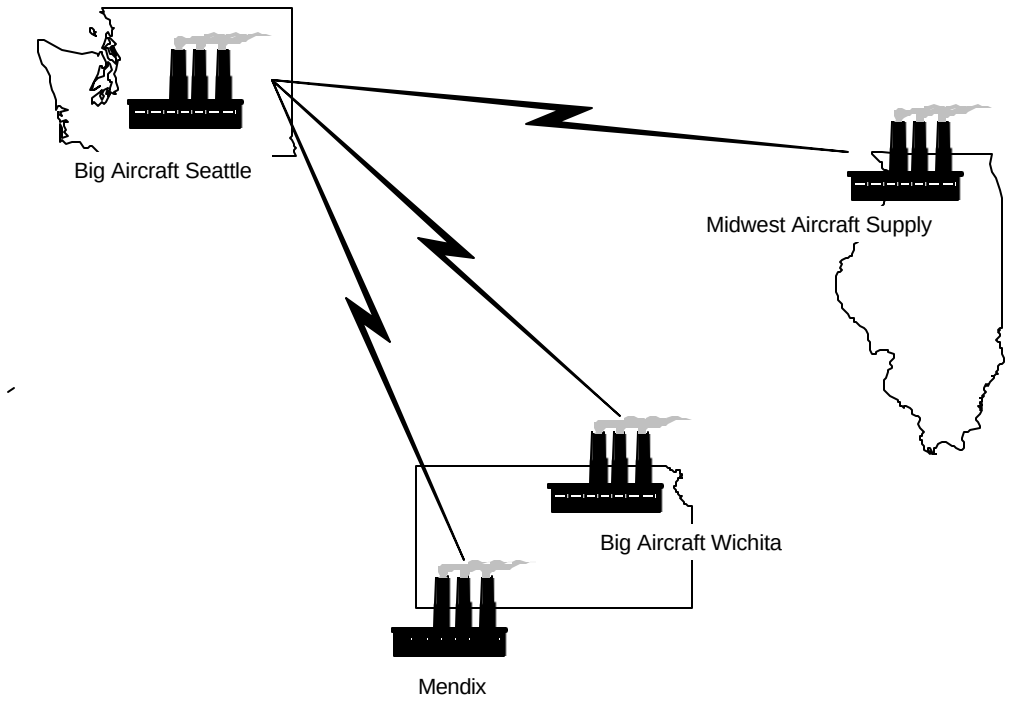

Figure 1 Participants

We have created an electronic commerce scenario to demonstrate the key capabilities of AWSP. With this scenario, we will trace the exchange of specific documents between applications. Hopefully this will provide you the level of understanding to begin experimenting and implementing an AWSP solution.

2 Integrating a small supplier

The two companies represented in this scenario by their AWSP servers: Big Aircraft Seattle with enoch.seattle.bigair.xom and Midwest Aircraft Supply with asher.midwest-air.xom. Note that we use the imaginary XOM top level in order to avoid any possible collision with real servers.

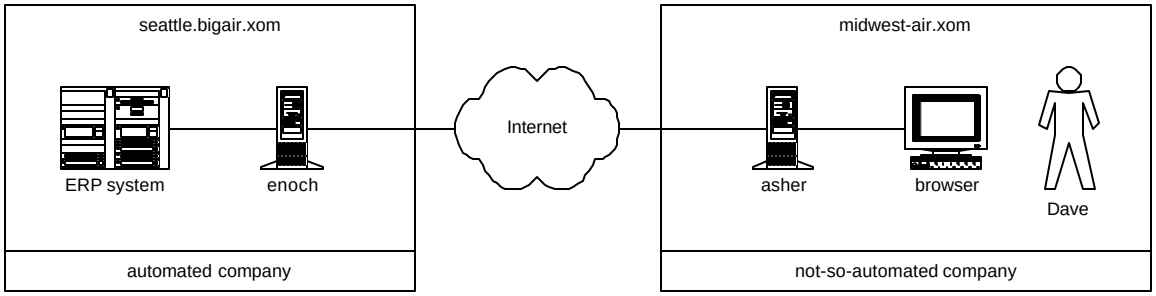

Figure 2 Scenario architecture

First, we will trace the messages from the perspective of the resource requesting the service, that is, the observer. Big Aircraft's system is the observer. Big Air is processing purchase order number 20561. Big Air's process has the URI of

```
http://enoch.seattle.bigair.xom/erp/purchase/20561
```

Midwest Aircraft Supply's inventory level service is located at

```
http://asher.midwest-air.xom/engine/checkInventory
```

In the second scenario we will trace the messages from Midwest's perspective. We will show how a simple engine can enable Dave in inventory to respond to Big Air's service request.

In the third scenario, we will show how the inventory checking process can be enhanced with further automation within Midwest without any disruption occurring to Big Air.

These scenarios follow a progression of detail. The first scenario addresses only the protocol itself. The second scenario addresses issues of actually implementing the protocol. The third scenario looks at strategic issues of how the protocol enables trans-enterprise integration.

2.1 Factory GetProperties

Big Air checks the properties of Midwest's checkInventory service with a GetProperties request.

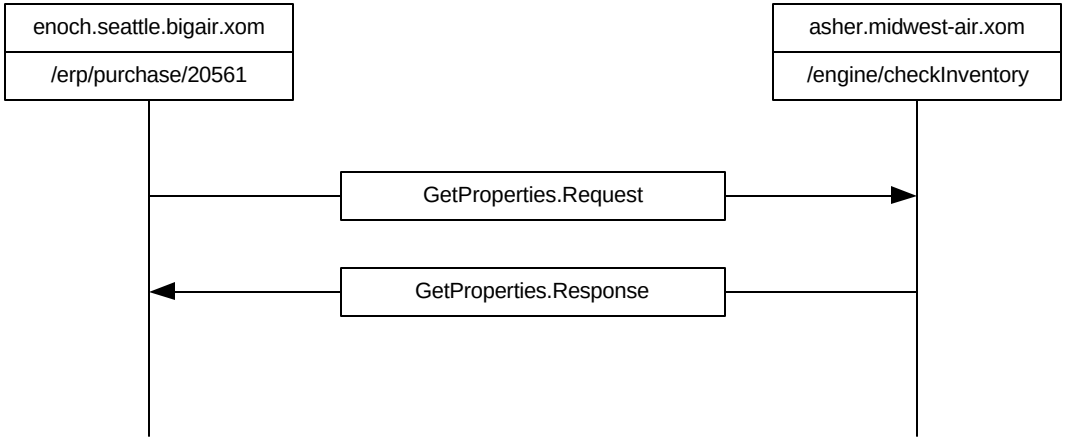

Figure 3 Big Air checks the properties of Midwest's checkInventory service factory

The Big Air server sends the following request.

