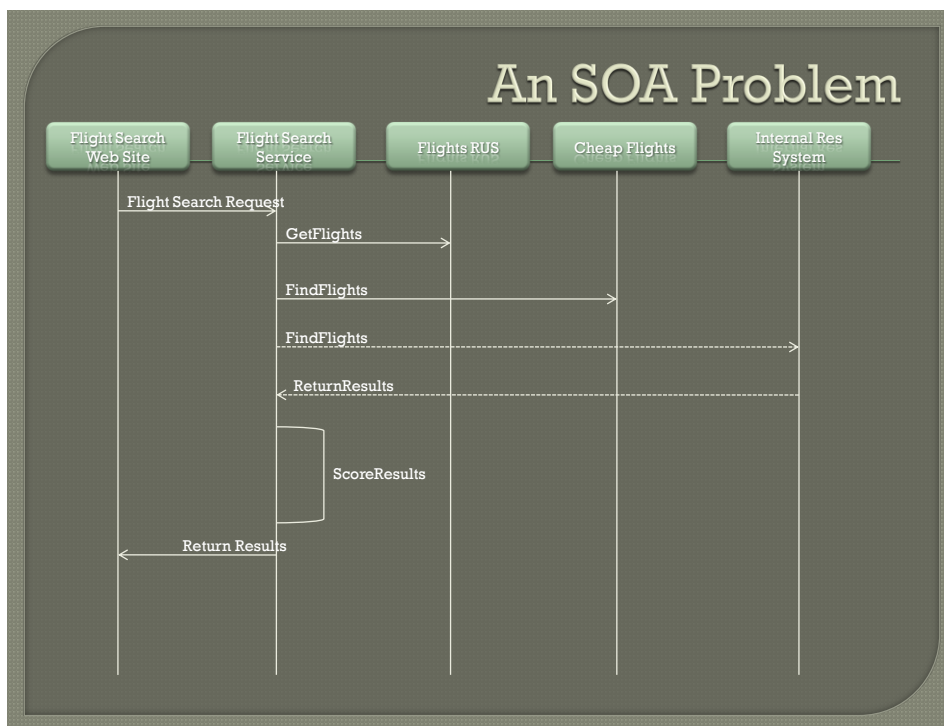


BizTalk vs. WF/WCF Smack Down

A few months ago I had a great opportunity to help create a sample Windows Workflow Foundation (WF) solution for a SOA Roundtable event at the North Central District Microsoft office. I had previous experience with WF but this was still a challenging experience that gave me the opportunity to learn even more about the technology and when it should be utilized. The purpose of building the sample was to be able to compare it, at the SOA Roundtable event, to a BizTalk solution that had already been created. Both solutions solved the same business problem. The business problem was centered on a service-oriented solution to requesting and aggregating the responses from three airline ticket services.

From a process perspective, a client sends a request to a SOAP endpoint that was actually an aggregation service. The aggregation service, originally built in BizTalk Server 2006, communicates with three different mock airlines' ticket systems, exposed as three services, over different transports, all of which required different message formats. After receiving responses from the three different airlines, the service then aggregated the responses and returned a single response message to the client application.



Before we get to deep into the "Smack Down" let's go back to 2004 and briefly review the four fundamental tenants of SOA:

Boundaries are explicit

This means is that we're acknowledging boundaries (geographic, trust boundaries, processes, etc.) as opposed to trying to hide them like previous distributed technologies attempted to do.

Services are autonomous

Instead of imagining that your application is deployed as a single atomic unit, we're acknowledging that our applications are built on constantly changing services and that different services age at different rates.

Services share schema and contract, not class

We only share structures and behaviors with our services. Each service defines the structures and patterns of interaction that it accepts. With services, we don't share the actual classes that we used underneath the services.

Service compatibility is determined based on policy

Each service should publish its capabilities and requirements to the world in a machine readable way. If a message is sent to the service that doesn't meet the policy assertions that the service defines then the message doesn't get to interact with the service.

If you haven't come across the four tenants before, check out the [A Guide to Developing and Running Connected Systems with Indigo](#) from MSDN magazine.

Now we have some context for our SOA solution and get back to the problem at hand. The specific business problem that we were trying to solve is a typical one for BizTalk to solve and we wanted to discover how you'd implement a similar solution using WF. The results of that discovery will be covered in the remainder of this article. In this article we'll loosely follow the development process:

1. Define and Publish External Server Service Contract
2. Define Composite Service
3. Identify external Service contracts and import definitions
4. Define and Build the Business Rules
5. Build and Deploy Service
6. Review real world impacts to the new service

As we go through the development process we'll evaluate, where appropriate, the following architectural Issues:

- Service Hosting
- Tracking
- Persistence
- Communication
 - Web Services
 - WCF

- MSMQ
- Transformation
- Rules Engine
- Deployment
- Administration

After comparing the two solutions and some of their infrastructure, I'll finish up with some observations on where the two technology solutions should and shouldn't be used. I have some fun with the comparing and contrasting but in all seriousness, both technologies have their benefits and can be used in many different scenarios. Enough with the introduction... LET THE SMACK DOWN BEGIN!!!

1. Define and Publish External Service Contract

We won't spend a ton of time on the define part of this stage. Defining what's contained in a service contract is really the domain of your requirements and design process. However, one of the generally accepted rules in defining these contracts is that they are described in XML. This is where the fun begins in our SMACK DOWN!

Now that you've gone through your design process and you have this beautiful XML contract you'd like to publish that as say an .asmx 1.1 web service or a WS-I* service so you break open Visual Studio. Easy you say.... So try... I'll wait.... Hummmmmmm hmuuummm. Give up yet? (OK I know there are a few of you out there that have tried this and know the trick is to use either XSD.EXE that ships with Visual Studio or Svcutil.exe that ships with the .NET Framework 3.0 and is used as part of the WCF).

So what do these utilities do? They generate a set of nice .NET classes decorated with persistent attributes that describe your contract in .NET. This is no longer your pure XML contract. The upside is you can tweak and change how things work. But you being a good architect and understanding that things change say no way. But we still need to add this code to a web service we have defined in Visual Studio. We'll use this as an interface veneer for our service. This service will simply expose the .NET/XML contract and call another service or .NET assembly that actually does the work! Now when ever things change in the contract we can run the utilities, generate the new interface and add the lines of code to our common service. So let's keep score:

We now have three .NET projects

- Contract project generated by the utilities
- Web Service Interface Project used to expose the document contract classes
- Business Service Project where we do the real work

OK now the .NET guns are saying all right big shooter, how do you do this in BizTalk? Well, full disclosure: we have a set of Wizards that publish our XML document contracts as either 1.1 Web Services or WS-I web services. But they're a tad more advanced as they not only generate the .NET classes representing the XML contract but also generate the web services interface code complete with extended headers to support security and Single-Sign-On (SSO) capabilities. BizTalk ultimately produces a Visual Studio

project and code with all the plumbing complete to expose the service externally as well as to connect to the BizTalk messaging engine!

So we're keeping score:

- Contract and Interface Project Generated by BizTalk
- With WF you generate the contract and interface manually

But those of you keeping a close eye with notice that we don't have a project for the Business Services. You are correct, we will but unlike a pure .NET approach, BizTalk is extremely loosely coupled. As a result there is NO CODE that ties the interface façade to the Business Service! BizTalk binds these two up at run time!

2. Define Composite Service

This is really the meat and potatoes of SOA. If we can't add flexibility and agility to our systems with this approach, then it is just gold plated plumbing. If you remember from our introduction, our new service is an aggregator. An aggregator could be deterministic where we know we'll always send out three requests and receive three responses or it could be non-deterministic where we send a variable or static number of requests but only wait for responses until a specific condition is met. The new flight search service will aggregate responses from two external partners and one internal systems. Based on these responses it will score the result set based on business rules and return the best deal.

