Table of Contents

1 BizTalk Overview

BizTalk is a *Business Process Management; Service Oriented Architecture (SOA),* or an *Enterprise Services Bus (ESB)* platform. You can think of *BizTalk* as a group of tools and application services that facilitate the rapid creation of integration solutions. Most enterprises are looking for *BizTalk* to enable unrelated and disconnected systems to exchange data in a standard, consistent, and reliable way. The tools that *BizTalk* provides allow you to design reliable and robust solutions faster than is often achievable by custom coding a solution from scratch using standard development languages and tools. With this in mind, it is important to understand what *BizTalk* achieves well and what is does not in order to fully realize any efficiency what a *BizTalk* solution can bring.

If you were to look at two *BizTalk* projects that address identical problems, designed by two different groups, you would see two completely different solutions. Both solutions would solve the problem they were designed to address; however, you will find that one solution utilizes more *BizTalk* tools. This solution is the better one because it will be more robust and easier to maintain. Think of *BizTalk* as a toolkit, and within this toolkit, you find tools to help you build your application.

1.1 Why BizTalk?

For those of you who are completely new to *Enterprise Application Integration (EAI)* and *Business Process Management (BPM),* perhaps some context can clarify the ideas behind messaging and orchestration. In addition, this section will attempt to clarify the

concept of “messaging”. If you’re familiar with some basic EAI principles like loose coupling, publish /subscribe and orchestrations, you may safely skip to the following section.

If two applications need to establish some form of communication between them, this can be done in many ways. For example, application “A” needs information from application “B”, which was developed prior to “A”. This means “B” isn’t even aware of “A”’s existence. Luckily, the developers of “B” have exposed an API to integrate with. Application “A” could thus call “B”’s API to gather the information it needs. No problem, works fine, life’s easy isn’t it? Not really, integrating applications is usually about making assumptions. And in the case of our example: we’ve made many assumptions, such as:

- “B” is running while “A” makes the call.
- Communication links between “B” box and “A” box are stable, secure, well established, etc.
- “B” runs on a similar platform as “A” does, i.e. same data representation.
- “A” is programmed in a language that can consume “B”’s API.
- “A” will never need another application to provide the information.

Now, imagine that you have four or six applications that you need to integrate. Even if you only have two applications as above imagine following scenarios for a moment:

- What if we wanted to replace “B” by another, better application?
- What if we wanted to phase-out “B” and move it to another box or cluster it?
- What if application “B” is down while “A” needs it?

Because “A” makes a few assumptions about “B”, your integration solution might run into serious trouble. Loose coupling of applications is exactly about preventing these kinds of problems. The fewer assumptions an application makes when integrating with another application, the more loosely coupled they are.

Messaging is a very popular way to reduce coupling between applications. Messaging relies on middle-ware to transmit chunks of data (“messages”) from one application to another. Usually this is done using a kind of “channel” called “message queues”. The nice thing about queuing is that it works asynchronously. This means that the sending application can put messages in the queue while the receiving application can pull the messages out of the queue as soon as it has time for it. For example, when it has been down and brought up again. Obviously, not all applications know how to integrate with queues. In most scenarios there will be some “adapter” converting the application’s output or input, to or from messages. Messaging reduces coupling and makes solutions more scalable. Because of the queue acts as a load buffer, each application can work at its own speed. When you integrate application “A” with Application “B” directly (even through queues) this is referred to Point to Point integration. Now if you have 5 or 6 applications that you need to integrate. Point to point would lead to many complications and intertwining between how the applications communicate (or cooperate together) and leads to a case of spaghetti, shown in figure 1.1. In real life, most organizations rely on many software applications and they need to integrate them together. This “spaghetti” kind of integration is likely to become unmanageable. Having a lot of point to point connections acting separately from each other is not only unmanageable; it also makes “business overviews” impossible. What do I mean with “business overviews” is, if you succeeded in connecting each and every system in an insurance organization

claim processing systems, you'd expect to be able to make queries like: "How many claims were approved today?".

