Summary Report on the Workflow Management Coalition Asynchronous Service Access Protocol Interoperability Test

Keith D. Swenson, Fujitsu Software Corporation
Justin Brunt, TIBCO

June 2004

1 Synopsis

On June 23 at the BrainStorm BPM Conference in San Francisco a worldwide interoperability demonstration and test were successfully concluded. Five different implementations of the protocol were able to demonstrate the exchange of data across the Internet following the OASIS ASAP Committee Draft specification. This is the culmination of years of developing the proposed standard and nine months of planning for the demonstration. This document contains the details on who participated, and what specific interactions were supported, as well as some insight into the process of getting the demo running.

The successful conclusion of the demo means that ASAP has been proven to be an effective way to connect systems easily without having to use programming techniques for each connection. It shows that ASAP is viable now, without having to wait for other standards to be refined. It also shows that it is not difficult to implement the ASAP protocol. While there will always be newer technologies to jump to, the participants in the demonstration, and the members of the OASIS technical committee, hope that systems that need to communicate to other systems asynchronously will include this important basic protocol as an option.
2 Background

The history of the development of ASAP will not be included here, as much of this can be found in the specification and related documents itself. It is sufficient to know that WfMC members have been working for many years to find a standard way to link workflow and BPM engines together. In 1996 a demonstration was made of the WfMC “interface 4” which involved messages being sent between systems using SMTP email messages. This was before the invention of XML. Subsequently, the WfMC worked with the OMG to define a standard CORBA interface to workflow services. After the advent of XML, several WfMC members got together to define a protocol based on the OMG interface, but defined by sending XML across HTTP connections. This was proposed in 1998 to the IETF as the Simple Workflow Access Protocol (SWAP). Further refinement of this, called Wf-XML was implemented by a number of WfMC members, and was demonstrable. But SWAP and Wf-XML were not based on the SOAP message structure, and so could not benefit from the recent standards around transmitting SOAP messages reliably, securely, and transactionally, so the decision was made to redefine this protocol on top of SOAP.

While the Wf-XML protocol is useful in many generic situations, it had too much functionality, some of which is too specific to BPM engines, to be of general interest to the general public. We decided to break the protocol into two layers. The lower layer that is generally useful to any asynchronous service would be donated to OASIS for standardization there; an OASIS TC was formed in September of 2003. The upper layer would be contain only the extensions that are necessary for specific BPM functionality, and it is known today as Wf-XML 2.0.

At a meeting in Tokyo in October 2003, the WfMC decided to host an ASAP interoperability demonstration in the June 2004 timeframe. After discussions with several venues, the Brainstorm BPM conference was chosen as the optimal place to show a new protocol that is vital to BPM itself.

Early in 2004 announcements were distributed letting people know that a demonstration was being planned and asking for people to register to be an observer of the demonstration. The main benefit of being an observer is to receive the report of the results of the demonstration, and this document is being distributed as that report. While we had hoped to get 60 to 80 observers, we were a bit overwhelmed when over 2275 people registered to be observers. This certainly was evidence that there is intense interest in a protocol that might normally be considered somewhat esoteric.
Preparing for the demonstration required the development of several supporting documents. The ASAP Cookbook was written by Keith Swenson as a non-normative guide to how one might approach the implementation of the protocol on top of an engine that already supported BPM features of being able to execute processes. The ASAP Cookbook became a chapter in the 2004 Workflow Handbook, and was also made available through the ASAP TC website. A demo specification was drawn up by Justin Brunt of TIBCO to detail exactly what messages needed to be passed back and forth to successfully participate in the demo. The demo scenario had to complete enough to demonstrate the benefit of the protocol, while being easy to implement on a wide variety of underlying platforms. Details are presented later in the document.

A demo client and server needed to be developed. Jeff Cohen, a software developer with immense talent, volunteered his time for this. He was personally interested in using Microsoft’s .Net framework as the means of generating and responding to SOAP messages. By taking this approach, he has demonstrated that ASAP protocol can be implemented using the existing SOAP tools and libraries that are easily available today. Jeff’s implementation was chosen as the reference implementation to test all other implementations with because he does not work for a BPM vendor, and his approach is a pure implementation of a simple client, without any preconceived notions that might be introduced by implementations on top of an existing BPM engine. He started by using the .Net capability to generate C# code from our existing WSDL file definition of the services. This C# source is available for download from the OASIS ASAP TC website.

In early April 2004 an announcement went out to invite participants to implement the protocol, and to demonstrate the ability to interoperate with other implementations. John Fuller had already been developing his EasyASAP open source project, and had also been a major driving force in closure of many of the open technical issues with the spec. Fujitsu, HandySoft, and TIBCO accepted the invitation, bringing the number of participants to five: three BPM vendors and two open source initiatives.