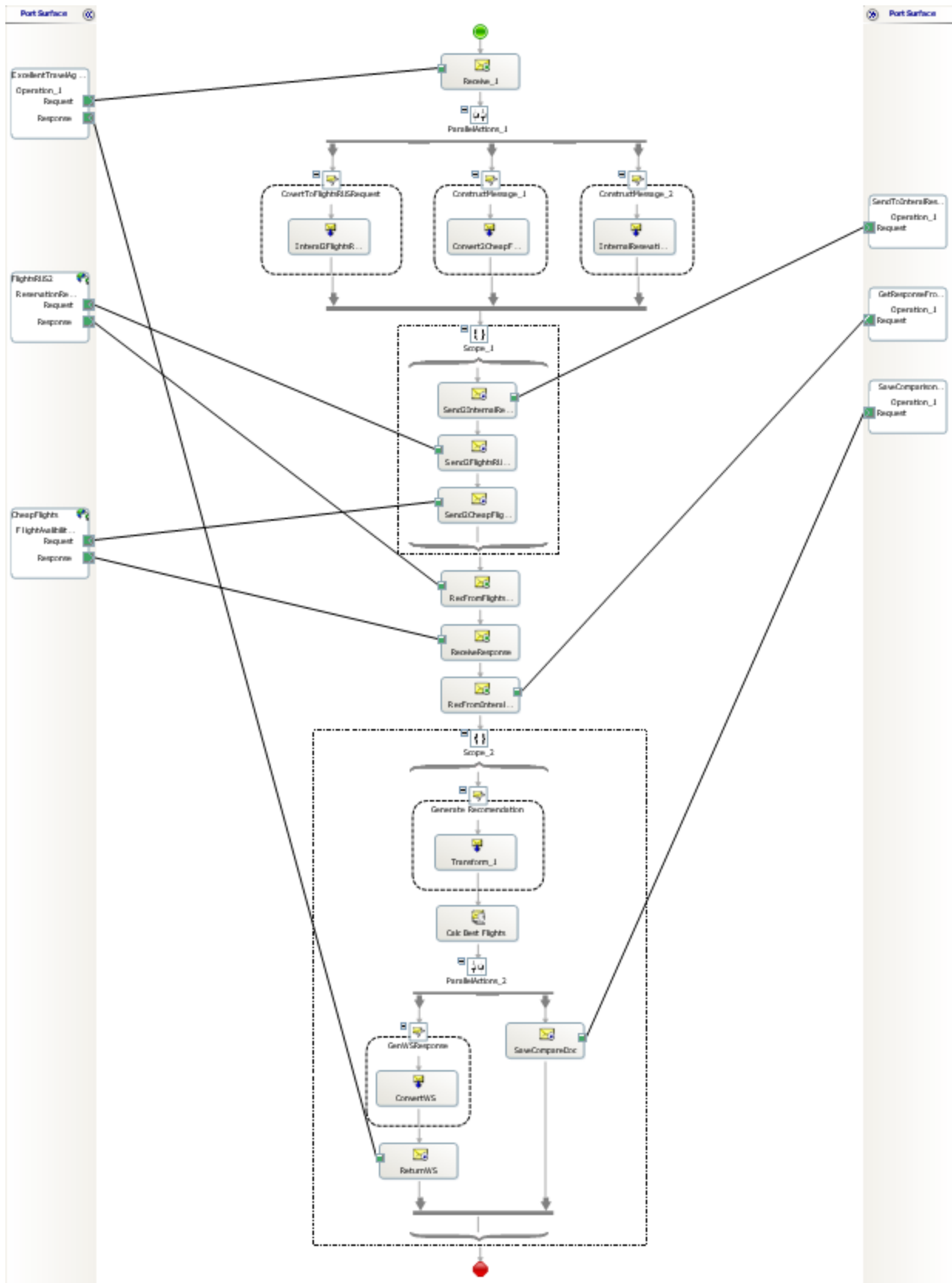
There are a couple of architectural issues here:

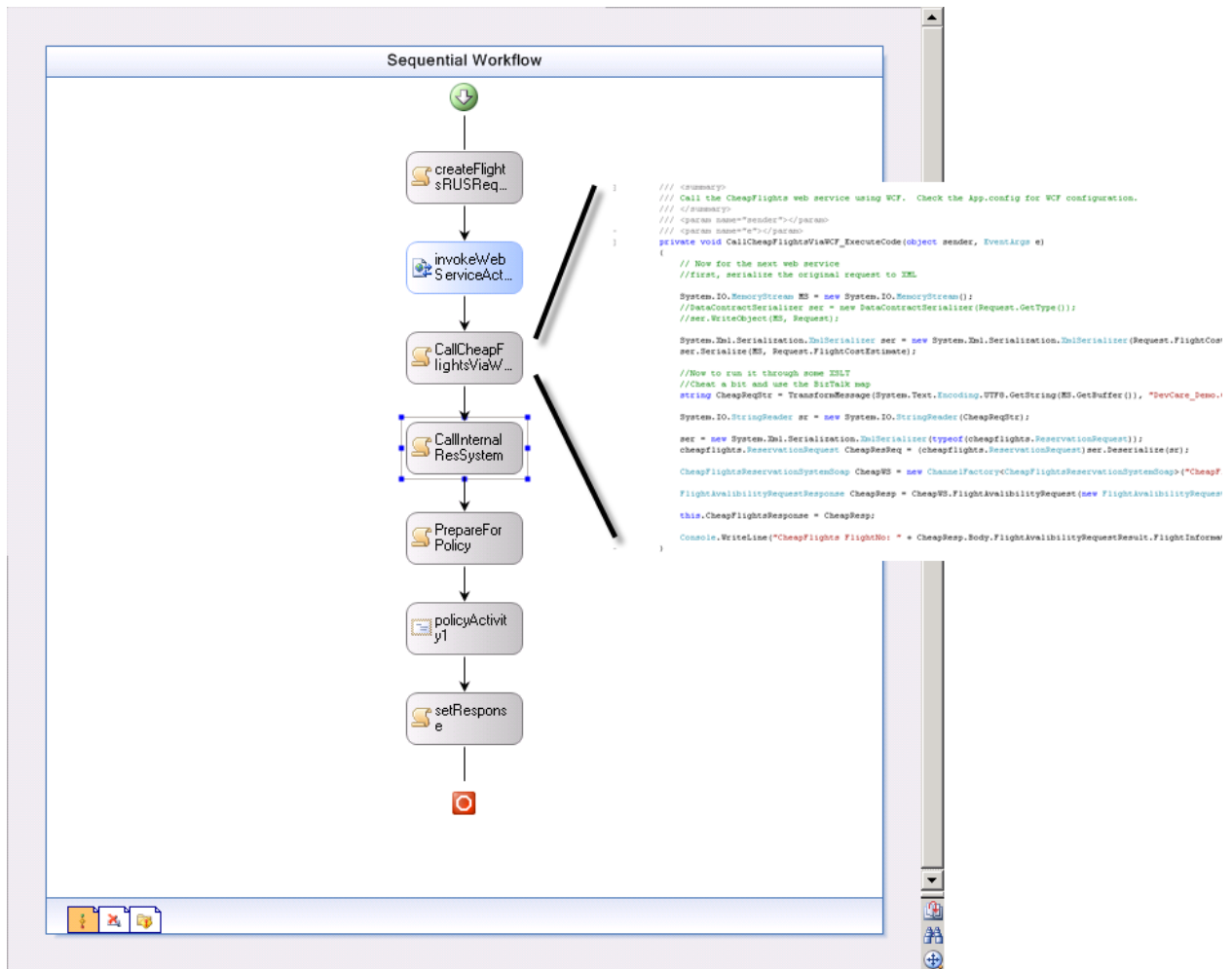
1. The external partners have different service definitions
2. Our internal reservation system uses MSMQ as its communications protocol
3. External Vendors can and will change their interfaces
4. We may need to add another partner

To make this easier we'll assume that we'll have the same partners and we won't address interface changes we'll only talk about them.

In both the WF and BizTalk solutions a graphical designer was used to draw the process flow. In each case, .NET code is generated from the diagram that is drawn. In BizTalk this concept is called orchestration, as in orchestrating services. In WF this is, unsurprisingly called a workflow.

BizTalk orchestrations are used to generate C# code which is then compiled into intermediate language (MSIL.) WF Workflows on the other hand can be created using the WF designer, using XML that follows the XAML format, using VB, C# and combinations of designer-XAML-code.





It is worth noting that BizTalk includes a static set of orchestration shapes that all BizTalk orchestrations can use. You cannot add to this set of shapes but you can compose orchestrations from other orchestrations and call out to external .NET assemblies. WF, on the other hand, includes a set of Activities that can be added to and built upon to build workflows, which are really specific Activities. If you need a new Activity, you can build it by sub-classing some WF defined Activity classes. In WF, you can also call out to external .NET assemblies.