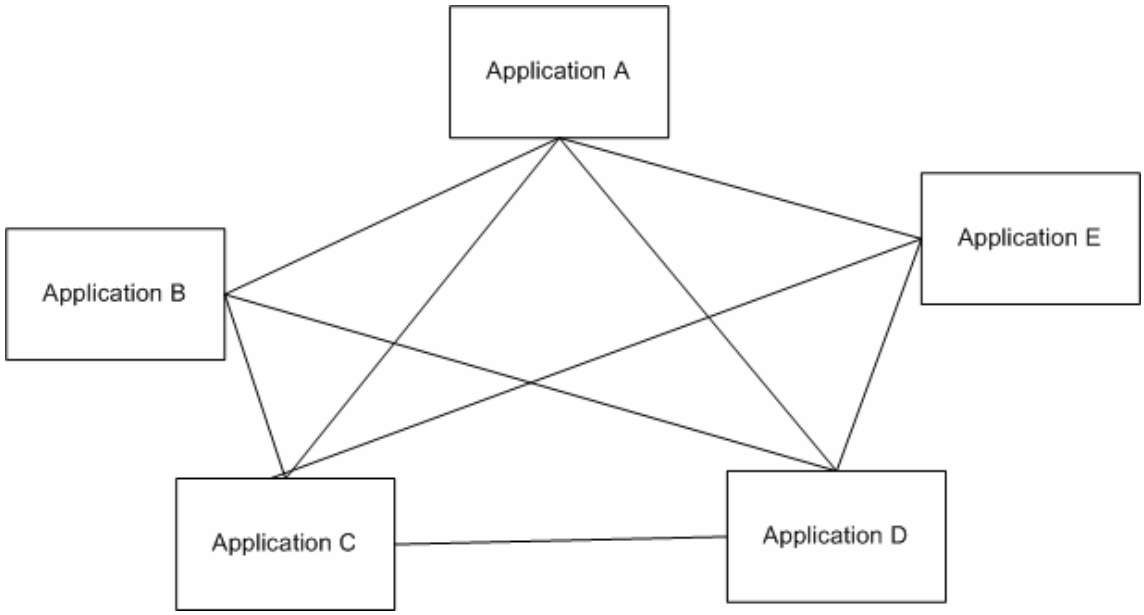

Figure 1.1: Spaghetti Application Integration.

This requires some "business overview", exceeding point to point communications. To overcome the problems related to having point to point connections, Hub and Spoke Integration services was invented, shown in figure 1.2.

Architecturally, a Hub and Spoke Integration services are centralized processing machinery "the hub", that accept messages from multiple applications, the Spokes. Applications interact with the hub through adapters deployed on the hub, so the spokes (applications) do not require any modifications. The hub manages routing, mapping and tracking messages between applications. The hub and spoke mode provide strong total

cost of ownership by reducing the cost to add and remove connections and through centralized management.

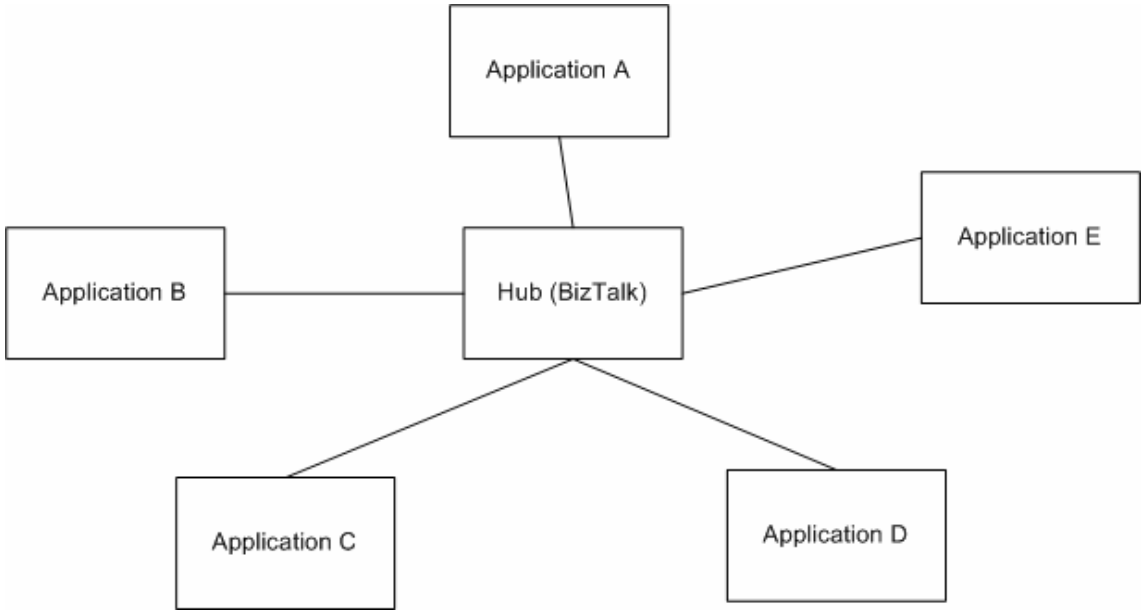

Figure 1.2: Hub and Spoke.

The hub and spoke integration services lead to the "publish/subscribe" pattern. In the publish/subscribe pattern, two parties communicate without any shared knowledge other than the agreement on the message format. These two parties are often referred to as the producer and the consumer or – more appropriately in the context of publish subscribe – the "publisher" and the "subscriber". The publisher publishes its output using messaging to the pub/sub engine while the subscriber subscribes itself with the engine using a filter. The filter, based upon message content criteria, specifies which messages are of interest to the subscriber.

BizTalk is Microsoft implementation of the Hub and Spoke architecture. BizTalk is the Hub that receives messages from the spokes and routes these messages to their destinations.