Doing the five implementations exposed some gaps in the specification – grey areas that had been interpreted differently by different people. Getting the systems to talk required working out the details, and filling in the gaps in the specification. Mayilraj Krishnan of Cisco Systems helped tremendously by editing and maintaining the specification document through this period.

In the days leading up to June 23 the 5 implementations were tried in almost every combination to make sure all possibilities were covered. Participants had to host a machine on the Internet with their implementation running on it, at their locations in Virginia, California, Nebraska, and South Africa. The exact addresses were specified for the exact configurations to
be run. Fujitsu hosted one reference client in California, while TIBCO hosted a redundant backup client in England. HandySoft was able to demonstrate basic interoperability within three weeks of declaring their intent to participate in the demo.

The only thing left was the actual demonstration. On June 23, from 8:30 to 9:15, a plenary session at their BPM conference was kindly hosted by Brainstorm Group. Internet connections were brought in to the podium for the demonstration as well as to display slides to remote attendees. A bridge was set up between the podium microphone and the telephone system for phone conference attendees. The only surprise was the volume of attendees at the last minute exceeded some of the preset limits, so some people were turned away from being able to attend the live demonstration. A repeat demonstration was hosted a week later to allow more people to attend.

3 Demo Scenario
The demo scenario consists of three roles: a customer, a retailer, and a manufacturer. The customer sends information to the retailer as if making a purchase of a product. The retailer then delays for some number of seconds in order to demonstrate the asynchronous aspect of the protocol. Then it does one of two things: either it returns the order confirmation immediately, or it simulates being out of stock, and it makes a further request to the manufacturer role. The manufacturer then delays another moment before returning the order confirmation, which ultimately brings about the confirmation from the retailer back to the customer.

A two-party interaction was defined just between the client and retailer that was called the ASAP scenario because it did not assume any internal process logic. The full three-party interaction was dubbed the Wf-XML scenario because you need some form of process logic to call out to another server. In the end, four of the participants (EasyASAP, Fujitsu, Handysoft and TIBCO) implemented the full three party interaction, while the .Net reference client supported only two party interactions.

The demo scenario specifies an exact schema for the context data and the result data for the retailer role, as well as context data and result data of the manufacturer. The schema was kept as a simple list of named scalar values. The protocols put no restraint on the complexity of the XML structure that may be passed as context or result data, but it was assumed that keeping the structure simple would make it easy to map unto the underlying technology in the short time available. There was an obvious mapping between the retailer and manufacturer so that data could be easily carried through the whole scenario. Each implementation was asked to set a particular field with a value specific to that implementation’s sponsoring organization, so
that there would be evidence at run time that the correct engine had been accessed, and that
data had been transferred.

Each of the four implementations of the full scenario exposed to the Internet two factory
addresses: one for the retailer process and one for the manufacturer process. The reference
client was used to invoke the retailer process for each of the four retailer implementations, and
that process in turn invoked the manufacturer implementation from the same vendor. For
example, the Staffware retailer invoked the Staffware manufacturer role, and so on. Each of the
4 full implementations experimented with invoking the .Net reference asynchronous service to
be sure that they could call out to a standard service. Then cross vendor tests were performed.
The .Net reference client was used to invoke a Fujitsu Interstage BPM retailer process, which in
turn invoked the manufacturer process on EasyASAP, HandySoft, or TIBCO1. All of these
interactions were shown to work before finally settling on having the Fujitsu process call the
HandySoft process for the actual demo. This choice was made simply because it gave the best
visual impression at demo time, but any vendor could have been substituted for either of the
roles for the actual demo.

4 Lessons Learned

The biggest area of difficulty was coming to a precise definition of how to define the context
and result data schema. It was easy enough to define XML structures that come from the
factory resource that are defined as XMLSchema tags. The difficulty was how to define the
messages to and from the instance resource, where the main part of the message has a fixed
structure defined by the WSDL definition of the operation, but inside is section of that message
(the context data) that is defined by the schema that is elsewhere part of the protocol. Initial
implementations attempted to do this a couple of different ways. The final solution was to use
the XML Schema tags in the factory resource to define two named types: ContextDataType
and ResultDataType. The fixed WSDL interface for the base protocol defines messages that
have tags of this type, but does not define the type. This was determined as the best way to
have a structure that is run-time modifiable. It is, of course, possible to generate a WSDL file
at runtime that contains the complete structure (in fact this is highly recommended), but most
tools that consume WSDL are more oriented toward design-time than run-time.