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <aws:Request xmlns:aws="http://www.awsp.info/1.0/">
      <aws:SenderKey>http://enoch.seattle.bigair.xom/erp/purchase/20561</aws:SenderKey>
      <aws:ReceiverKey>http://asher.midwest-air.xom/engine/checkInventory</aws:ReceiverKey>
    </aws:Request>
  </env:Header>
  <env:Body>
    <aws:GetProperties.Request/>
  </env:Body>
</env:Envelope>
```

Listing 1 Factory resource GetProperties request

The Midwest server replies with the following message.

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <aws:Response xmlns:AWSP="http://www.awsp.info/1.0/">
      <aws:SenderKey>http://asher.midwest-air.xom/engine/checkInventory</aws:ReceiverKey>
      <aws:ReceiverKey> http://enoch.seattle.bigair.xom/erp/purchase/20561<aws:ReceiverKey>
    </aws:Response>
  </env:Header>
  <env:Body>
    <aws:GetProperties.Response>
      <aws:PortType>Factory</aws:PortType>
      <aws:Name>checkInventory</aws:Name>
      <aws:Key>http://asher.midwest-air.xom/engine/checkInventory</aws:Key>
      <aws:Subject>Check inventory level</aws:Subject>
      <aws:Description>
        This process/service provides inventory levels by part number (SKU). If
        you do not know the correct SKU, please see http://www.midwest-air.xom/catalog/
        or call Dave Stephenson at 630.555.7263.
        As an option, you can also submit the product name for clarification and
        the quantity you. The process will return the SKU, official product name
        and the quantity we currently have in inventory.
      </aws:Description>
      <aws:ValidStates>
        <aws:State>open.notRunning </aws:State>
        <aws:State>open.notRunning.suspended</aws:State>
        <aws:State>open.running</aws:State>
        <aws:State>closed.completed</aws:State>
        <aws:State>closed.abnormalCompleted</aws:State>
        <aws:State>closed.abnormalCompleted.terminated</aws:State>
        <aws:State>closed.abnormalCompleted.aborted</aws:State>
      </aws:ValidStates>
      <aws:ContextDataSchema href="http://www.midwest-air.xom/schema/productlevel.xsd"/>
      <aws:ResultDataSchema href="http://www.midwest-air.xom/schema/productlevel.xsd"/>
    </aws:GetProperties.Response>
  </env:Body>
</env:Envelope>
```

Listing 2 Factory resource GetProperties response

2.2 Context data

The context data and the result data point to an outside schema stored on Midwest's website. The product information this service creates looks like the following:

```
<?xml version="1.0"?>
<product xmlns="http://www.midwest-air.xom/schema/productlevel.xsd">
   <partNum>872-NF</partNum>
   <name>Bulkhead socket plugs<name>
   <quantity>12</quantity>
</product>
```

Listing 3 Sample of resulting process data

The schema for this data structure is shown in Listing 4.

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.midwest-air.xom/schema/productlevel.xsd"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.midwest-air.xom/schema/productlevel.xsd">
 <annotation>
   <documentation xml:lang="en">
     Inventory levels for Midwest.
     Copyright (c) 2002 Midwest Aircraft Supply. All rights reserved.
   </documentation>
 </annotation>
 <xsd:complexType name="product">
   <xsd:sequence>
     <xsd:element name="prodNum" type="SKU"/>
     <xsd:element name="prodName" type="xsd:string" minOccurs="0"/>
     <xsd:element name="quantity" minOccurs="0">
       <xsd:simpleType>
         <xsd:restriction base="xsd:positiveInteger">
           <xsd:maxExclusive value="100"/>
         </xsd:restriction>
       </xsd:simpleType>
     </xsd:element>
   </xsd:sequence>
 </xsd:complexType>
 <!-- Stock Keeping Unit, a code for identifying products -->
```

```
 <xsd:simpleType name="SKU">
   <xsd:restriction base="xsd:string">
     <xsd:pattern value="\d{3}-[A-Z]{2}"/>
   </xsd:restriction>
 </xsd:simpleType>
</xsd:schema>
```

Listing 4 The schema for the process data and process results

It is possible to have the data schema stored directly in the factory properties. Rather than have a link to the XML schema file as shown before, the GetProperties results document could have contained a ContextDataSchema element that looked like Listing 5. Storing the schema in a separate file increases reuse of the schema and makes management easier.

```
...
<aws:ContextDataSchema>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="product">
      <xsd:sequence>
        <xsd:element name="prodNum" type="SKU"/>
        <xsd:element name="prodName" type="xsd:string" minOccurs="0"/>
        <xsd:element name="quantity" minOccurs="0">
          <xsd:simpleType>
            <xsd:restriction base="xsd:positiveInteger">
              <xsd:maxExclusive value="100"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="SKU">
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{3}-[A-Z]{2}"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
</aws:ContextDataSchema>
...
```

Listing 5 Schema embedded directly into the factory properties document

2.3 Factory CreateInstance

In order to check inventory levels, the Big Air server needs to ask the Midwest server to create a dedicated instance of the inventory level service. A resource on the Big Air server must register itself as the observer of the service instance the Midwest server is creating.

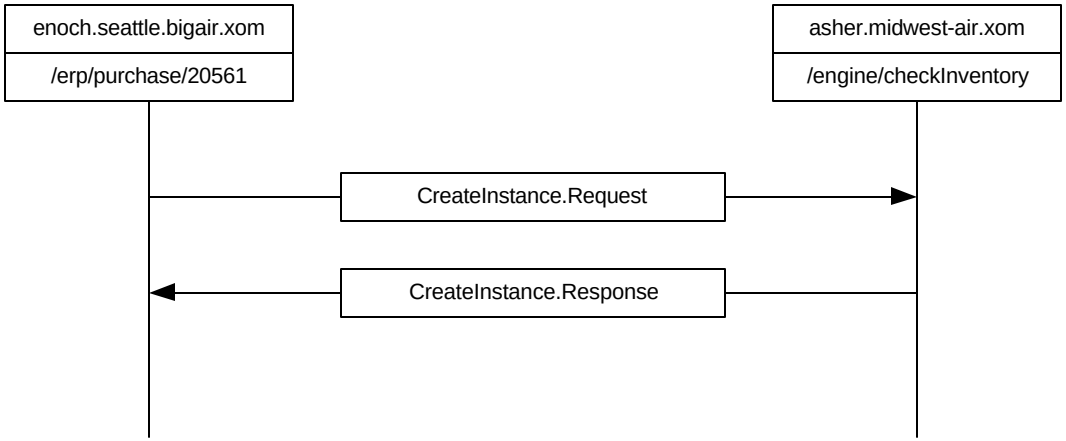

Figure 4 Big Air creates an instance of Midwest's checkInventory service

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
<env:Header>
  <aws:Request xmlns:AWSP="http://www.awsp.info/1.0/">
    <aws:ReceiverKey>http://asher.midwest-air.xom/engine/checkInventory</aws:ReceiverKey>
  </aws:Request>
</env:Header>
<env:Body>
  <aws:CreateInstance.Request>
    <aws:StartImmediately>Yes</aws:StartImmediately>
    <aws:ObserverKey>http://enoch.seattle.bigair.xom/erp/purchase/20561
    </aws:ObserverKey>
    <aws:Name>20561.1</aws:Name>
    <aws:Subject>838-800 bulkhead pass through connectors</aws:Subject>
    <aws:Description>New bulkhead and air frame length requires new connectors</aws:Description>
    <aws:ContextData>
```

```
      <us:product xmlns:us="http://www.midwest-air.xom/schema/productlevel.xsd">
        <us:prodNum>872-NF</prodNum>
        <us:prodName>Bulkhead socket plugs<us:prodName>
        <us:quantity>12</us:quantity>
      </us:product>
    </aws:ContextData>
  </aws:CreateInstance.Request>
</env:Body>
</env:Envelope>
```

Listing 6 Factory resource CreateInstance request

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
<env:Header>
  <aws:Response>
    <aws:SenderKey>http://asher.midwest-air.xom/engine/checkInventory</aws:SenderKey>
  </aws:Response>
</env:Header>
<env:Body>
  <aws:CreateInstance.Response>
    <aws:Key>http://asher.midwest-air.xom/engine/checkInventory/2002-01-24T09:45:51Z</aws:Key>
  </aws:CreateInstance.Response>
</env:Body>
</env:Envelope>
```

Listing 7 Factory resource CreateInstance response

Big Air requested that the service instance be named 20561.1. The Midwest server ignored this request and named the service instance using a date-time stamp. The URI of the new service instance that Big Air will be using is