1.2 How BizTalk Works?

BizTalk is designed to receive *inbound messages*, pass them through some form of *logical processing*, and then deliver the result of that processing to an *outbound location* or subsystem. BizTalk views all data and events as "Messages". A *message* is a finite entity within the BizTalk *MessageBox*. Messages are immutable once they are published to the MessageBox. Messages have context properties and zero-to-many message parts. BizTalk extends the idea of *TCP/IP Port* to include other protocols such as *file, SQL* , etc. A Port is either a *"receive location"* (i.e. BizTalk would listen on this port for incoming messages) or a *send port* that BizTalk will send through messages. BizTalk groups receive locations into what it calls a *"receive port"* and groups send ports into *send port group* (I know the naming is a little bit confusing). A *receive location* receives messages using a component called *"BizTalk Adapter"*. A *BizTalk Adapter* is a component that understands the *Transport Protocol* like *HTTP, FTP* etc. The received message is processed by a *pipeline* associated with the receive location. A pipeline is a component that performs transformations on messages to prepare them to enter or leave *BizTalk Server MessageBox database*. In the receive locations, the main task of a receive pipeline is to convert input data into proper *XML*. Likewise, in send ports, the main task of a send pipeline is to convert input data into the proper data format for the transport protocol (Adapter).

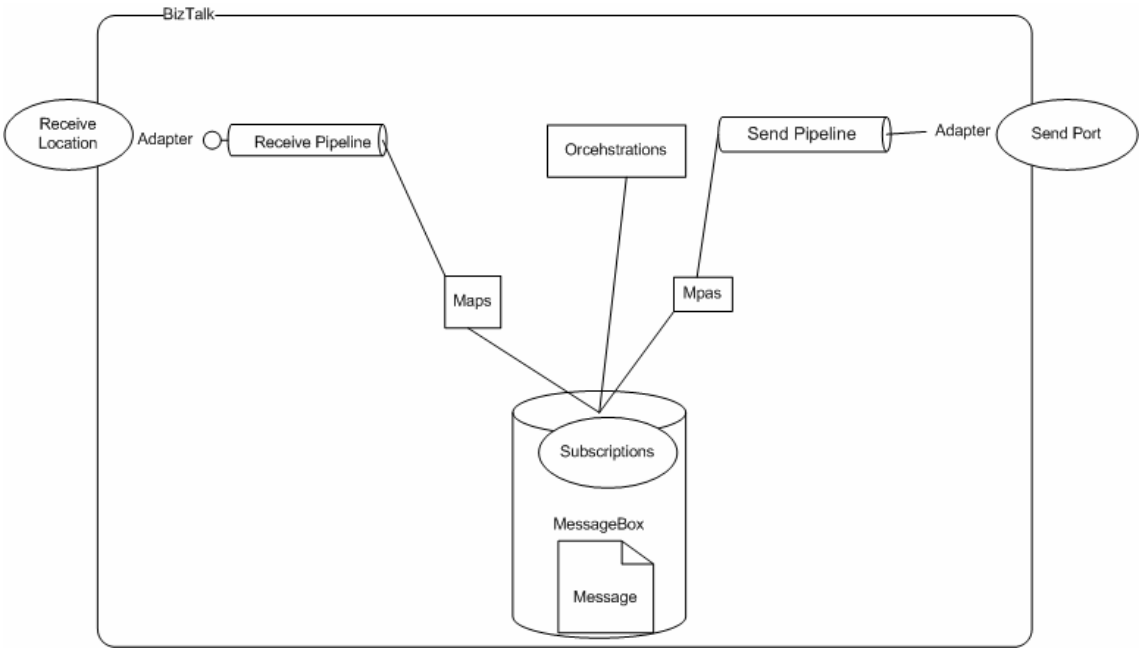

Figure 1.3: BizTalk Functional view.

A receive or send port can have one or more maps associated with it. A *Map* is a component that transforms a message from *one XML schema* to another *XML Schema*. The resulting *message* is then published to the MessageBox database. The *MessageBox* evaluates active *subscriptions* and *routes* the message to those *orchestrations*, and send ports with matching subscriptions. A *subscription* is a collection of comparison statements, known as predicates, involving message "*context" properties* and the values specific to the subscription. *Subscriptions* match particular context properties for a message and determine endpoints (orchestrations, or Send Ports), which are interested in processing it. *Orchestrations* may process the message and publish messages through the *MessageBox* to a send port where it is pushed out to its final destination. An *Orchestration* is an executable business process (workflow) that can subscribe to (receive) and publish (send) messages through the MessageBox database.

The design of a *BizTalk* solution begins with selecting which of the BizTalk tools to use for each of these three simple yet all-important tasks (receive, process, and send a message). I'll introduce these tools in this chapter and continue to explore them further throughout the rest of the book. First, let's start with a simpler overview:

- *Messaging* is a component that provides the ability to communicate with a range of other software. *BizTalk* introduces the concept of *Ports and adapters. Ports and adapters* provide the logical abstraction for sending and receiving messages to and from BizTalk. They allow you to code your application in a generic fashion and not worry about the implementation details of how these messages will be consumed and delivered. A port is a logical construct that can receive and send messages to/from the BizTalk MessageBox. A receive port must be associated with a specific receive location to accept the message. The *receive location* is tied to a specific adapter, which provides the implementations of the Transport protocol, and a pipeline. A *Pipeline provides* a way to examine, verify, and potentially modify messages as they are received and sent out from BizTalk. I will discuss Messaging in more details in chapters 2, 3, and 4.