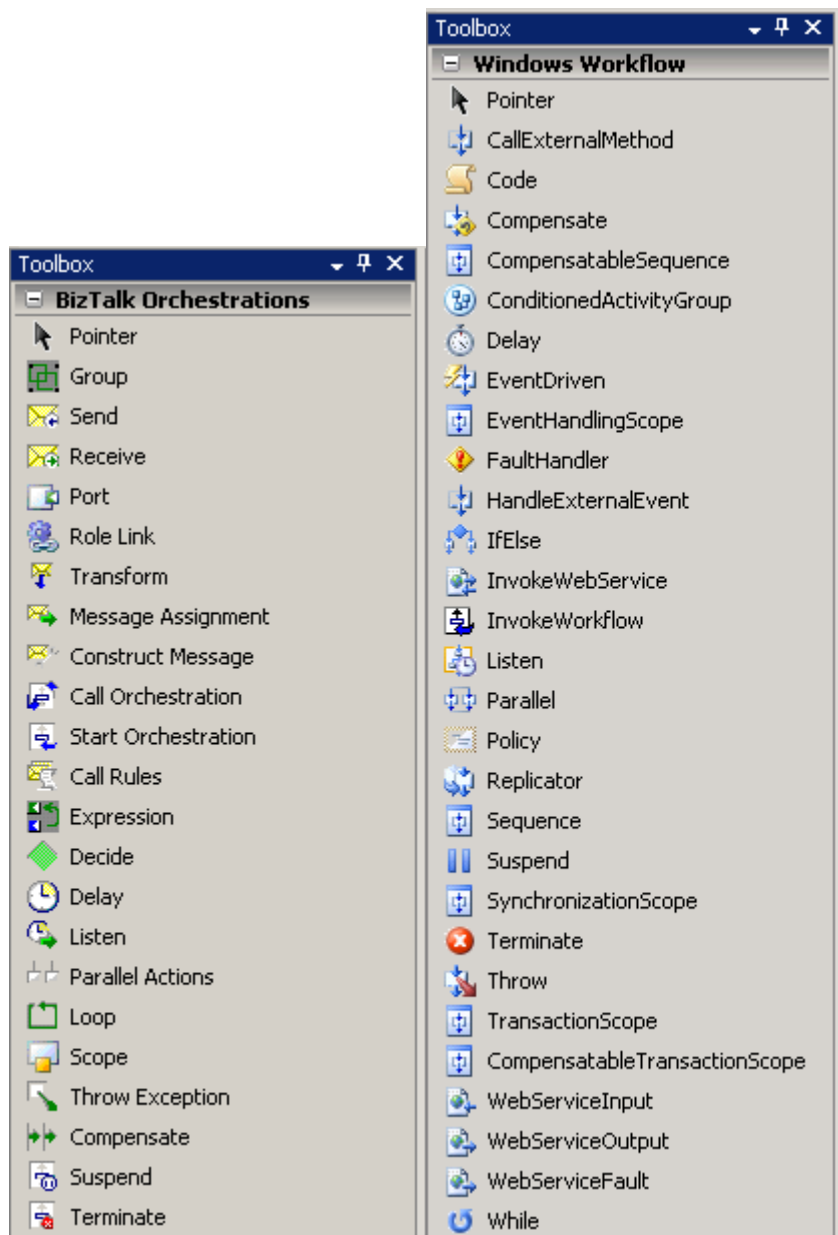
But there are some fundamental differences. BizTalk orchestrations take care of a lot of plumbing code for the developer. Things like correlation, state management, service contract exposure to name a few. WF on the other hand provides a lower level of control as many of the activities that a BizTalk Orchestration performs for you code must be written in WF. One of the major differences is BizTalk is designed to process XML data and perform transformation. WF was developed more generically so things like data transformations need to be hand written. But WF does have the advantage of an interactive debugger ;-)

In this solution BizTalk communicated with the two web services using logical ports within the orchestration as well as Send and Receive shapes. The actual sending and receiving functionality was

handled through Send Ports. Send Ports are essentially endpoint metadata; however, part of the metadata is actually a reference to a BizTalk Adapter. Adapters are the components of BizTalk that perform all communication with external actors (systems, applications, users, etc.) In this solution the two web services endpoints were communicated with using the BizTalk SOAP Adapter. There is a wizard included with BizTalk that allows you to “Add Web Reference”, similar to how you “Add Web Reference” in a “normal” .NET project. After you add the solution you wire up some orchestration shapes to the generated ports and you’re done.

It is worth noting that BizTalk is fundamentally loosely coupled. We designed our aggregation services as an orchestration and interacted with the external services via logical ports. The logical ports are essentially contracts describing the communication patterns and documents being exchanged. In the orchestration diagram, the grey surfaces on the left and right side of the screenshot, referred to as the Port Surfaces, are the logical port surfaces. The center portion of the diagram is the orchestration design surface and where you implement your process in a diagram-centric manner.

WF contains a handful of activities for communicating with Web Services. Activities are shapes that you can place on the WF designer surface, within a Workflow and utilize them. They are analogous to components in component-based development. Here’s a side by side comparison of the out-of-the-box shapes/activities:



In this solution, we used the `InvokeWebServiceActivity` for one of our web services calls. The `InvokeWebServiceActivity` is used when communicating with the older model web services, such as web services created using the .NET Framework 1.1 or 2.0. For the other web service call, we generated a proxy using the .NET 3.0 `svcutil.exe` and called it from a `CodeActivity` within the workflow. We decided to do this so we could compare and contrast using these two different techniques, just within WF.

Here's a screenshot of the code we used to invoke one of the services using WCF:

```

]     /// <summary>
]     /// Call the CheapFlights web service using WCF. Check the App.config for WCF configuration.
]     /// </summary>
]     /// <param name="sender"></param>
-     /// <param name="e"></param>
]     private void CallCheapFlightsViaWCF_ExecuteCode(object sender, EventArgs e)
    {
        // Now for the next web service
        //first, serialize the original request to XML

        System.IO.MemoryStream MS = new System.IO.MemoryStream();
        //DataContractSerializer ser = new DataContractSerializer(Request.GetType());
        //ser.WriteObject(MS, Request);

        System.Xml.Serialization.XmlSerializer ser = new System.Xml.Serialization.XmlSerializer(Request.FlightCostEstimate.GetType());
        ser.Serialize(MS, Request.FlightCostEstimate);

        //Now to run it through some XSLT
        //Cheat a bit and use the BizTalk map
        string CheapReqStr = TransformMessage(System.Text.Encoding.UTF8.GetString(MS.GetBuffer()), "DevCare_Demo.Convert2CheapFlightsFormat, DevCare Demo");

        System.IO.StringReader sr = new System.IO.StringReader(CheapReqStr);

        ser = new System.Xml.Serialization.XmlSerializer(typeof(cheapflights.ReservationRequest));
        cheapflights.ReservationRequest CheapResReq = (cheapflights.ReservationRequest)ser.Deserialize(sr);

        CheapFlightsReservationSystemSoap CheapWS = new ChannelFactory<CheapFlightsReservationSystemSoap>("CheapFlightsReservationSystemSoap").CreateChan
        FlightAvailabilityRequestResponse CheapResp = CheapWS.FlightAvailabilityRequest(new FlightAvailabilityRequestRequest(new FlightAvailabilityRequestReq
        this.CheapFlightsResponse = CheapResp;

        Console.WriteLine("CheapFlights FlightNo: " + CheapResp.Body.FlightAvailabilityRequestResult.FlightInformation.FlightNumber);
    }
}

```

Two other interesting things about these implementations: Sequential vs. parallel execution. The BizTalk Orchestration engine is designed so that the three services that we need to call can be called in parallel without requiring us to manage threads. This was accomplished using an atomic scope shape which sent all three messages to the external systems at the same time. The WF implementation calls the services in sequence. This means that the WF component needs to complete the call before calling the next service!