```
http://asher.midwest-air.xom/engine/checkInventory/2002-01-24T09:45:51Z
```

Note that this instance URI is an extension of the factory URI. This may be a good practice, but it is not mandatory. The URI need only be unique and not repeated by other service instances.

2.4 Factory ListInstances

An asynchronous web service can have many instances. To know all of the instances, you send a ListInstances request to the factory. The list should now include the instance we just created.

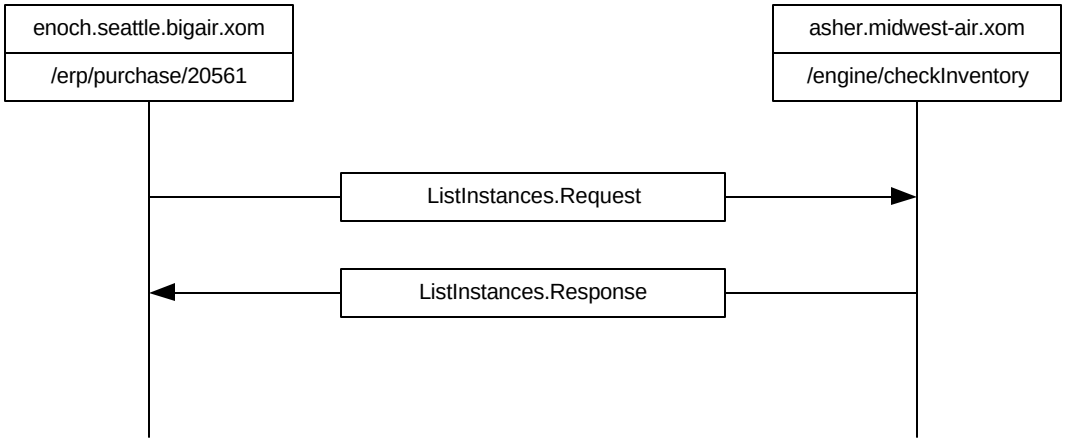

Figure 5 ListInstances

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
<env:Header>
  <aws:Request xmlns:AWSP="http://www.awsp.info/1.0/">
    <aws:ReceiverKey>http://asher.midwest-air.xom/engine/checkInventory</aws:ReceiverKey>
  </aws:Request>
</env:Header>
<env:Body>
  <aws:ListInstances.Request/>
</env:Body>
</env:Envelope>
```

Listing 8 Factory resource ListInstances request

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
<env:Header>
  <aws:Response xmlns:AWSP="http://www.awsp.info/1.0/">
    <aws:SenderKey>http://asher.midwest-air.xom/engine/checkInventory</aws:SenderKey>
  </aws:Response>
</env:Header>
<env:Body>
```

```
  <aws:ListInstances.Response>
    <aws:Instance>
      <aws:Key>http://asher.midwest-air.xom/engine/checkInventory/2002-01-24T09:45:51Z</aws:Key>
      <aws:Subject>838-800 bulkhead pass through connectors</aws:Subject>
    </aws:Instance>
    <aws:Instance>
      <aws:Key>http://asher.midwest-air.xom/engine/checkInventory/2002-01-23T12:05:17Z</aws:Key>
      <aws:Subject>New Century Fighter cokpit lighting</aws:Subject>
    </aws:Instance>
  </aws:ListInstances.Response>
</env:Body>
</env:Envelope>
```

Listing 9 Factory resource ListInstances response

We see the instance we just created along with another instance created the day before with the subject “New Century Fighter cockpit lighting”. With a business application, this list could be quite large. It could also be quite sensitive. For these reasons, the list of instances is not a part of the GetProperties response.

2.5 Instance GetProperties

Now that the service instance is created, we can send requests to it. The following is a basic GetProperties request to the service instance we created.

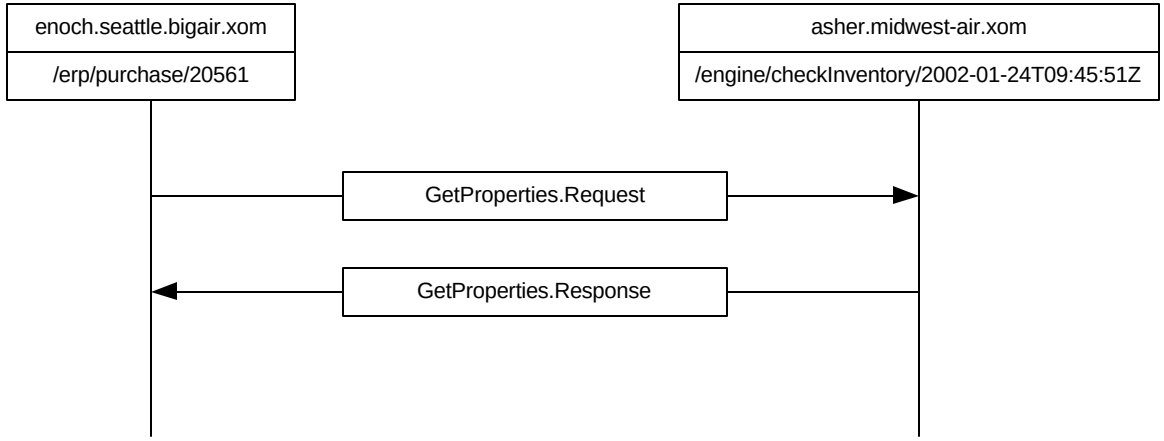

Figure 6 Instance resource GetProperties exchange

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <aws:Request xmlns:AWSP="http://www.awsp.info/1.0/">
      <aws:ReceiverKey>http://asher.midwest-air.xom/engine/checkInventory/2002-01-24T09:45:51Z</aws:ReceiverKey>
    </aws:Request>
  </env:Header>
  <env:Body>
    <aws:GetProperties.Request/>
  <env:Body>
</env:Envelope>
```

Listing 10 Instance GetProperties request

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <aws:Response xmlns:AWSP="http://www.awsp.info/1.0/">
      <aws:SenderKey>http://asher.midwest-air.xom/engine/checkInventory/2002-01-24T09:45:51Z</aws:SenderKey>
    </aws:Response>
  </env:Header>
  <env:Body>
    <aws:GetProperties.Response>
      <aws:PortType>Instance</aws:PortType>
      <aws:PortType>Observer</aws:PortType>
      <aws:Name>2002-01-24T09:45:51Z</aws:Name>
      <aws:Key>http://asher.midwest-air.xom/engine/checkInventory/2002-01-24T09:45:51Z</aws:Key>
      <aws:Subject>838-800 bulkhead pass through connectors</aws:Subject>
      <aws:Description>New bulkhead and air frame length requires new connectors</aws:Description>
      <aws:State>open.running</aws:State>
      <aws:FactoryKey>http://asher.midwest-air.xom/engine/checkInventory</aws:FactoryKey>
      <aws:Observers>
        <aws:ObserverKey>http://enoch.seattle.bigair.xom/erp/purchase/20561</aws:ObserverKey>
      </aws:Observers>
      <aws:ContextData>
        <us:product xmlns:us="http://www.midwest-air.xom/schema/productlevel.xsd">
          <us:prodNum>872-NF</prodNum>
          <us:prodName>Bulkhead socket plugs<us:prodName>
          <us:quantity>12</us:quantity>
```