- *Orchestration engine* supports the creation and running of graphically-defined processes. Orchestration is the most powerful tool within the BizTalk Server toolbox. Think of orchestrations as a special form of workflows similar to the Windows *Workflows Foundations*. Orchestrations are used to graphically model process workflow and provide the primary mechanism to implement business process automation within BizTalk based project. *Orchestrations* are

created within Visual Studio and compiled into .NET assemblies that are registered in the BizTalk *Management Database*. Assemblies deployed to this database must also be installed into the *Global Assembly Cache (GAC)*. I will discuss Orchestrations in more detail in Chapter 6.

- A *Business Rule Engine* (BRE) that evaluates complex sets of rules. The BRE allows you to model business rules using a simple GUI. It is designed to allow for versioning and modification of implemented business rules without the need to change the processes within BizTalk that uses them. You use the BRE to implement logic that requires frequent updates such as a discount policy percentage or calculations that are updated frequently as a result of legal or government regulations. I will discuss the Business Rule Engine in more detail in Chapter 7.

- The *Health and Activity Tracking (HAT)* tool lets you monitor and manage the engine services. HAT will be presented in more details in section 1.2.3 later in this chapter.

- *Enterprise Single Sign-On (SSO)* facility which provides the ability to map authentication information between Windows and non-Windows systems.

- *Business Activity Monitoring (BAM)* framework you use to build tracking and monitoring into you BizTalk based solutions. Information workers will use it to monitor the state of the running business processes. Business Activity Monitoring (BAM) and Business Activity Services (BAS) provide the infrastructure to perform application instrumentation and metric reporting.

BAM provides the ability to instrument your application to provide business-level data that is much more relevant than the default system-level information which is available in the base product. I will discuss BAM in chapter 9.

- *Business Activity Services (BAS)*, which is used by information workers to set up and manage interactions with transacting partners. BAS provides a simple yet powerful way to display metric data from BAM and other system-level subservices using Microsoft SharePoint Portal Services. Most organizations will integrate the BAS portal into an existing SharePoint infrastructure rather than build an entire SharePoint site around BAS itself.

1.2.1 *How the Publish/Subscribe mechanism Works?*

Understanding the *Publish/Subscribe* mechanism is central to understanding *BizTalk*. The idea of subscription can be:

1. Messages are received (via receive handlers or from orchestration Send ports) and handed to the *BizTalk* engine.
2. The engine stores data about the message called contextual properties.
3. The engine queries a rule store to determine a set of matching subscriptions.
4. For each matched subscription, a record is entered into an application-specific queue, and is associated with a specific instance of a service.
5. Queued messages are then de-queued using a number of separate worker threads, and routed to the designated instances of services.
6. The service (either an orchestration or a send port) handles the message.

In essence, the subscription mechanism acts as a rules engine that infers, from a set of predicate-based rules, which messages should be handed to which service instances.

Subscriptions are defined primarily in terms of message context property predicates. Each predicate describes an expression used in subscription matching. For example, a predicate might define an expression that tests for messages with an orderQty property value that is greater than 3500. *BizTalk* supports the following predicate types:

- Bitwise AND
- Equals
- Exists
- GreaterThanOrEquals.
- GreaterThan.
- LessThanOrEquals.
- LessThan.
- NotEqual.

Predicates are collected into 'And' and 'Or' groups. BizTalk does not allow these groups to be nested, and therefore is somewhat constrained in terms of the subscription rules which potentially could be created. Subscriptions are created by services, and results in messages being routed to orchestrations or send ports.

1.2.2 BizTalk Databases

When you install and configure *BizTalk* it creates several databases in the assigned SQL Server. You do not need to have all these databases it depends on the features that you

configure through the *BizTalk Configuration* that I will discuss in section. The following are mandatory *BizTalk* databases:

- *BizTalk Management database*: This database is the central meta-information store for all instances of *BizTalk Server*.
- *BizTalk MessageBox database*: This database is used by the *BizTalk Server* engine for routing, queuing, instance management, and a variety of other tasks.
- *BizTalk Tracking database*: This database stores health monitoring data tracked by the *BizTalk Server* tracking engine.

- *Rule Engine database*: This database is a repository for:
  - *Policies*, which are sets of related rules.
  - *Vocabularies*, which are collections of user-friendly, domain-specific names for data references in rules.
- *SSO database*: The Enterprise Single Sign-On database securely stores the configuration information for receive locations.