Let's check the score:

- Both WF and BizTalk have visual designers for implementing processes
- BizTalk shapes are static; you cannot add to the out-of-the-box shapes (although you can encapsulate shapes into orchestrations that can be called from other orchestrations)
- WF's activities (shapes) are extensible
- WF has "F5" debugging ;)

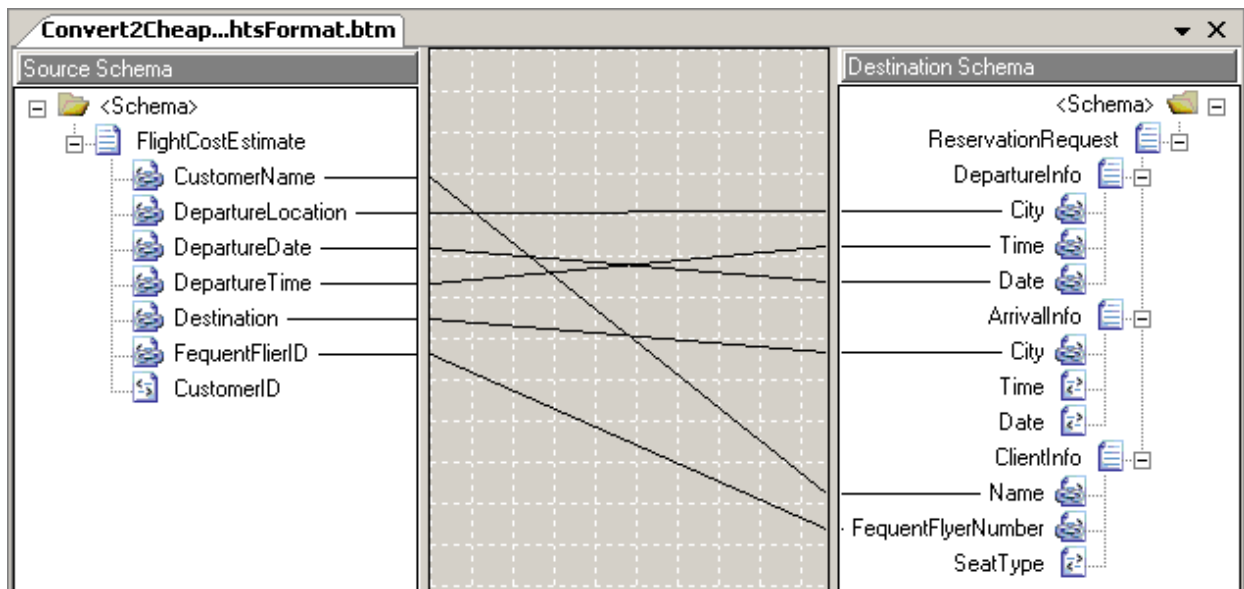
But we've missed something important. Those external services are NOT using our XML contract. So we need to convert our message contract to their message contract and vice versa. We'll cover how we did this in the Transformation section, below.

3. Identify External Service Contracts and Import Definitions

In both solutions, we were dealing with different data structures when we communicated with the three ticket services. In the WF solution, we were dealing with the generated class structure (from either "Add Web Reference" or WSDL.exe) as well as an object that was "passed into" the Workflow which in this case was the "Request" object that our WCF host accepted from the client. The "Request" object did not have the same class structure that we needed to send to the three ticket services. We had to

transform the request into each service's specific class format prior to sending the request to the respective service. In two of the cases, we accomplished this using code maps from one class structure to another, in a WF CodeActivity. In the other case, we used some pit crew ingenuity and called a BizTalk Map using code, in a WF CodeActivity. We were able to accomplish this because the source and target types were generated using WSDLs; the same WSDLs that were created and used in the BizTalk solution.

In the BizTalk solution, we also had to map between different data structures but you accomplish this using the Mapper in BizTalk. The Mapper is a Visual Studio Add-In for graphically creating XSLT. In our case, we created the Maps and then configured them to run within the orchestration. In most cases with BizTalk, you can accomplish your transformation without resorting to writing XSLT manually or by writing code.



```

/// <summary>
/// Call the FlightsRUS web service using a web reference.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void createFlightsRUSRequest_ExecuteCode(object sender, EventArgs e)
{
    FlightsRUS.ReservationQuery ResQuery = new FlightsRUS.ReservationQuery();

    FlightsRUS.FlightRequestInfo ResInfo = new FlightsRUS.FlightRequestInfo();

    ResInfo.DepartureCity = Request.FlightCostEstimate.DepartureLocation;
    ResInfo.DepartureDate = Request.FlightCostEstimate.DepartureDate;
    ResInfo.DepartureTime = Request.FlightCostEstimate.DepartureTime;
    ResInfo.ArrivalCity = Request.FlightCostEstimate.Destination;

    ResQuery.FlightRequest = ResInfo;

    FlightsRUS.ClientData ResClient = new FlightsRUS.ClientData();
    ResClient.Name = Request.FlightCostEstimate.CustomerName;
    ResClient.FrequentFlyerNumber = Request.FlightCostEstimate.FrequentFlierID;

    ResQuery.ClientInfo = ResClient;

    this.FlightsRUSQuery = ResQuery;
}

```

Now before you go off and say that either the BizTalk Orchestrations or WF component are the way to go lets point out that both technologies are written by the same team and that if we look at BizTalk R2 they are moving closer to each other.

But since we've been keeping score:

WF

- Consuming External Web Services – Need to write some code and we're tightly coupled
- Transformations - .NET Code based
- Debugging – More natural in Visual Studio then BizTalk

BizTalk

- Consuming External Web Services – Reference WS and contract is generated
- Transformation – Defined graphically and performed in XSLT
- Debugging – Can be done from Visual Studio but you need to use and read the compiler generated code.

The orchestration and workflow concepts won't be covered in detail in this article; instead they are mentioned as a reference for other topics as they could be the subject of an article by themselves.

4. Define and Build the Business Rules

In both the WF and BizTalk solution, we leveraged a Rules Engine to score the responses we received from the airline ticket services. A full comparison of the two Rules Engines would be several articles length so this will be a cursory view of what we did for the SOA Roundtable event.

BizTalk's Rule Engine (BRE) is a complex and powerful Rules Engine that has many advanced features compared to the WF Rules Engine. A few of these are:

- An external Rules Composer application
- Support for forward-chaining
- Support for Vocabulary, which is a way to abstract the rules into domain specific (business) language.
- Support for multiple fact types: XML, .NET Objects and SQL Database structures
- Support for multiple facts going into and coming out of the BRE
- A testing tool within the Rules Composer
- Support for versioning Rules
- A database rules store

The WF Rules Engine has:

- A Rules Editor that resides within Visual Studio
- Support for forward chaining as well as options for defining how dependencies are defined
- Supports .NET Objects as Facts
- Supports one fact going into the Rules Engine and one fact coming out
- No support for versioning Rules
- Compiled (within the workflow's assembly) rule store (although rules can be stored in external rule stores if you build the support yourself)

With this background information in mind we can discuss the implement of the two solutions. In both cases the Rules were very simple and the RuleSets, which are containers for a set of Rules, only contained a couple of Rules.

In the WF solution, because of the more restrictive way that you have to interact with it, we had to build a container object for the three response objects that were received from the three ticket services. We also had to add a "Selected" property to this new class. At runtime, the container object was then prepared for the WF Rules Engine like this:

```

/// <summary>
/// Setup the PolicyHelper object that the PolicyActivity will execute against.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void PrepareForPolicy_ExecuteCode(object sender, EventArgs e)
{
    this.PolHelper = new PolicyHelper();

    this.PolHelper.CheapFlightsResponse = this.CheapFlightsResponse;
    this.PolHelper.FlightsRUSQueryResults = this.FlightsRUSQueryResults;
    this.PolHelper.InternalResResponse = this.InternalResResponse;

    this.PolHelper.Response = new Operation_1Response();
    this.PolHelper.Response.WSResponseDoc = new WSResponseDoc();
    this.PolHelper.Response.WSResponseDoc.TicketCost = "";

    Console.WriteLine("Prepared PolicyHelper...");
}

```

The this.PolHelper object was then used by the Rules Engine as the container object for its execution.

In the BizTalk solution, we simply mapped our three response documents, from the three services, to a single aggregated document which we then passed into a Policy that determined which ticket option to recommend. The winner was marked by setting a value in the aggregated document.

Since we're keeping score:

- Both BizTalk and WF have built-in rules engines
- The BizTalk rules engine is more complex and more powerful
- The WF rules engine is simpler but more accessible to your average developer.