1 Note, due to time and equipment constraints, Staffware actually only exposed one factory resource, the retailer
resource, so the Fujitsu process actually called Staffware’s retailer process instead of the manufacturer’s process,
but this is an entire equivalent interaction. Note also that in the middle of preparation for this demonstration,
TIBCO announced that it was purchasing Staffware, and so the final demonstration was actually represented by
TIBCO. This document uses the name “Staffware” to refer to the software product, while “TIBCO” is used as the
organization that the people work for and that supported the effort, even though that organization was not
named TIBCO at the time.
A second area that caused a great deal of trouble was simply that of XML namespaces. There are three ways to declare the namespace of a particular tag in XML, but the .Net tools were requiring that if an XML Schema declared the tags with one of these, that the actual implementation had to use the same method of specifying the namespace. We ended up requiring that all implementations declare a namespace prefix, and fully qualifying all tags with the prefix.

We also had a lot of trouble with a seemingly innocuous change of taking an unnamed substructure (the history element) and changing the schema to explicitly declare that substructure as a named type. Even though the actual structure did not change, the introduction of naming the part of the structure cascaded through and caused unexpected problems with some implementations.

## 5 Successful Configurations

The following cases were tried and shown to work:

<table>
<thead>
<tr>
<th>Observer</th>
<th>Factory</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>.Net reference client</td>
<td>Fujitsu (retailer)</td>
<td>✓</td>
</tr>
<tr>
<td>.Net reference client</td>
<td>Staffware (retailer)</td>
<td>✓</td>
</tr>
<tr>
<td>.Net reference client</td>
<td>HandySoft (retailer)</td>
<td>✓</td>
</tr>
<tr>
<td>.Net reference client</td>
<td>EasyASAP (retailer)</td>
<td>✓</td>
</tr>
<tr>
<td>.Net reference client</td>
<td>Fujitsu (manufacturer)</td>
<td>✓</td>
</tr>
<tr>
<td>.Net reference client</td>
<td>HandySoft (manufacturer)</td>
<td>✓</td>
</tr>
<tr>
<td>.Net reference client</td>
<td>EasyASAP (manufacturer)</td>
<td>✓</td>
</tr>
<tr>
<td>Fujitsu (retailer)</td>
<td>.Net reference server</td>
<td>✓</td>
</tr>
<tr>
<td>Staffware</td>
<td>.Net reference server</td>
<td>✓</td>
</tr>
<tr>
<td>HandySoft</td>
<td>.Net reference server</td>
<td>✓</td>
</tr>
<tr>
<td>EasyASAP</td>
<td>.Net reference server</td>
<td>✓</td>
</tr>
<tr>
<td>Fujitsu (retailer)</td>
<td>HandySoft (manufacturer)</td>
<td>✓</td>
</tr>
<tr>
<td>Fujitsu (retailer)</td>
<td>EasyASAP (manufacturer)</td>
<td>✓</td>
</tr>
<tr>
<td>Fujitsu (retailer)</td>
<td>Staffware (retailer)</td>
<td>✓</td>
</tr>
</tbody>
</table>
6 Instructions for Self Demonstration

The participants in the recent live demonstrations of the ASAP and Wf-XML protocols have continued to make their implementations available for others to try out. The demonstration implementations are accessed using a test harness which is hosted on two Web Servers: one is hosted by Fujitsu (http://67.113.139.106/ASAPClient/) and the other is hosted by TIBCO (http://193.131.190.26/ASAPClient/).

All of the implementations with the exception of the .NET reference factory have implemented a simple supply chain business process. Details of the scenario can be found in the accompanying document: Demo_scenario_1k.pdf

6.1 Select the Factory to Test

Navigating to one of the URLs will display the screen below showing the different implementations available to test.

The list of factories that are available for testing may change over time. You may either select one of the implementations from the list and press the appropriate button to initiate the test or you can type the URL in the field at the top of the screen and press the submit button. The latter option is useful if you want to test an implementation that is not listed.
6.2 View Factory Information

Once a selection has been made a screen similar to the one below will be displayed:

![Factory WfXML 2.0 Demo - Retailer System](image)

Using this screen you can create a test instance using the selected implementation, list all previous instances for this implementation or go back to select a different implementation to try out.
6.3 **Create an Instance of the Service**

If you choose the *Create Instance* a screen similar to the one below will be displayed.

Although not mandatory, you should fill in the details in the top three fields so that you can review your test results later using the *List all instances* option. In the *Context Data* field at the bottom fill in your test data using the schema presented as a guide. The scenario devised for the tests was specifically designed so that they accept any product description and code etc. so that testers have no need to know what are valid products and product codes. Consequently, don’t be surprised to see some strange prices for the *products* you order since each implementation has devised its own algorithm for calculating prices!