The following are optional databases that based on your configuration BizTalk would create:

- *BAM Analysis*: This database contains Business Activity Monitoring (BAM) OLAP cubes for both online and offline analysis.
- *BAM Archive*: This database archives old business activity data. Create a BAM Archive database to minimize the accumulation of business activity data in the BAM Primary Import database.
- *BAM Notification* Services Application database: This database contains alert information for BAM notifications. For example, when you create an alert using the BAM portal, entries are inserted in the database specifying the conditions

and events to which the alert pertains, as well as other supporting data items for the alert.

- *BAM Notification Services Instance database*: This database contains instance information specifying how the notification services connect to the system that BAM is monitoring
- *BAM Primary Import database*: This is the database where *BAM* collects raw tracking data.
- *BAM Star Schema*: This database contains the staging table, and the measure and dimension tables.
- *Tracking Analysis Server*: This database stores health monitoring online analytical processing (OLAP) cubes.
- *Windows SharePoint Services configuration database*: This database contains all of the global settings for the server.
- *Windows SharePoint Services content database*: This database contains all of the site content, such as list items and documents.

BizTalk Server runtime operations typically use BizTalk Server Management database, MessageBox databases, tracking database, and SSO database. Depending on the BizTalk Server functionality that you use, you may have some or all of the other databases in the table.

1.2.3 BizTalk SQL Jobs

BizTalk uses the following SQL Jobs and you should schedule them to run on the SQL server in order to clean the *BizTalk MessageBox*[1] database:

[1] Note: The names of the jobs change depending on the database names given during configuration. If you have deployed multiple MessageBox databases in your environment, there will be several jobs for each MessageBox.

- *Backup BizTalk Server (BizTalkMgmtDb)*: This job performs full database and log backups of the *BizTalk Server* databases.
- *CleanupBTFExpiredEntriesJob_BizTalkMgmtDb*: This job cleans up expired *BizTalk Framework (BTF)* entries in the *BizTalk Management (BizTalkMgmtDb)* database.
- *MessageBox_DeadProcesses_Cleanup_BizTalkMsgBoxDb*: This job detects when a *BizTalk Server* host instance (*NT service*) has stopped and releases all work that was being done by that host instance so that it can be worked on by another host instance.
- *MessageBox_Message_Cleanup_BizTalkMsgBoxDb*: This job removes all messages that are no longer being referenced by any subscribers in the *BizTalk MessageBox (BizTalkMsgBoxDb)* database tables.
- *MessageBox_Message_ManageRefCountLog_BizTalkMsgBoxDb*: This job manages the reference count logs for messages and determines when a message is no longer referenced by any subscriber.
- *MessageBox_Parts_Cleanup_BizTalkMsgBoxDb*: This job removes all message parts that are no longer being referenced by any messages in the *BizTalk MessageBox (BizTalkMsgBoxDb)* database tables. All messages are made up of one or more message parts, which contain the actual message data.
- *MessageBox_UpdateStats_BizTalkMsgBoxDb*: This job manually updates the statistics for the *BizTalk MessageBox (BizTalkMsgBoxDb)* database.
- *PurgeSubscriptionsJob_BizTalkMsgBoxDb*: This job purges unused subscription predicates from the *BizTalk Server MessageBox (BizTalkMsgBoxDb)* database.

1.2.3.1 How to activate and schedule these jobs?

By default these Jobs are enabled and scheduled to run at Midnight. You might want to change these schedules depending on the schedules you setup for the data conversion processing discussed earlier.

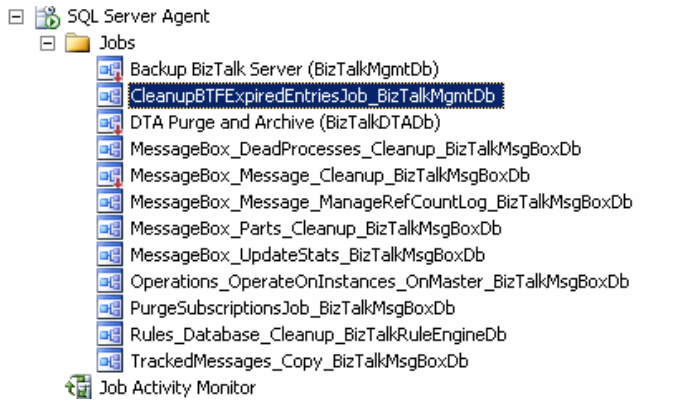

Figure 1.4: SQL Management Studio

Start Microsoft SQL Management Studio select SQL Server Agent and expand Jobs as shown in figure 1.4. You will find the jobs defined there. To change a job schedule right click on the job and select properties, shown in figure 1.5. Select schedules and the select edit to edit the schedule.

1.3 About BizTalk 2009

The new *BizTalk 2009* have many improvements over the *BizTalk 2006 R2*, most importantly is the full support for *Windows 2008*, *Visual Studio 2008* and *Hyper-V* technologies. Updated with *.Net 3.5 framework SP1* and improved *Failover Clustering* that can be deployed in multi-site cluster scenarios, where cluster nodes could reside on separate IP subnets and avoid complicated VLANs. Business Activity Monitoring has

been enhanced to support scalable real-time aggregations. *EDI and AS2* Protocols now support multiple message attachments, configurable auto message resend, end-to-end filename preservation, improved reporting to address new features, and Drummond re-certification for AS2. Complete support for *Application Lifecycle Management (ALM)* with *Microsoft Team Foundation Server (TFS)*.