5. Build and Deploy Services

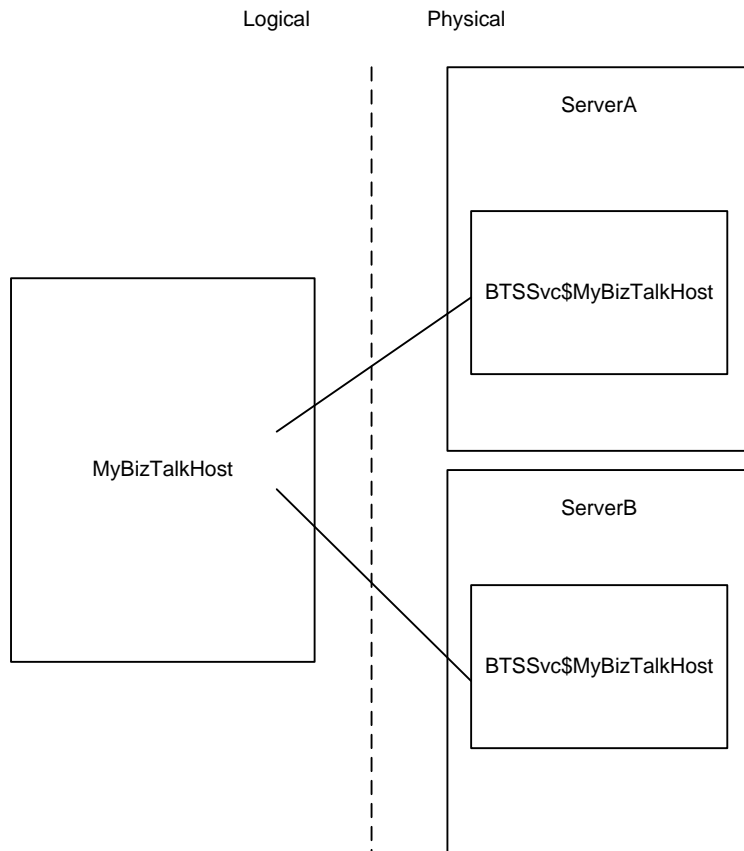
Hosting

One of the requirements for the ticket aggregation service was that it be exposed via a web service. In both cases, we needed to create a host for the service using different mechanisms because of the differing technologies. To be clear, a host in this context refers to a Windows process and the infrastructure required for hosting a web service endpoint.

In BizTalk, we first used an included wizard, the BizTalk Web Services Publishing Wizard, to generate an SOAP .asmx project, and a Receive Port and Receive Location for our orchestration. The Receive Port and Receive Location can be thought of as that SOAP endpoint's configuration data, including things such as a URL, security settings, message batching options and protocol specific items. The .asmx endpoint was hosted within IIS 6.0 which in BizTalk terminology is an Isolated Host.

BizTalk has a hosting concept, and they are called Hosts. Hosts are logical concepts in BizTalk that are containers for the BizTalk runtime engines as well as the solutions running within those engines. Hosts apply to an entire group, which is BizTalk terminology for a server farm. The Host concept in BizTalk handles load distribution, high-availability, scale-out, provisioning of Host Instances (see below) and they act as containers for BizTalk's internal queue tables. Hosts are also containers for performance and load tuning configuration. You can also think of the host concept in BizTalk as being analogous to a COM+ or MTS middle tier server although it is slightly more advanced and has more features.

Host Instances are instances of their specific Host, running the BizTalk engines and solutions on a specific server in the BizTalk group. These are real Windows processes, starting with the name "BTSNTSvc". You can have one Host Instance, per logical Host within your BizTalk group, for each BizTalk server within your group.



To keep things simple, you can think of Hosts as your Hotrod's blueprint for its wiring harness and infrastructure while Host Instances are the actual harnesses.

In WF and WCF, there aren't any fancy knobs or dials for the host concept. For that solution we created a WCF service, starting with Svcutil.exe used against the .wsdl from the BizTalk solution, to act as the host for a WF workflow that performed the bulk of the processing for our solution. Svcutil.exe is included with .NET 3.0 and it generates proxies for WCF. Interestingly, it doesn't have a switch for

creating server proxies. We generated a proxy and then deleted the client-side code that it generated. For convenience, the WCF service itself was hosted within a Console application although we could have hosted it within IIS 6.0 with only a little more effort – we simply ran out of gas and time. The console host we created was very simple and we did not attempt to implement features such as load distribution, high-availability or scale out. Since you have to build your own host you have more control over its functionality compared to BizTalk's host infrastructure. It also takes a lot of code and effort to deliver a quality host solution using WF.

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Workflow.Runtime;
using System.Workflow.Runtime.Hosting;
using AggregateReservationsWorkflows;

namespace AggregateReservations
{
    /// <summary>
    /// This is the WCF Aggregate Reservations Service.
    /// </summary>
    public class AggregatorServiceType : DevCare_Demo_AggrigateReservations_ExcellentTravelAgentsSoap
    {
        private Operation_1Response _response = null;

        #region DevCare_Demo_AggrigateReservations_ExcellentTravelAgentsSoap Members

        public Operation_1Response Operation_1(Operation_1Request request)
        {
            Console.WriteLine("AggregateReservations Request Received...");

            //Prepare the parameters to pass to the workflow instance.
            Dictionary<string, object> parameters = new Dictionary<string,object>();
            parameters.Add("Request", request);

            //Now instantiate a workflow runtime
            Type workflowType = typeof(ResWorkflow);
            WorkflowRuntime runtime = new WorkflowRuntime();
            runtime.ServicesExceptionNotHandled +=new EventHandler<ServicesExceptionNotHandledEventArgs>(runtime_ServicesExceptionNotHandled);
            runtime.WorkflowAborted += new EventHandler<WorkflowEventArgs>(runtime_WorkflowAborted);
            runtime.WorkflowCompleted += new EventHandler<WorkflowCompletedEventArgs>(runtime_WorkflowCompleted);
            runtime.WorkflowTerminated += new EventHandler<WorkflowTerminatedEventArgs>(runtime_WorkflowTerminated);

            //Set up the ManualWorkflowSchedulerService so that we can handle when
            //the workflow instance runs.
            ManualWorkflowSchedulerService schedulerService = new ManualWorkflowSchedulerService();
            runtime.AddService(schedulerService);

            //Start the workflow instance.
            Guid g = Guid.NewGuid();
            WorkflowInstance instance = runtime.CreateWorkflow(workflowType, parameters, g);
            instance.Start();
            schedulerService.RunWorkflow(g);

            Console.WriteLine("AggregateReservations Sending Response...");

            //Return the response
            //See the runtime_WorkflowCompleted, method, below to see where _response is set.
            return _response;
        }

        void runtime_WorkflowTerminated(object sender, WorkflowTerminatedEventArgs e)
        {
            Console.WriteLine(e.Exception);
        }

        void runtime_WorkflowCompleted(object sender, WorkflowCompletedEventArgs e)
        {
            _response = (Operation_1Response)e.OutputParameters["Response"];
            Console.WriteLine("Workflow completed");
        }

        void runtime_WorkflowAborted(object sender, WorkflowEventArgs e)
        {
            Console.WriteLine("Workflow Aborted.");
        }

        void runtime_ServicesExceptionNotHandled(object sender, ServicesExceptionNotHandledEventArgs e)
        {
            Console.WriteLine(e.Exception);
        }