```
        </us:product>
      </aws:ContextData>
      <aws:ResultData/>
    </aws:GetProperties.Response>
  <env:Body>
</env:Envelope>
```

Listing 11 Instance resource GetProperties response

Note a couple of critical properties. The Big Air resource is listed as an observer to the process instance. The state of the process instance is “open.running” because the CreateInstance request said start immediately. The context data contains the information that Big Air provided, but the result data is currently empty.

2.6 Observer Complete

Eventually, Dave in inventory will enter the inventory levels. When he does, the Midwest engine needs to send the results back to Big Air. To do this, the service instance sends a Complete request to its observer.

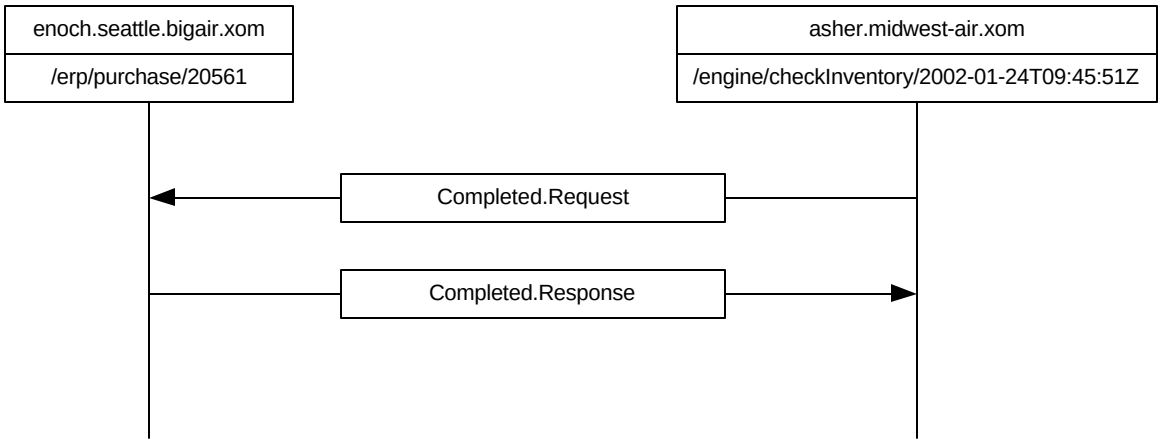

Figure 7 Observer resource Completed exchange

The service instance initiates the message to the observer. In this message we see the result data.

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <aws:Request xmlns:AWSP="http://www.awsp.info/1.0/">
      <aws:ReceiverKey>http://enoch.seattle.bigair.xom/erp/purchase/20561</aws:ReceiverKey>
    </aws:Request>
  </env:Header>
  <env:Body>
    <aws:Completed.Request>
      <aws:InstanceKey>http://asher.midwest-air.xom/engine/checkInventory/2002-01-24T09:45:51Z</aws:InstanceKey>
      <aws:ResultData>
        <us:product xmlns:us="http://www.midwest-air.xom/schema/productlevel.xsd">
          <us:prodNum>872-NF</prodNum>
          <us:prodName>Bulkhead socket plugs<us:prodName>
          <us:quantity>317</us:quantity>
        </us:product>
      </aws:ResultData>
    </aws:Completed.Request>
  </env:Body>
</env:Envelope>
```

Listing 12 Observer resource Completed request

The result data says there are 317 of the bulkhead socket plugs available, more than the 12 that Big Air inquired about. The Big Air observer resource acknowledges the receipt of the Completed request.

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <aws:Response xmlns:AWSP="http://www.awsp.info/1.0/">
      <aws:SenderKey>http://enoch.seattle.bigair.xom/erp/purchase/20561</aws:SenderKey>
    </aws:Response>
  </env:Header>
  <env:Body>
    <aws:Completed.Response/>
  </env:Body>
</env:Envelope>
```

Listing 13 Observer resource Completed response

The state of this web service instance is now

```
<aws:State>closed.complete</aws:State>
```

2.7 Summary of scenario 1

The following diagram summarizes the message exchanges we have covered in this scenario.

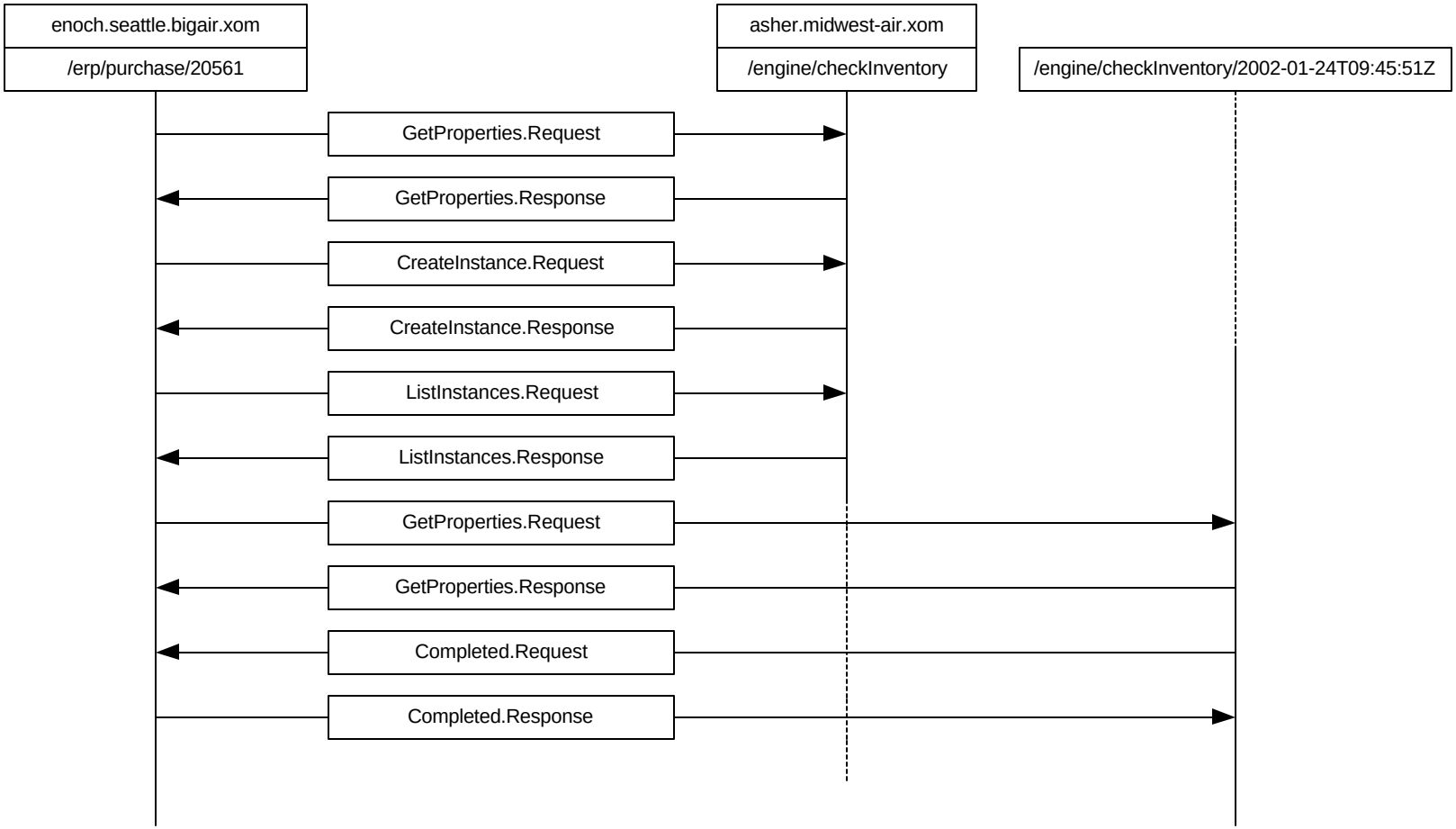

Figure 8 Scenario 1 summarized

Note that the GetProperties and ListInstances exchanges are really only included for demonstration purposes. The most basic implementation of AWSP requires only a CreateInstance method to a factory followed by a Completed method back to the observer.

3 Inventory example expanded

We will now take a look at how Dave knew to fill out the inventory level form. We also look at how the AWSP engine processed the form data to respond to the AWSP request.

How does Dave found out that he has an inventory level request? There is a special type of web service factory called a *worklist* that represents a person. When worklist service factory is instantiated it is called a *work item*. The work item informs the person what needs to be done. In our case, it is completing a web form with the inventory level information. This should usually be the case. Web forms are the typical way computers get information from people anyway.

In this scenario we will discuss implementation issues to show the practicality of the protocol. Developers are free to build implementations of the protocol however they wish so long as it exchanges the AWSP messages properly. The purpose of a protocol is, after all, to free developers from the concerns of the specifics of other developers' implementations.

3.1 WorkList CreateInstance

The web service instance created by Big Air takes on the role of an observer and asks the work list process definition to create a work item process instance. You may not have noticed, but in Listing 11 Instance resource GetProperties response the service instance listed that it implemented the observer interface as well.