If you enter an odd product code then the implementations will pretend that the *product* you have ordered is not in stock and has to be *manufactured* for you. In such cases the implementations will invoke another process using the ASAP protocol to initiate the manufacture of your *product*. 
Once you have entered the test data press the *Create* button and your test instance will be instantiated and a screen similar to the one below will be displayed.

**Factory WfXML 2.0 Demo - Retailer System**

<table>
<thead>
<tr>
<th>Subject</th>
<th>&lt;none supplied&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Odd product code will determine whether the product is to be manufactured to order or is in stock at the retailer. If the product needs to be &quot;manufactured&quot; to order the retailer will supply manufacturer_order with the process instantiation.</td>
</tr>
<tr>
<td>Expiration</td>
<td></td>
</tr>
</tbody>
</table>

Create instance  List all instances  Change factory

**Your Instances**

<table>
<thead>
<tr>
<th>Name</th>
<th>Subject</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Justin Brunt</td>
<td>ASAP Test</td>
<td>Open running</td>
</tr>
</tbody>
</table>

Refresh  Remove checkpoint

The above information is what the client knows about the instances you created in this browser session. The server was not contacted in order to generate this page, though it does reflect any asynchronous notifications the server sent this client.

### 6.4 Wait for Completion

You can periodically press the *Refresh* button to find out when the test instance has completed. Once it has completed the screen will change as shown below.

**Your Instances**

<table>
<thead>
<tr>
<th>Name</th>
<th>Subject</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Justin Brunt</td>
<td>ASAP Test</td>
<td>Closed completed</td>
</tr>
</tbody>
</table>

Refresh  Remove checkpoint
6.5 Review Details of Instance

If you want to check the results of the test in detail press the Details button to see a screen similar to the one below where the context data shows your order.

```
Name: Justin Brunt
Subject: (none supplied)
Description: Testing the Handysoft implementation
State: closed:completed
Key: http://63.137.54.86/bizflow/services/wfmc/serviceProcInst?pid=206
Factory: http://63.137.54.86/bizflow/services/wfmc/serviceProcDef?pid=103
```

### History

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>State</th>
<th>Source Key</th>
</tr>
</thead>
</table>

### Context Data

```
<ns1:ContextData xmlns:ns1="http://63.137.54.86/bizflow/services/wfmc/service/ProcInst?pid=206">
  <ns1:customer_first_name>Justin</ns1:customer_first_name>
  <ns1:customer_surname>Brunt</ns1:customer_surname>
  <ns1:address_first_line>TIBCO Inc</ns1:address_first_line>
  <ns1:address_second_line>Kumbrey Park</ns1:address_second_line>
  <ns1:address_city>Boulogne</ns1:address_city>
</ns1:ContextData>
```

Hopefully the tests that you performed will be successful and will give you an idea of the power and applicability of ASAP and Wf-XML 2.0. Please post your questions, suggestions or comments to the Workflow Forums ([http://www.workflow-research.de/Forums/](http://www.workflow-research.de/Forums/)), under WfMC Technical Committee to the thread “ASAP/Wf-XML Demo Report.”

7 Future Challenge

Having demonstrated interoperability at the ASAP level between five different implementations, we now need to take the next step with another demonstration of interoperability at the Wf-XML level. Currently there is a great need for a standard protocol to allow business process design tools and BPM engines to work together. The factory resource represents a BPM process definition, and Wf-XML offers a way to retrieve this process definition, update it, and a way to create new factories for new process definitions. Our new goal then is to demonstrate interoperability between a couple of different BPM engines, and a couple of different process design tools, exchanging either BPEL or XPDL process definitions in the January 2005 timeframe.
8 Documents and Resources

The current ASAP specification is wd-asap-spec-01g.pdf located at:

http://www.oasis-open.org/committees/download.php/7151/wd-asap-spec-01g.pdf

The associated XMLSchema file is at:

http://www.oasis-open.org/committees/download.php/7147/asap.xsd

The demo scenario document is at:

http://www.oasis-open.org/committees/download.php/7177/Demo_scenario_1k.pdf

The reference .Net client and server with installation instructions is at:

http://www.oasis-open.org/committees/download.php/7197/ASAPclient.zip
http://www.oasis-open.org/committees/download.php/7197/ASAPserver.zip

The slide deck that was used at the demo presentation can be downloaded from:

http://www.oasis-open.org/committees/download.php/7590/ASAP_BrainStorm.ppt

The current Wf-XML 2.0 standard can be retrieved from:


Additional information may be found at:

http://www.wfmc.org/standards/wfxml_demo.htm

Workflow Management Coalition (WfMC.org)
2436 North Federal Highway #374,
Lighthouse Point, FL 33064, USA