        #endregion
    }
}

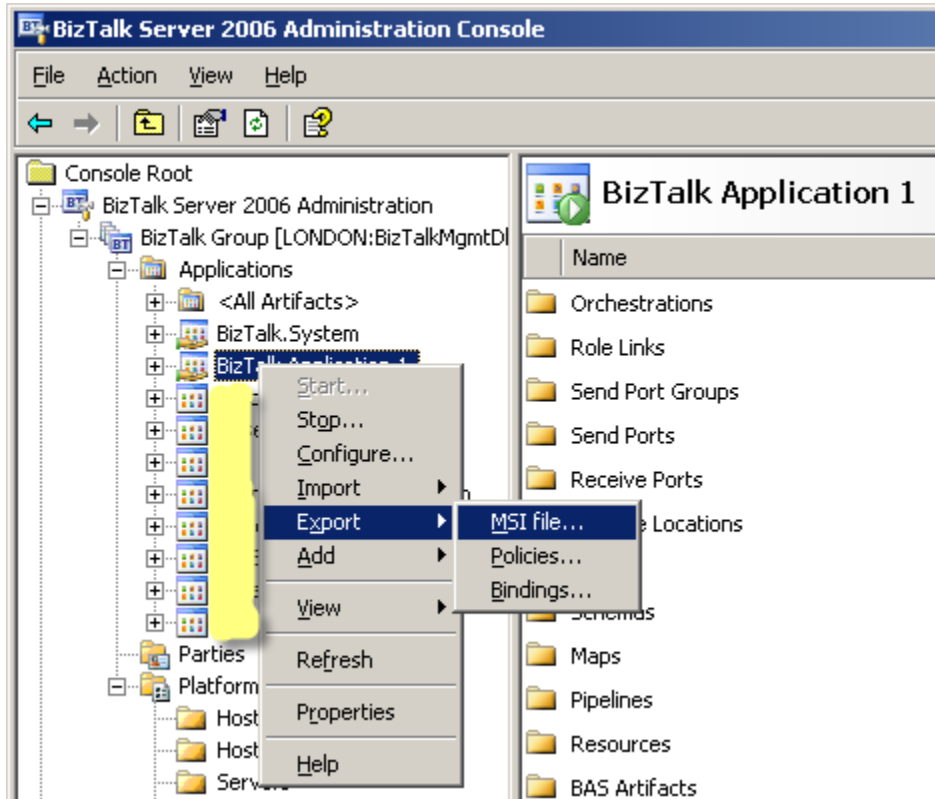
```

Deployment

Deploying the BizTalk solution was accomplished using Visual Studio 2005. When the BizTalk developer tools are installed, several BizTalk Add-Ins are installed for Visual Studio. The BizTalk Add-Ins include facilities for deploying to the local BizTalk environment. Within BizTalk, you must deploy your solutions before you can test and run them.

Note that BizTalk also includes facilities for deploying to other environments (QA, Production, etc.) that are related to the new Application (logical grouping of BizTalk artifacts) concept in BizTalk Server 2006.

In the simplest terms, you can generate .msi packages for Applications and use them to install and configure your BizTalk solutions in other BizTalk environments.



For the SOA Roundtable,

We did not need to deploy the WF solution. We executed it using Visual Studio "F5" debugging. So in development, this is simpler situation than within BizTalk; however, to move to a production (or other environment) you would either use the xcopy deployment features in the .NET Framework, or in more complex scenarios, which is likely, you would need to build your own .msi packages using Visual Studio Setup projects or third-party tools.

Continuing to keep score:

- WF can leverage xcopy deployment mechanisms but you must implement your own host
- For BizTalk, you can leverage the out-of-the-box host concept and you must use the BizTalk deployment mechanisms.

6. Review real world impacts to the new service

Tracking

Tracking is all about gathering technical data about your process. It is essentially a technical audit trail for a process that includes a processes exact flow and path, timestamps, messages bodies and their context, etc. Tracking data can be used for auditing, debugging, performance testing and for other

technical reasons. Imagine running your Hotrod on your local racetrack before your big race. During your practice runs you'd gather as much data as you could. After your practice, you'd pore over the data to learn as much as you could about how your car was running and why. This is analogous to the tracking concept.

WF includes tracking infrastructure but there is some work you need to do to be able to use out-of-the-box tracking infrastructure. The first thing you need to do is create the WF tracking database within SQL Server (2000 or 2005.) The .NET 3.0 SDK includes a script that creates the WF tracking database. The second thing you need to do is to configure your WF host to use the `SqlTrackingService`. You can do this using a `.config` file (web or app as appropriate) or through code. The option to do things via configuration exists throughout WF and is one of its best features; unfortunately, there aren't any tools for viewing or analyzing the tracking data you collect. Finally, you may want to create a Tracking Profile, which is basic a filter so that you only track details that you want to be tracked. We did not leverage the WF tracking infrastructure in our demo WF solution.

You can extend this WF tracking infrastructure to meet your solution's specific needs. You can define tracking profiles that state exactly what you want tracked (and with more granularity than you can with BizTalk.) You can also build your own tracking services to do things such as write to a file instead of SQL Server. Another, real world example of a custom tracking services is to track before and after snapshots of objects that are passed into the WF Rule Engine which is discussed in this blog post:

<http://blogs.msdn.com/skaufman/archive/2006/08/22/Windows-Workflow-Tracking-and-the-TrackingExtract-functionality.aspx>. For more information on extending the tracking infrastructure please reference the WF documentation. Also, there is a sample `ConsoleTrackingService` is included as a sample custom WF tracking service.

BizTalk also includes tracking infrastructure. Health and Activity Tracking is an out-of-the-box feature that is configurable but not directly extensible. You have the ability to specify when you want to track events that happen in BizTalk, such when a Receive Location receives a message as well as if you want to track the actual message bodies. We utilized the tracking infrastructure in our solution in order to show how it runs; asynchronously, in the background and in a configurable manner.

BizTalk also includes a tool for viewing the tracking data, called Health and Activity Tracking (HAT), which is also used to debug orchestrations. HAT is essentially a simple query interface that sits over the BizTalk Tracking and Message Box databases. A Tracking Analysis Services database is also included with BizTalk although that feature was deprecated in BizTalk 2006 and can only be leveraged if the BizTalk databases are running in SQL 2000.

BizTalk also includes Business Activity Monitoring (BAM) features that allow you to instrument your own business process if the out-of-the-box Health and Activity Tracking infrastructure doesn't meet your needs; you can use BAM to capture exactly what you need. You can easily instrument your BizTalk applications using the Tracking Profile Editor which allows you to visually identify (no code!) what information you want to intercept from your messaging and orchestration components. In addition to being able to capture the data the BAM features in BizTalk include a BAM Portal, which is an ASP.NET

2.0 web site that can be used to view your BAM data and create alerts from the data to notify you when key events occur.

It is worth noting that BizTalk Server 2006 R2 will ship with two new interceptors that will enable us to intercept information from WCF services and WF workflows through an XML configuration mechanism.

Persistence

Persistence is used in long running business processes to provide a more scalable and performant runtime. In a very simple and generic sense, it simply means saving your process' state to a physical storage; however, a discussion of persistence also needs to include rehydration (passivation in WF terminology.) Rehydration refers to restarting your previously persisted process. A discussion of persistence is critical to any business process because it will affect performance, scalability, maximum sustainable throughput and how you handle resuming your process when errors occur.

In BizTalk you don't have to implement persistence. It is handled for you by the Messaging and Orchestration Engines. BizTalk includes a MessageBox database, where all messages that flow through BizTalk are temporarily stored. The Messaging and Orchestration Engines communicate with the Message Box database. When a Receive Port finishes receiving a batch of messages, it persists that message batch to the Message Box before any further processing occurs. Orchestrations also save their messages, and also their overall state, to the Message Box database at specific points in their execution. The specific points where Orchestrations persist their state are documented in the BizTalk documentation. In almost all cases in BizTalk, if an orchestration encounters an error, it can be resumed from its last persistence point. BizTalk includes resume functionality the BizTalk Server Administration tool or through APIs.

For the purposes of the airline solution, persistence wasn't actively used or implemented because it is an out-of-the-box, "always-on" feature; however, we did implement the solution to optimize the persistence activities within the orchestration to reduce latency and improve performance. This approach allowed us to have fine grained services call in parallel (if you take a look at the orchestration screenshot in this article you'll note that we send the three requests in a scope which was marked as "atomic". This is difficult to do in WF if you don't want to write threading code.

WF includes persistence services that are in some ways similar to the persistence functionality provided in BizTalk but persistence is not enabled by default – you must configure your WF host to use a persistence service, such as the out-of-the-box `SqlWorkflowPersistenceService`, before you will have persistence functionality. You can configure your WF host to use a persistence service using a .config or code. You also have to create the WF persistence database, which can be created using a script included in the .NET 3.0 SDK.

In WF, persistence happens automatically for you, at the conclusion of some Activities' execution; however, you also have explicit control over persistence using the WF APIs.

The screenshot shows the Business Activity Monitoring (BAM) interface. The title bar indicates 'Business Activity Monitoring' and 'Microsoft BizTalk Server 2006'. The main window title is 'Business Activity Monitoring' and the subtitle is 'Activity Search: SOA Benchmark'. The interface includes a 'My Views' sidebar on the left, a 'Query' section with a search criteria field (Cheap Flights Response Time < Less than 1 Day), a 'Column Chooser' section with a list of available data and milestones, and a 'Results' section displaying a table of data.

Cheap Flights Request	Cheap Flights Response Time (Day)	Cheap Flights Response	CustomerID	Destination City	Destination State	Flight Request End	Flight