```
...
<aws:GetProperties.Response>
  <aws:PortType>Instance</aws:PortType>
  <aws:PortType>Observer</aws:PortType>
...
```

Listing 14 Excerpt from the Instance GetProperties response

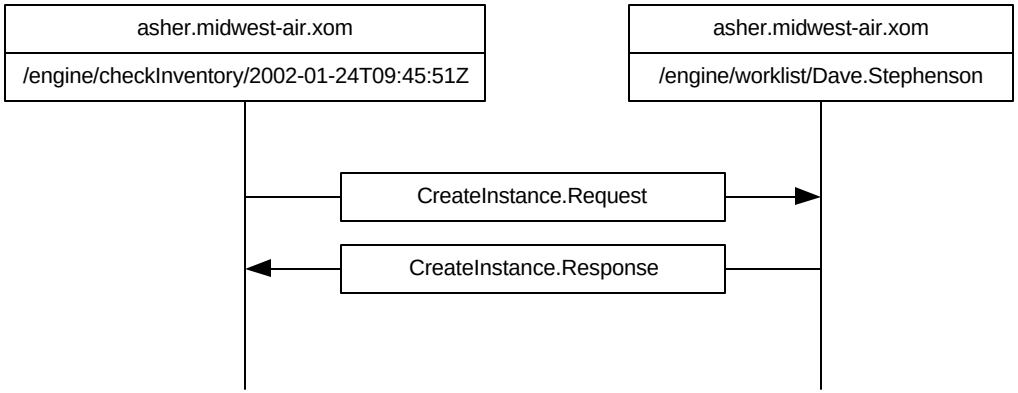

Figure 9 Worklist CreateInstance method

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <aws:Request xmlns:AWSP="http://www.awsp.info/1.0/">
      <aws:ReceiverKey>http://asher.midwest-air.xom/engine/worklist/Dave.Stephenson</aws:ReceiverKey>
    </aws:Request>
  </env:Header>
  <env:Body>
    <aws:CreateInstance.Request>
      <aws:StartImmediately>yes</aws:StartImmediately>
      <aws:Observer>http://asher.midwest-air.xom/engine/checkInventory/2002-01-24T09:45:51Z</aws:Observer>
      <aws:Subject>Inventory check: 872-NF</aws:Subject>
      <aws:ContextData>
        <form>http://asher.midwest-air.xom/forms/inventoryLevel?PID=2002-10-24T09:45:51Z</form>
      </aws:ContextData>
    </aws:CreateInstance.Request>
  </env:Body>
</env:Envelope>
```

Listing 15 Worklist resource CreateInstance request

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
```

```
    <aws:Response>
      <aws:SenderKey>http://asher.midwest-air.xom/engine/worklist/Dave.Stephenson</aws:SenderKey>
    </aws:Response>
  </env:Header>
  <env:Body>
    <aws:CreateInstance.Response>
      <aws:InstanceKey>http://asher.midwest-air.xom/engine/worklist/Dave.Stephenson/2002-01-24T09:46:01Z
      </aws:InstanceKey>
    </aws:CreateInstance.Response>
  </env:Body>
</env:Envelope>
```

Listing 16 Worklist resource CreateInstance response

Notice that the context data for the work item service instance is an element named “form” with a URL as its content.

3.2 Worklist client

Now the issue is one of Dave’s work items. If we wanted to see all of Dave’s work items, we would send a ListInstances request to his worklist since a worklist is a web service factory. Conceivably, Dave could use some fancy third-party worklist management client software to handle his worklist items, much the same way we use a fancy third-party client to handle our email. That fancy client could interface to the worklist using AWSP just as it uses POP or IMAP to interface with the e-mail. However, we want to focus on the simplicity of implementing AWSP, so we will assume that the AWSP engine that Midwest is using has a simple web-based worklist interface.

For our scenario, the engine provides a GET interface to the work list process definitions that generates an HTML showing the work items.

```
<html>
<head><title>Dave Stephenson worklist</title></head>
<body>
<h1>Dave Stephenson</h1>
<h2>Worklist</h2>
<ol>
  ...
  <li>
    <a href="http://asher.midwest-air.xom/forms/inventoryLevel?observer=/checkInventory/2002-10-
24T09:45:51Z&workitem=/Dave.Stephenson/2002-01-24T09:46:01Z&prodNum=872-NF&prodName=Bulkhead+socket+plugs&quantity=12">
      Inventory check: 872-NF
    </a>
  </li>
```

```
 ...
</ol>
</body>
</html>
```

Listing 17 HTML version of the work list

A possible rendering of this HTML is shown in Figure 10. The HTML display has a hyperlink to the form. When Dave clicks on that link, the form will appear. That's some crazy URL for the form, right? That is a GET statement that generates the form with the proper data. We will cover that further in a minute.

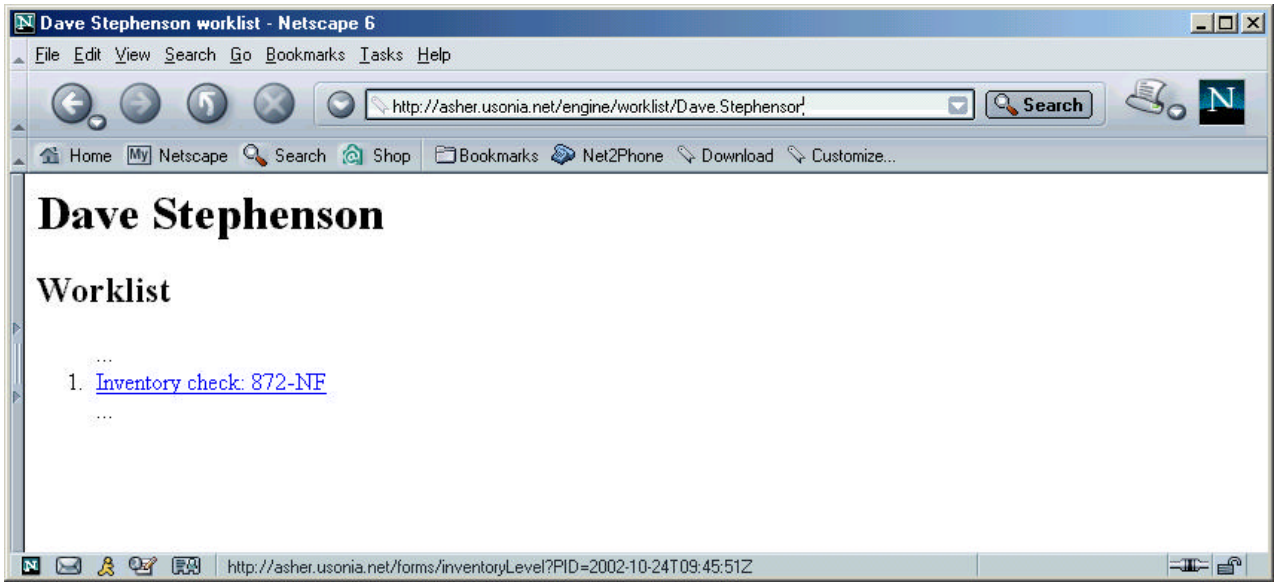

Figure 10 Web interface to the work list is linked to the form

Do we have to assume that Dave has to constantly check his work list on a web browser? Not really. It is easy to conceive that the engine allows email notice when a new work item arrives. The engine could even send a pager notice.

3.3 Forms

We now need a web form. A simple one will do.