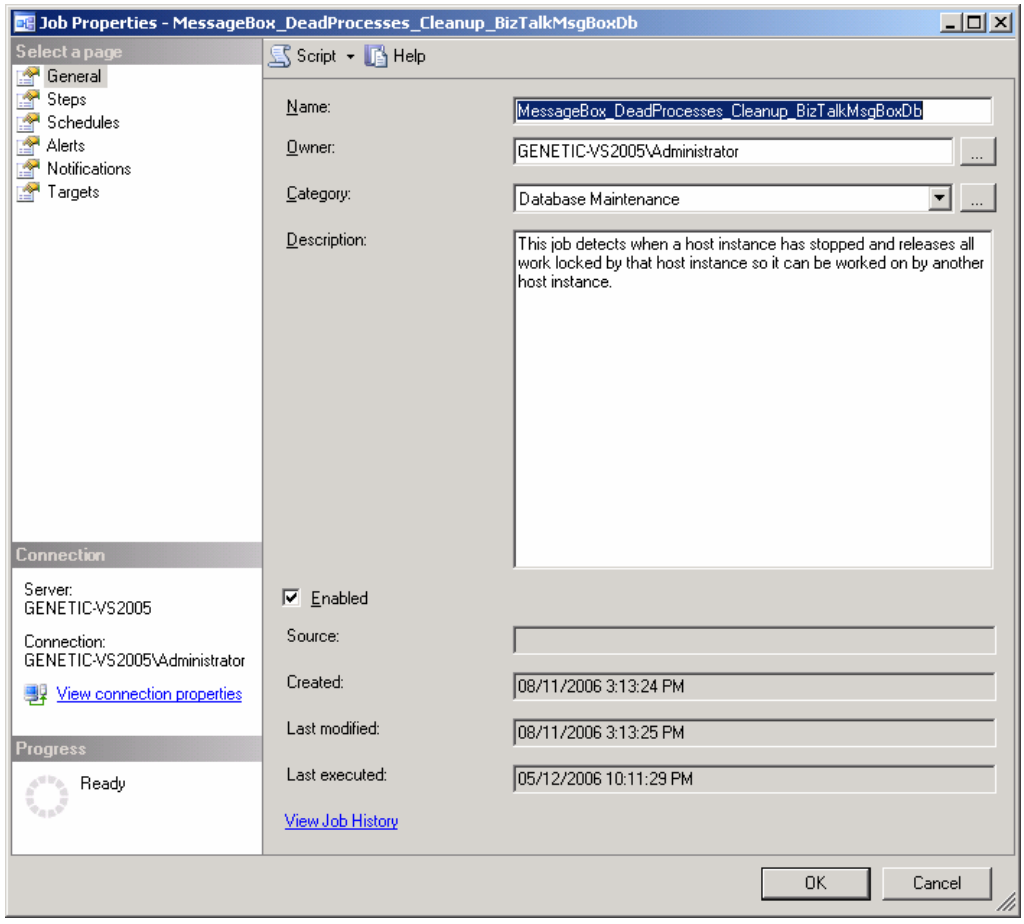

Figure 1.5: SQL Job Properties.

Most importantly for developers is the enhanced debugging support for artifacts such as *BizTalk Maps (XSLT),* pipeline components and *XLANG Orchestrations*, and enables

support for unit testing via *Visual Studio* Test. And the improved recoverable interchange processing of validation failures by providing support for recoverable interchange processing for disassembly (DASM) and a validation stage within the pipeline. Finally, the WCF Adapter is enhanced to provide support for configurable transactions and the ability to choose the transaction isolation level in the WCF-Custom Send Adapter.

1.4 Building the Development environment

1.4.1 BizTalk 2006 R2 Development Environment

The development environment that I am using for the samples accompanying this book for the BizTalk 2006 R2 is based on a virtual machine. You will need to build a similar environment in order for you to follow with the examples and walkthroughs in the book. You will need to install the following before install *BizTalk 2006 R2*.

- Windows 2003 Server.
- Windows Share Point services,
- MS Excel,
- MS SQL Server 2005, and
- Visual Studio 2005.

After running the installation of *BizTalk 2006 R2*, the installation package will start the *BizTalk Configuration* application. You need to configure the *BizTalk 2006 R2* server with the *SQL* server you are using and the users and user groups before you can use it. In Chapter 9, I will discuss the details of the infrastructure for a *BizTalk 2006 R2*

environment. However for running the samples you can use the "default configuration" option in the *BizTalk Configuration* tool. This option assumes that you are using a local *SQL* Server. You can find more information about how to install and configure *BizTalk* on *MSDN*.