9/12/2006 9:10:21 PM	3.90432105632499E-05	9/12/2006 9:10:24 PM	9:10:20 PM	SEATAC	MSP	9/12/2006 9:10:25 PM	9/12/2
9/12/2006 9:10:28 PM	9.3749986262992E-06	9/12/2006 9:10:29 PM	9:10:28 PM	SEATAC	MSP	9/12/2006 9:10:29 PM	9/12/2
9/12/2006 9:10:32 PM	1.18441312224604E-05	9/12/2006 9:10:33 PM	9:10:31 PM	SEATAC	MSP	9/12/2006 9:10:33 PM	9/12/2
9/12/2006 9:10:35 PM	1.20370386866853E-05	9/12/2006 9:10:36 PM	9:10:35 PM	SEATAC	MSP	9/12/2006 9:10:36 PM	9/12/2
9/12/2006 9:11:45 PM	2.42283931584097E-05	9/12/2006 9:11:47 PM	9:11:44 PM	SEATAC	MSP	9/12/2006 9:11:47 PM	9/12/2
9/12/2006 9:11:50 PM	1.49691331898794E-05	9/12/2006 9:11:51 PM	9:11:49 PM	SEATAC	MSP	9/12/2006 9:11:51 PM	9/12/2
9/12/2006 9:11:53 PM	1.48533945321105E-05	9/12/2006 9:11:54 PM	9:11:52 PM	SEATAC	MSP	9/12/2006 9:11:54 PM	9/12/2
9/12/2006 9:11:56 PM	1.97145054698922E-05	9/12/2006 9:11:58 PM	9:11:56 PM	SEATAC	MSP	9/12/2006 9:11:58 PM	9/12/2
9/12/2006 9:12:28 PM	1.23071004054509E-05	9/12/2006 9:12:29 PM	9:12:28 PM	SEATAC	MSP	9/12/2006 9:12:29 PM	9/12/2
9/12/2006 9:12:30 PM	1.56250025611371E-05	9/12/2006 9:12:32 PM	9:12:30 PM	SEATAC	MSP	9/12/2006 9:12:32 PM	9/12/2

Maintenance

After you deploy your BizTalk solution, you will have likely generated all of the artifacts (deployment packages, documents, processes, etc.) that you'll need to perform maintenance releases. That isn't to say that you don't have any work to do when defects are found or enhancements are made... but you shouldn't need to build out your own software infrastructure pieces to support the maintenance work. Additionally, with BizTalk, you can swap out endpoints, services, vendors etc. with relatively little work because of its loose coupling. Both of these facets make BizTalk a very nice tool to have in place.

With WF, you have to build out your own software infrastructure for deployment and maintenance. This isn't a bad thing, but you do need to spend time on these things or your maintenance work could be very time consuming because you may end up troubleshooting or attempting to remember your deployment processes and techniques during critical down times. WF also doesn't have loose coupling to the degree that BizTalk does so in comparison, swapping out endpoints, services and vendors will be more difficult, unless you've spent the time on implementing loose coupling during your design and development.

Communication

For our solutions, the aggregation service needed to communicate with two airline systems using web services and one using MSMQ. The solutions used either an Orchestration or a Workflow to "orchestrate" the calls to the airlines' systems' services.

WCF

In our solution, none of the three airlines services were built using WCF but I wanted to cover the options for exposing and communicating with WCF anyway for all of the gearheads out there. As mentioned, above, we did use a WCF client proxy to call one of the ticket services implemented in .asmx.

The current version of BizTalk, BizTalk Server 2006 the story for communicating with WCF services is not there yet – we’re still waiting on that last engine component before we can fire it up. BizTalk Server 2006 R2, scheduled for release in the 3rd quarter of 2007 will rectify this problem and provide full support for communicating with WCF endpoints and exposing solutions as WCF endpoints using WCF Adapters. There will be several out of the box WCF Adapters in R2 including:

- WCF-BasicHttp
- WCF-WSHttp
- WCF-NetTcp
- WCF-NetMsmq
- WCF-NetNamedPipe
- WCF-Custom
- WCF-CustomIsolated

For WCF support today, you could build a custom BizTalk Adapter to communicate with your specific WCF endpoint. There is also a publically available (Codeplex) WCF adapter available and a third party WSE 3.0 adapter which is wire-level compatible with WCF.

Perhaps more surprisingly, since they were both released together as part of the .NET Framework 3.0, support between WF and WCF is also lacking. WF will have more integration with WCF when Visual Studio “Orcas” is released. For now, when you utilize the two together, you’ll need to manually created proxies, using the same techniques outlined in this article, and use them from a CodeActivity or create your own custom activities that support WCF interaction.

MSMQ

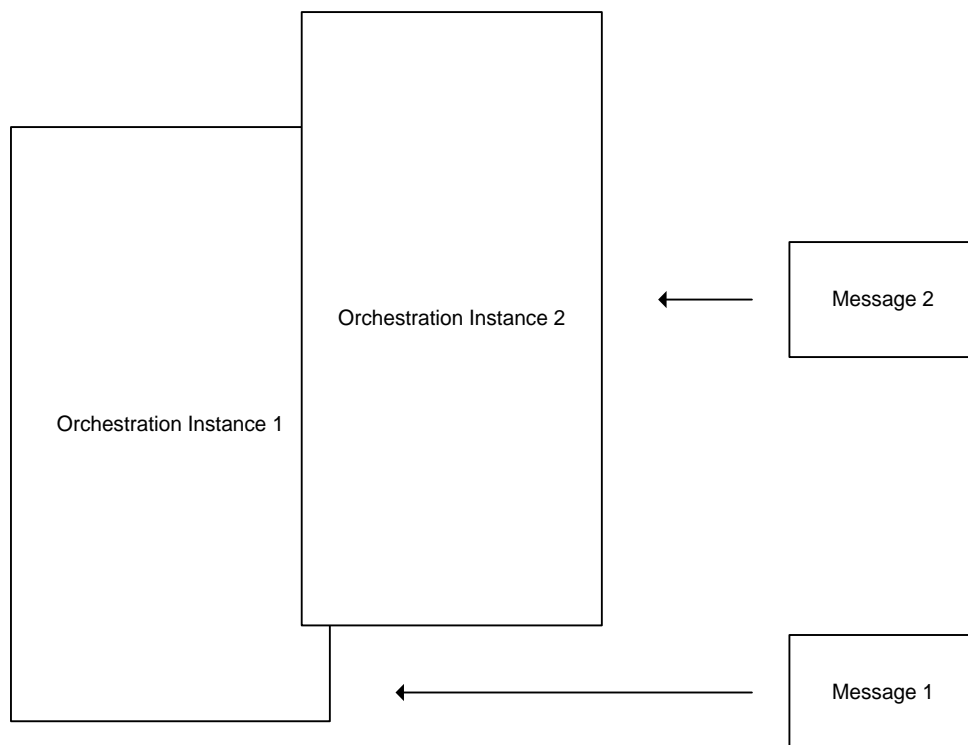
One of the airline ticket services was exposed through MSMQ. Our intent with the WF solution was to utilize the WCF netMSMQBinding; however, due to time constraints, we were unable to complete this. Instead, we implemented a code solution, called from a CodeActivity within the Workflow that wrote a message to a MSMQ queue and waited for a response using the System.Messaging namespace API. We leveraged an xsd.exe generated class type to create an object which we sent to a queue using a MessageQueue class its Send method. We then attempted to receive a message from a response queue using another instance of the MessageQueue class and its Receive method. We did provide some level of abstraction for the two queues that we used by leveraging System.Configuration.ConfigurationManager to abstract the queue paths from being hard coded into our solution. Correlation was also implemented in the code so that we made sure we received the correct response message in the response queue. However, correlation wasn’t implemented in a robust manner in this solution; the receipt code simply looped and received messages until the correct

message was received from the response queue. If the response message wasn't the correct message, based on the correlation id, we threw the message away and continued looping.

It is worth noting that we considered implementing custom MSMQ Activities for our WF solution. We also examined the MSMQ activities on <http://wf.netfx3.com>. We quickly determined that either of these options would be too much effort given our timelines due to the complexity involved around creating local communication services for WF and in understanding the activities. A WF local communication service is essentially a class that you develop to handle inbound and outbound communication for your Workflow. You implement method calls and event handlers on the local communication service to expose your inbound and outbound communication. The complexity is related to the fact that a local communication service is a singleton and it must handle communication to/from multiple workflow instances which basically means you must implement a router. It should be noted that with WF you can also interact with other workflows using Web Services which eliminates the need for implementing a local communication service.

In the BizTalk solution, a logical Port was created in the Orchestration, and then the solution was configured to use a Send Port and Receive Port that used the out of the Box MSMQ adapter. In this solution, we defined a Correlation Set on the logical Port which makes sure that the correct response comes back to the correct orchestration instance from the airline system using MSMQ. This pushes the burden of routing onto BizTalk instead of handling this manually in code. BizTalk guarantees that the right message gets to the right service instance.

Note that correlation is handled automatically by BizTalk for the two web services calls (the logical Send Ports that were used are called solicit-response ports in BizTalk terminology); correlation sets are built for us on-the-fly, inside the BizTalk messaging engine. In the MSMQ case, we used a one-way Send Port and a one-way Receive port because MSMQ doesn't natively support the request-reply pattern.



Since we're keeping score:

- BizTalk has many tools for monitoring, administration and you can also use the standard Windows tools.
- With WF, you need to build your own tools and leverage the standard Windows tools.

Observations

I've had the pleasure of working with BizTalk almost exclusively for the last few years. My largest time away from it was when I was building a client-side rules engine solution using WF for part of last year. I've had ups and downs with both technologies and have learned a few things along the way.

BizTalk is big, it has tons of functionality and it takes a while to come up to speed on. It is a framework for integration and business process automation that sits on top of the .NET Framework. It excels in situations where you need to connect and integrate various systems, using various communication methods and protocols. You want to use it when you can't lose messages and when you have business processes that might take minutes or months to complete. You want to use it when you need a complete audit trail for regulatory compliance. You want to use it when your production support engineers need a standard mechanism for deploying, maintaining and operationalizing your solutions. You want to use it when you need to surface information about those business processes to operational analysts and business sponsors.

You don't want to use BizTalk if you need to connect one simple system to another simple system. You want to use it when you need to implement workflow between applications and systems. You don't want to use BizTalk if you just need a mapping tool.