```
<html>
<head><title>Check Inventory</title></head>
<body>
<form>
<h1>Check Inventory</h1>
<form method="POST" action="/engine/forms/inventoryLevelResponse">
<input type="hidden" name="observer" value="/checkInventory/2002-10-24T09:45:51Z"/>
<input type="hidden" name="workitem" value="/Dave.Stephenson/2002-01-24T09:46:01"/>
<input type="hidden" name="prodNum" value="872-NF"/>
<table>
<tr><td>Part number</td><td>872-NF</td></tr>
<tr><td>Name</td><td><input type="text" name="prodName" value="Bulkhead socket plugs"/></td></tr>
<tr><td>Requested quantity</td><td>12</td></tr>
<tr><td>Quantity available</td><td><input type="text" name="quantity"/></td></tr>
</table>
<input type="submit">Submit</input>
</form>
</body>
</html>
```

Figure 11

This form has to be created because it contains information specific to the service instance. There are two options for creating this form. The first option is that a servlet or CGI script generates the form based on the parameters passed to it. This is the traditional method. It combines logic and presentation, that is, the look-and-feel is grouped in with the code, but most web developers are use to that.

The second approach is that the service instance creates an XML document representing the form instance, possibly even using XForms (http://www.w3.org/MarkUp/Forms/). A stylesheet is applied to present the form as HTML. With this approach, we can separate the logic from the presentation. We can also get rid of the complex GET form URL used in the work list above. Such as XML document might look like Listing 18. The only drawback is that it requires server-side XSLT processor such as Apache Cocoon (http://xml.apache.org) or Trans-enterprise Common Engine (http://www.trans-enterprise.com/products/).

```
<?xml version="1.0"?>
<form>
<xforms:model xmlns:xforms="http://www.w3.org/2001/12/xforms">
```

```
  <xforms:instance>
    <us:observer>/checkInventory/2002-10-24T09:45:51Z</us:observer>
    <us:workitem>/Dave.Stephenson/2002-01-24T09:46:01Z</us:workitem>
    <us:product xmlns:us="http://www.midwest-air.xom/schema/productlevel.xsd">
      <us:prodNum>872-NF</prodNum>
      <us:prodName>Bulkhead socket plugs<us:prodName>
      <us:quantity>12</us:quantity>
      </us:product>
    </xforms:instance>
  <xforms:submitInfo action="/engine/forms/inventoryLevelResponse" method="POST"/>
  </xforms:model>
<xforms:input ref="us:product/us:prodName">
  <xforms:caption>Name</xforms:caption>
  </xforms:input>
<xforms:input ref="us:product/us:quantity">
  <xforms:caption>Quantity available</xforms:caption>
  </xforms:input>
<xforms:submit>
  <xforms:caption>Submit</xforms:caption>
  </xforms:submit>
</form>
```

Listing 18 Possible XML representation of the form

Regardless of how the form is generated from the service instance, the data comes back from the HTML form in a format that must be translated into XML result data. There are some cleaver ways to have the browser create an XML document as its form response, but at the time of writing they are still browser-specific in their implementation. It is up to the form's action, be it a servlet or CGI script, to generate the Complete.Request message that includes the ResultData element created from the form posting. That posting looks like Listing 19 below. For clarity, we have not encoded the parameters.

```
POST /engine/forms/inventoryLevelResponse HTTP/1.1
Host: asher.midwest-air.xom
Content-length: NNNN

observer=/checkInventory/2002-10-24T09:45:51Z&workitem=/Dave.Stephenson/2002-01-24T09:46:01Z&prodNum=872-NF&prodName=
Bulkhead+socket+plugs&quantity=317
```

Listing 19 HTTP POST message from form

3.4 Observer Completed

A servlet (or CGI script) must create the Completed request to the observer. The servlet must identify itself as the work item.

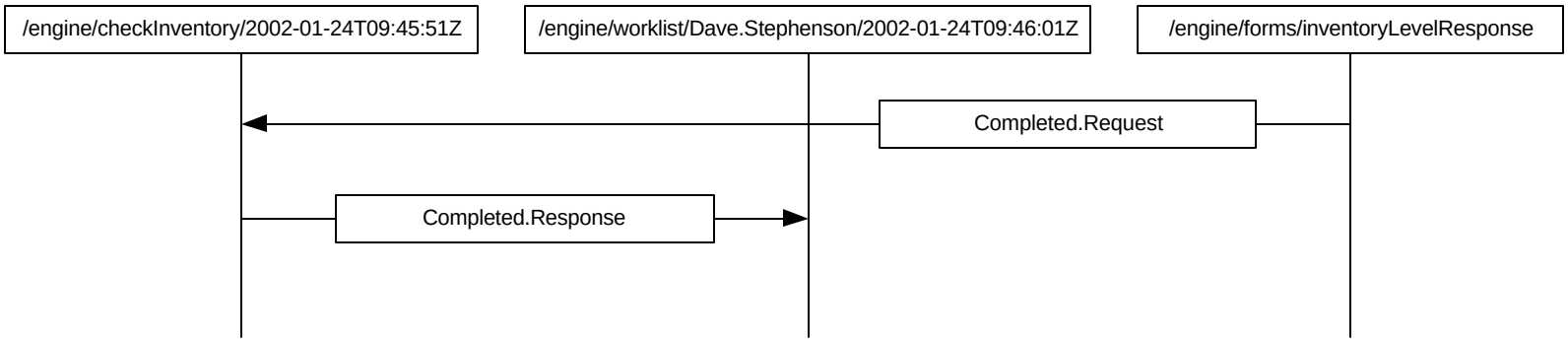

Figure 12 Generating the Completed request for the work item

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <aws:Request xmlns:AWSP="http://www.awsp.info/1.0/">
      <aws:ReceiverKey>http://asher.midwest-air.xom/engine/checkInventory/2002-01-24T09:45:51Z</aws:ReceiverKey>
    </aws:Request>
  </env:Header>
  <env:Body>
    <aws:Completed.Request>
      <aws:InstanceKey>http://asher.midwest-air.xom/worklist/Dave.Stephenson/2002-01-24T09:46:01Z</aws:InstanceKey>
      <aws:ResultData>
        <us:product xmlns:us="http://www.midwest-air.xom/schema/productlevel.xsd">
          <us:prodNum>872-NF</prodNum>
          <us:prodName>Bulkhead socket plugs<us:prodName>
          <us:quantity>317</us:quantity>
        </us:product>
      </aws:ResultData>
    </aws:Completed.Request>
  </env:Body>
</env:Envelope>
```

Listing 20 Observer resource Complete request

The observer acknowledges the receipt of the Completed request.