After you have installed and configure *BizTalk 2006 R2*, you will be able to see the *BizTalk Administration Tool*, *Health and Activity Tracking Tool*, and *Business Rules Composer* on the *Programs→Microsoft BizTalk 2006 Server Start* menu. You will also notice that *BizTalk Project* appear in *Visual Studio* as an option for creating new projects. Section 1.5 introduces you to the *UI* tools and components that come with *BizTalk Server*.

1.4.2 BizTalk 2009 Development Environment

The development environment that I am using in this book for the BizTalk 2009 Samples is based on a physical windows 2008 environment; however you can use a virtual machine. You will need to build a similar environment in order for you to follow with the examples in the book. You will need to install the following before you install *BizTalk 2009*:

- Windows Share Point services 3.0 with service pack 1,
- MS Excel,
- MS SQL Server 2008, and
- Visual Studio 2008.

Before Installing *BizTalk 2009 Beta* you should first read the Microsoft document "*Installing and Configuring BizTalk Server 2009 (Beta)*". After running the installation of *BizTalk*, the installation package will start the *BizTalk Configuration* application. You

need to configure the *BizTalk* server with the *SQL* server you are using and the users and user groups before you can use it. Figure 1.6 shows the configuration user interface for *BizTalk 2009*.

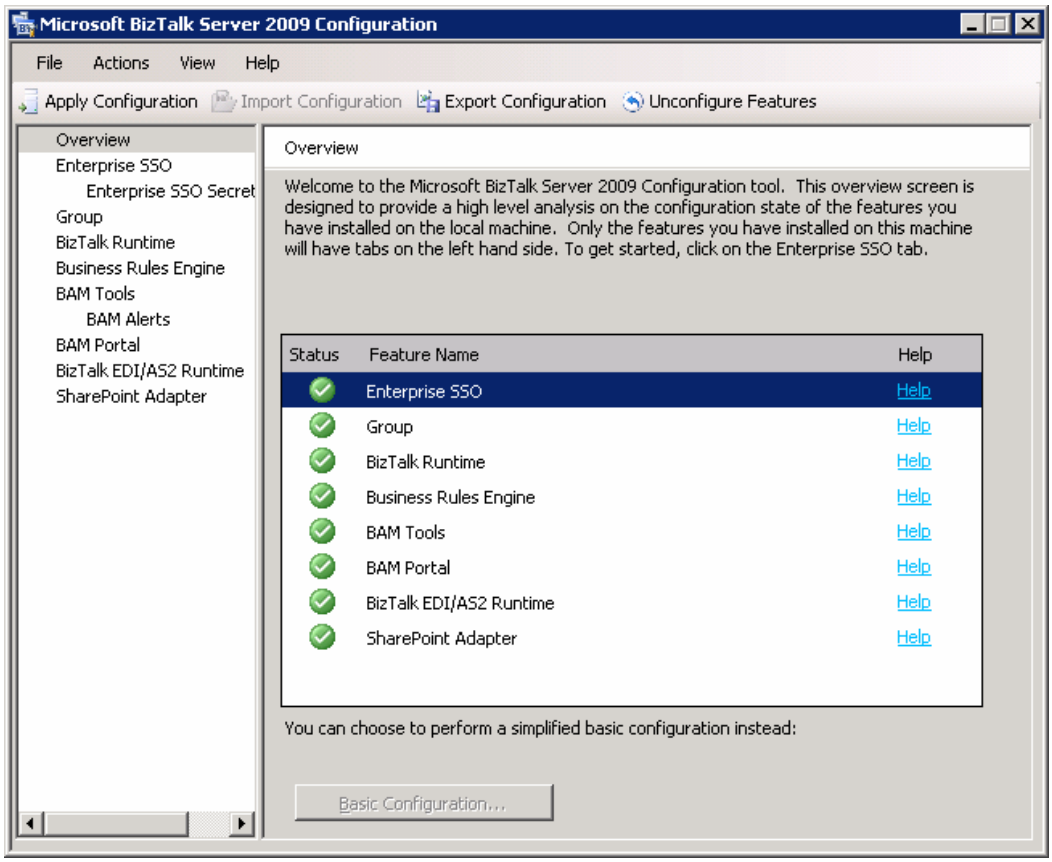

Figure 1.6: BizTalk 2009 Configuration.

After you have installed and configured *BizTalk*, you will be able to see the *BizTalk Administration Tool, Health and Activity Tracking Tool, Business Rules Composer on the Programs→Microsoft BizTalk Server 2009 Start menu*. You will also notice that *BizTalk*

Project appears in *Visual Studio* as an option for creating new projects. Section 1.6 introduces you to the UI tools and components that come with *BizTalk Server*.

1.5 Developing BizTalk Projects Basics

When you install BizTalk on your development environment BizTalk will add several components to Visual Studio. The first thing you should notice is the new BizTalk Projects templates in the *New Project* dialog, shown in figure 1.7.