BizTalk has tons of features and you should be using most of them on the solutions you build with it.

WF is also big but not on the same scale as BizTalk. It is part of the .NET Framework. It doesn't have all of the bells and whistles of BizTalk and its feature set is rudimentary compared to BizTalk. It is used as a foundation for current and future products, both from Microsoft and third party vendors. You want to use WF when you want absolute control over what you're building so that you can control the lower level aspects of your Hotrod.

You want to use it when you need to dynamically generate custom workflows on the fly at runtime. You want to use it when you are building a solution where you want users to be able to create their own custom workflows within the solution. You want to use it if you're building a commercial product for the Microsoft platform that uses workflow (I don't mean a WF workflow but human or system workflow in general) capabilities. You want to use it if you're building software factories built on your domain model. You want to use it when you need an incredible amount of control and flexibility within your solution. If you need workflow capabilities within your application, WF is what you should use. If you want an "F5" debugging experience, across your entire solution, you should use WF.

You don't want to use WF if you don't want to write code. Right now, WF solutions will require a large amount of custom coding. You don't want to use WF if you are looking for robust functionality around tracking and persistence data maintenance, rules engine functionality such as versioning and information worker rules creation, or need to connect to various systems, over various protocols.

Remember, WF is a part of the .NET Framework. It's there for you, for free, but you'll have to build on top of it to build your solutions just like you would when using the rest of the .NET Framework.

Conclusion

WF is like a custom built Hotrod. You might build out your car from scratch or you might procure most of the pieces individually or in sets of pieces. In either case, you build out the car, one step at a time, and in some cases you might even build some of the parts you using from scratch. You might even trick it out a bit with some custom art or a slick set of hubs and wheels. After months of hard work, you're finally ready to take it for its first spin. During that first few test runs you figure out that you need to tweak some of the carburetor settings as well as some of the frame. It starts performing better and as you learn more about it, you tweak it some more... and finally its ready for the Smack Down.

BizTalk is like an off the shelf, super slick, high performance racing machine. It has the best available engine, hidden computer chips everywhere, a navigation system, superb sound system, tinted windows (if you're in to that sort of thing) and a bumper to bumper warranty. It will perform well enough and is flexible enough for most racing situations. The first time you get behind the wheel you probably don't know what all of the buttons and knobs do. After a while you figure it out and get a feel for it. But it

needs to be tuned a bit for your specific racing scenario. You tweak it a bit and then you're ready for the Smack Down.

A special thanks to Lee Monson, Jim Gaudette and Todd Van Nurden for their work on the SOA Roundtable event as well as their help on this article. Without their hard work and support, this article wouldn't have been possible!

BizTalk vs. WCF Sidebar:

I'm often asked why organizations should consider using BizTalk now that WCF has been released. The truth of the matter is that these are not competing technologies but rather complementary technologies that serve different purposes but that can and should be leveraged together with BizTalk Server 2006 R2's release later this year. I also think it is worth noting that the two technologies both reside within Microsoft's Connected Services Division; you can expect more collaboration between these teams and technologies in the coming years.

WCF is a .NET Framework component that provides us with a mechanism for communicating across physical processes. It allows us to do this using different communication protocols and transports as well as extend the protocols and transports that it supports. It unifies web services, MSMQ, distributed transactions, and WS-* support in a single programming model. You use WCF to provide services to your organization in an elegant and flexible manner. You can use WCF to implement point to point interfaces between systems. You could use WCF as a basis for building something like BizTalk but you probably don't have the thousands of person years available to do this.

BizTalk is a framework for integration and business process automation that sits on top of the .NET Framework - and it will sit on top of WCF later this year. BizTalk is used to build composite, loosely coupled applications on top of the services that WCF exposes. It provides things such as state management, compensation, access to various legacy protocols and applications and an administration model for operationalizing the applications. BizTalk is a broker that sits between various systems in order to abstract them from one another; it decouples them and eliminates point to point interfaces.

Both have value and will be used going forward.

BizTalk	WCF
State Management	
Rules Engine	
Administration Model	
Legacy application access	
Enterprise Single Sign-On	WS-* Support

Service Publishing	Service Publishing
Implement using Visual Studio designers and coding	Implement by code

Appendix: SOA Roundtable Technology Comparison

Task	.Net 2.0	.Net 3.0	BizTalk
Create web service to accepts requests	<ul style="list-style-type: none"> • New ASPX Project • Create Request and Response Schema • generate serializeable class representation of schemas • Attribute class to ensure document centric (Non RPC) WSDL 	<ul style="list-style-type: none"> • Generate data contract objects with SvcUtil • Add endpoint configuration XML • Create WCF host 	<ul style="list-style-type: none"> • Create request and response schemas • Use web service publishing wizard to publish schemas as a web service
Add artifacts for external web services	<ul style="list-style-type: none"> • Add web reference to project, maintaining URL in App.config • Call generated proxy in code 	<ul style="list-style-type: none"> • Add WCF endpoint binding to Config • Generate client proxy data contract • Call endpoint in code 	<ul style="list-style-type: none"> • Add schemas from web reference • Create send port in Admin console.
Add artifacts for Async communication with MSMQ	<ul style="list-style-type: none"> • Use messaging namespace to send request • Create separate queue listener • Correlate responses to request threads via <ul style="list-style-type: none"> ○ Events ○ custom DB ○ Enterprise caching library 	<ul style="list-style-type: none"> • Add netMsmqBinding endpoint binding • Use LocalService Interface to send MsMq message through WF • Raise an ExternalData event obtaining the WorkflowInstanceId of the corresponding request based on <ul style="list-style-type: none"> ○ Custom DB ○ Enterprise caching library 	<ul style="list-style-type: none"> • Add MSMQ send port and receive location • Promote common data in request and response schemas to correlate the response to the request
Create process to call all services	<ul style="list-style-type: none"> • Write some code 	<ul style="list-style-type: none"> • Create work flow • Create Host for WF Runtime • Initialize work flow runtime • create WF instance and start it 	<ul style="list-style-type: none"> • Create an orchestration • bind the initial receive port to the webservice receive location
Create External	<ul style="list-style-type: none"> • Code request object 	<ul style="list-style-type: none"> • Code request object 	<ul style="list-style-type: none"> • Create the transforms

service requests based on canonical request	<p>creation</p> <ul style="list-style-type: none"> • Use XMLSerialization and XSLT 	<p>creation or</p> <ul style="list-style-type: none"> • Use XMLSerialization and XSLT 	<p>graphically</p>
Determine the best flight from all responses	<ul style="list-style-type: none"> • Custom component with all logic 	<ul style="list-style-type: none"> • Custom object to hold responses • Use Policy activity in the work flow 	<ul style="list-style-type: none"> • Call a business rules policy using the messages already defined in the orchestration
Scale for large throughput	<ul style="list-style-type: none"> • Define and implement a persistence architecture to correlate the Msmq responses • Let IIS scale it for you 	<ul style="list-style-type: none"> • Either use IIS or create a custom threading host for the WCF endpoint • Use WF SQL Workflow Persistence Service 	<ul style="list-style-type: none"> • Use the BizTalk host structure for H/A and scale out
Track all requests and their results	<ul style="list-style-type: none"> • Add tracing code • Custom store • reporting 	<ul style="list-style-type: none"> • Use WF tracking API • Custom store • Reporting 	<ul style="list-style-type: none"> • Add a BAM activity • Create graphical tracking profile • use the built in BAM portal