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <aws:Response xmlns:AWSP="http://www.awsp.info/1.0/">
      <aws:SenderKey>http://asher.midwest-air.xom/engine/checkInventory/2002-01-24T09:45:51Z</aws:SenderKey>
    </aws:Response>
  </env:Header>
  <env:Body>
    <aws:Completed.Response/>
  </env:Body>
</env:Envelope>
```

Listing 21 Observer resource Completed response

Note that the observer will POST this message to the URL of the work item, not the URL of the form processing servlet that created it. Otherwise, our servlet's doPost method would have to handle (that is, ignore) the Completed response. Listing 22 provides a very simple Java servlet that generates the form with its GET method and creates the AWSP Completed request with its POST method. This is a grunt force approach in its execution, granted. It should serve its purpose in showing how practical it is to create an AWSP form interface. You may use the AWSP software development kit (SDK) to make such tasks a bit easier and more elegant to implement.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.net.*;

/**
 * A simple servlet to handle the check inventory form.
 * The GET method creates the form. The POST method processes the form and
 * generates the Observer Completed request.
 */
public class inventoryLevel extends HttpServlet {

/**
 * Constructs the form from the GET parameters
 * observer  the URI of the AWSP observer. This form only handles one.
 * workitem  the URI of the workitem
```

```
 * prodNum   product SKU
 * prodName  product name
 * quantity  quantity requested
 */
public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws IOException, ServletException {
  PrintWriter out = res.getWriter();
  out.println("<html><head><title>Check Inventory</title></head>");
  out.println("<body><form><h1>Check Inventory</h1>");
  out.println("<form method=\"POST\" action=\"/engine/forms/inventoryLevel\">");
  out.println("<input type=\"hidden\" name=\"observer\" value=\"");
  out.println(req.getParameter("observer"));
  out.println("\"/><input type=\"hidden\" name=\"workitem\" value=\"");
  out.println(req.getParameter("workitem"));
  out.println("\"/><input type=\"hidden\" name=\"prodNum\" value=\"");
  String prodNum = req.getParameter("prodNum"); //we need it twice
  out.println(prodNum);
  out.println("\"/><table><tr><td>Part number</td><td>");
  out.println(prodNum);
  out.println("</td></tr><tr><td>Name</td><td>");
  out.println("<input type=\"text\" name=\"prodName\" value=\"");
  out.println(req.getParameter("prodName"));
  out.println("\"/></td></tr><tr><td>Requested quantity</td><td>");
  out.println(req.getParameter("quantity"));
  out.println("</td></tr>");
  out.println("<tr><td>Quantity available</td><td>");
  out.println("<input type=\"text\" name=\"quantity\"/>");
  out.println("</td></tr></table><input type=\"submit\">Submit</input></form>");
  out.println("</body></html>");
  }

/**
 * Translate the POST into a AWSP Complete message to the Observer
 * observer  the URI of the AWSP observer. This form only handles one.
 * workitem  the URI of the workitem
 * prodNum   product SKU
 * prodName  product name
 * quantity  quantity available
 */
 */
public void doPost(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
```

```
  PrintWriter resout = res.getWriter();
  try {
    //create Observer Complete request
    String observer = req.getParameter("observer");
    URL url = new URL(observer);
    URLConnection con = url.openConnection();
    con.setDoOutput(true);
    PrintWriter out = new PrintWriter(con.getOutputStream());
    out.println("<?xml version=\"1.0\"?>");
    out.println("<env:Envelope xmlns:env=\"http://www.w3.org/2001/12/soap-envelope\">");
    out.println("<env:Header>  <aws:Request xmlns:AWSP=\"http://www.awsp.info/1.0/\">");
    out.println("<aws:ReceiverKey>");
    out.println(observer);
    out.println("</aws:ReceiverKey></aws:Request></env:Header>");
    out.println("<env:Body><aws:Completed.Request>");
    out.println("<aws:InstanceKey>");
    out.println(req.getParameter("workitem"));
    out.println("</aws:InstanceKey><aws:ResultData>");
    out.println("<us:product xmlns:us=\"http://www.midwest-air.xom/schema/productlevel.xsd\">");
    out.println("<us:prodNum>");
    out.println(req.getParameter("prodNum"));
    out.println("</prodNum><us:prodName>");
    out.println(req.getParameter("prodName"));
    out.println("</us:prodName>");
    out.println("<us:quantity>");
    out.println(req.getParameter("quantity"));
    out.println("</us:quantity></us:product>");
    out.println("</aws:ResultData></aws:Completed.Request>");
    out.println("</env:Body></env:Envelope>");
    out.close();
    //send an acknowledgement back to the browser
    resout.println("<html><head><title>Check Inventory</title></head>");
    resout.println("<body><form><h1>Check Inventory</h1>");
    resout.println("<p>Work item completed.</p></body></html>");
    }
  catch (Exception e) {
    //send error back to the browser
    resout.println("<html><head><title>Check Inventory</title></head>");
    resout.println("<body><form><h1>Check Inventory</h1>");
    resout.println("<p>There was a error.</p><pre>");
    resout.println(e.getMessage());
    resout.println("</pre></body></html>");
```

```
    }
  }

}
```

Listing 22 A servlet that handles the check inventory form

3.5 Instance Terminate

Depending on the implementation, the observer may need to send a Terminate request to the work item service instance in order to remove it from the worklist. It is conceivable that some implementations might automatically remove the work item when the Completed response is received.

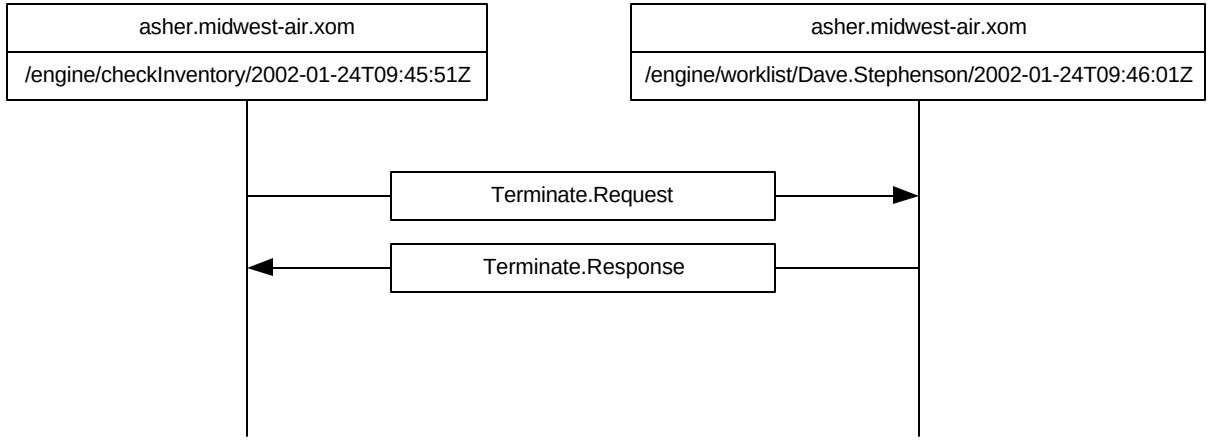

Figure 13 Terminate exchange

3.6 Scenario summary

The following diagrams summarize the interactions of the scenario we have just reviewed.

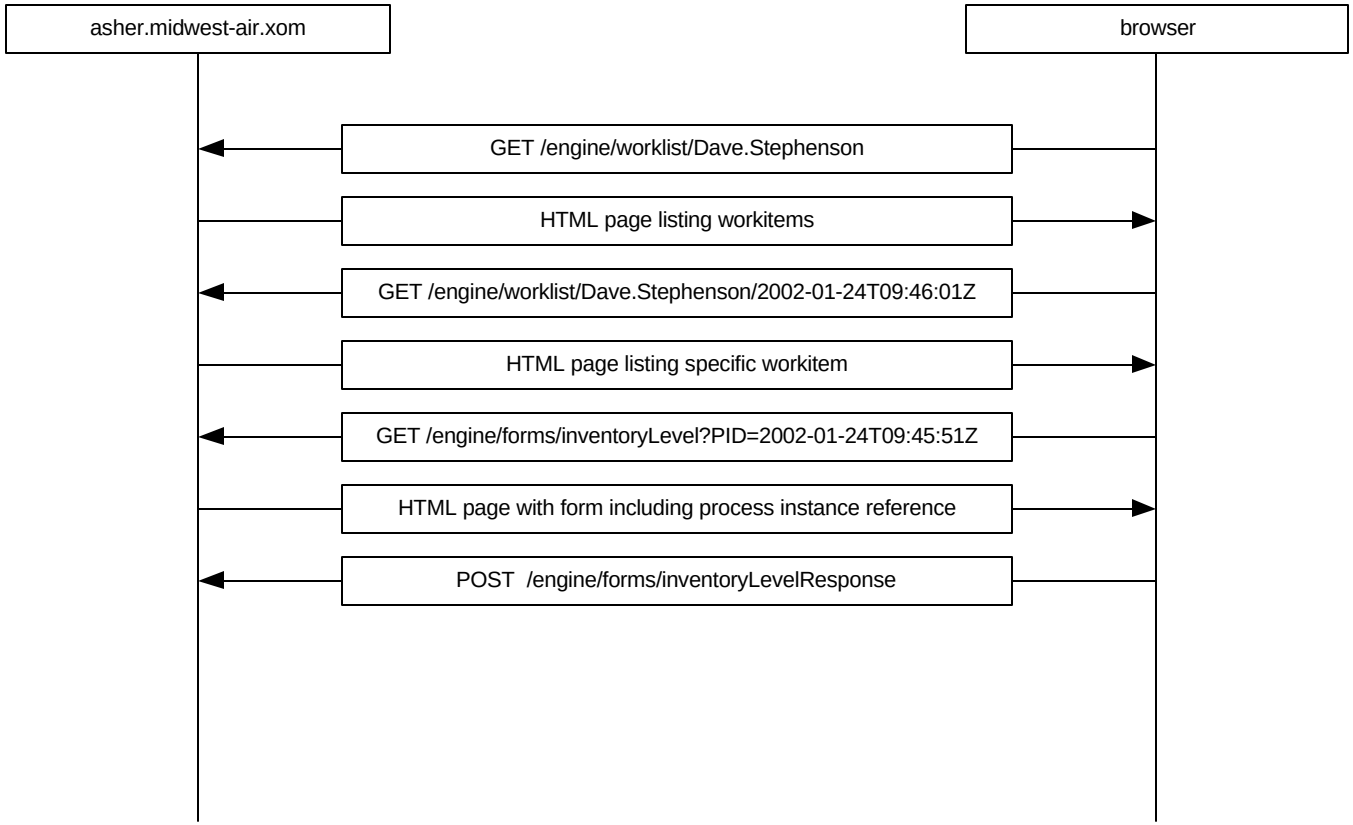

Figure 14 Interactions with forms

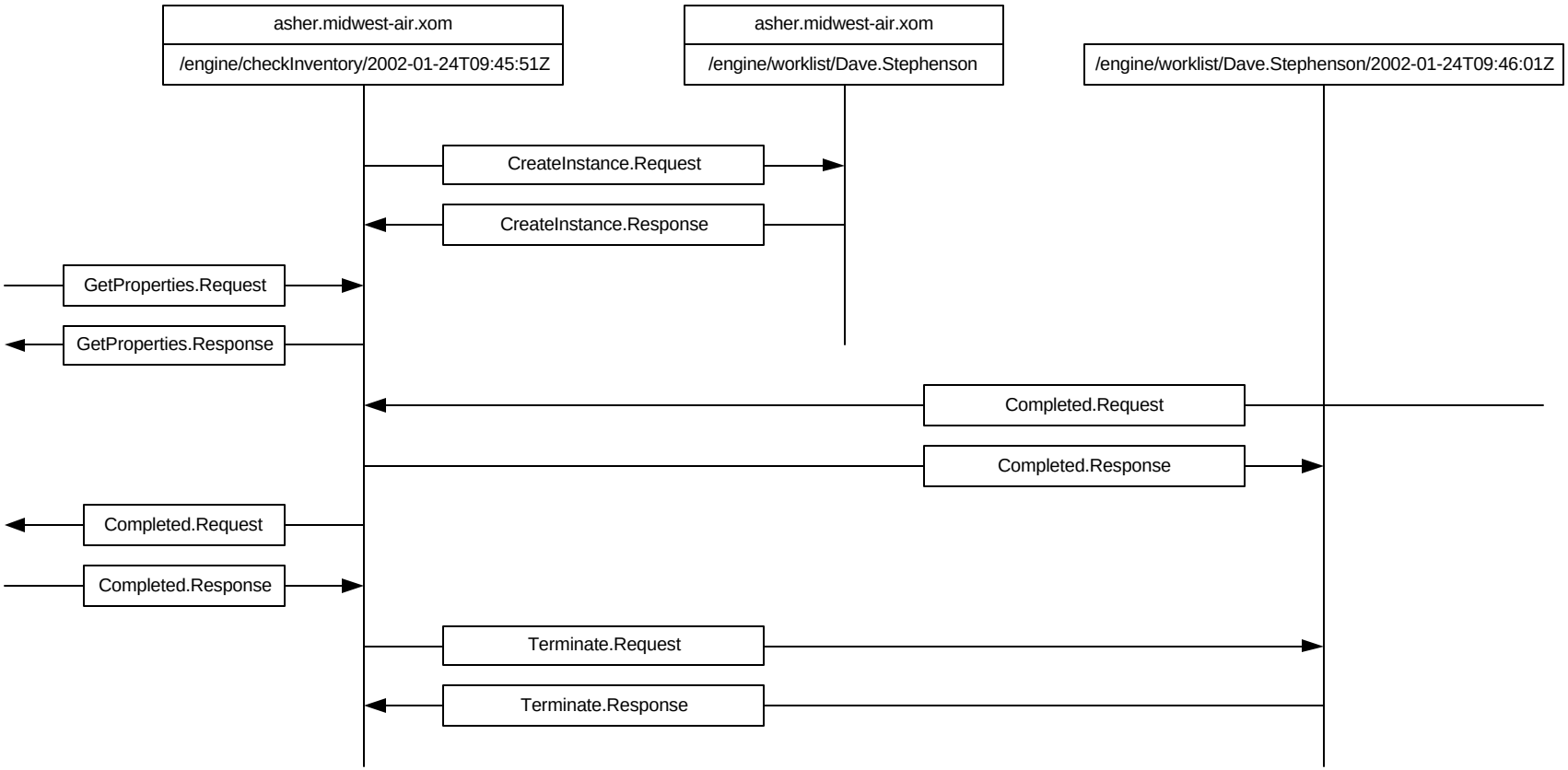

Figure 15 Scenario 2 summary