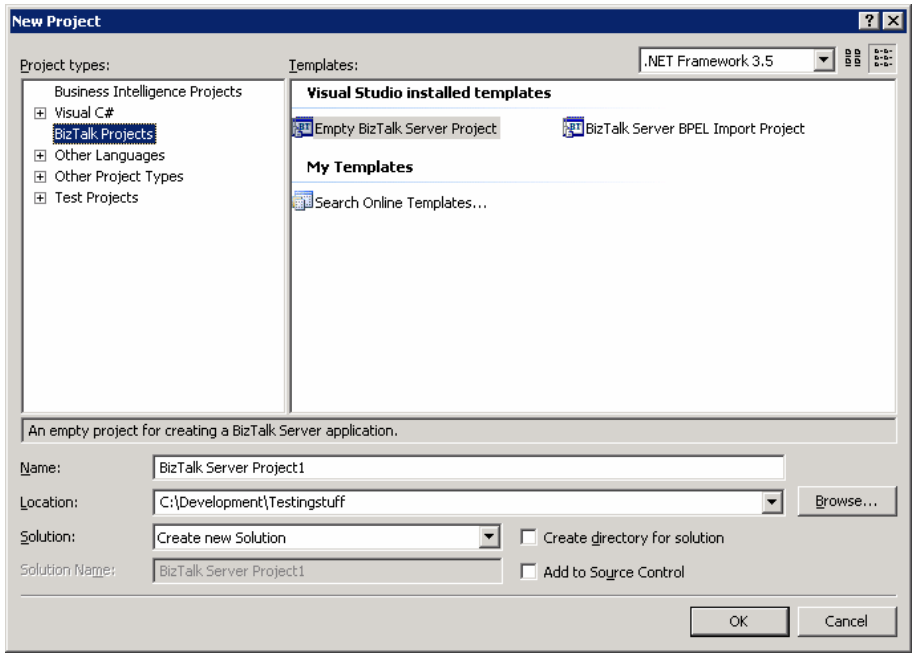

Figure 1.7: BizTalk Projects in Visual Studio.

These templates allow you to quickly create BizTalk projects. The Empty BizTalk Server Project template is the one that you will use the most. Figure 1.6 shows you what the created project would look like. It is a regular .Net class library with reference to a few BizTalk assembles that enables the integration with BizTalk.

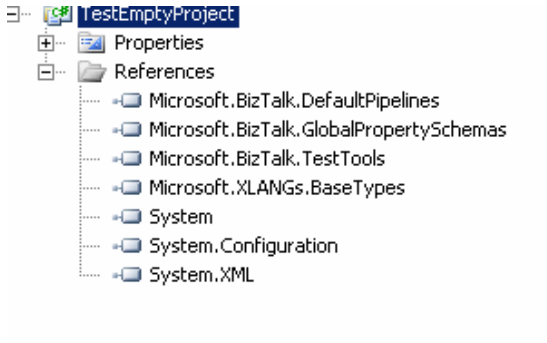

Figure 1.8: Empty BizTalk Project.

In Visual Studio 2008 with BizTalk 2009 the properties are displayed as shown in figure 1.9. These are the same as with *Visual Studio 2005* and *BizTalk 2006* shown in Figure 1.10, the only difference is in the UI look and feel.

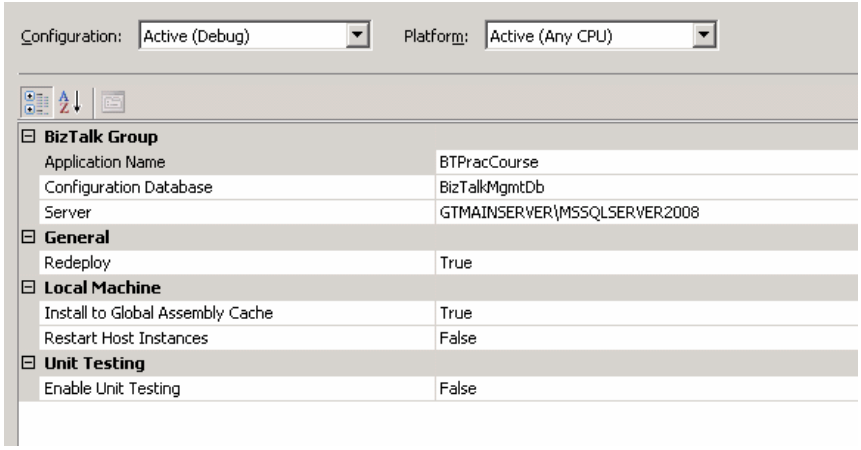

Figure 1.9: BizTalk Project Properties in Visual Studio 2008.

A few details you should be concerned with:

1. *Signing the Assembly*: you should always make sure that any BizTalk project assemblies are signed and any assemblies they reference are signed and deployed to *Global Assembly Cache* (*GAC*).

2. *The BizTalk Group settings* in Visual Studio 2008 or deployment in Visual Studio 2005 are set to your BizTalk development environment. It is a good practice to get into the habit of setting an application name for the project that you develop and not deploy to the default "BizTalk Application 1".

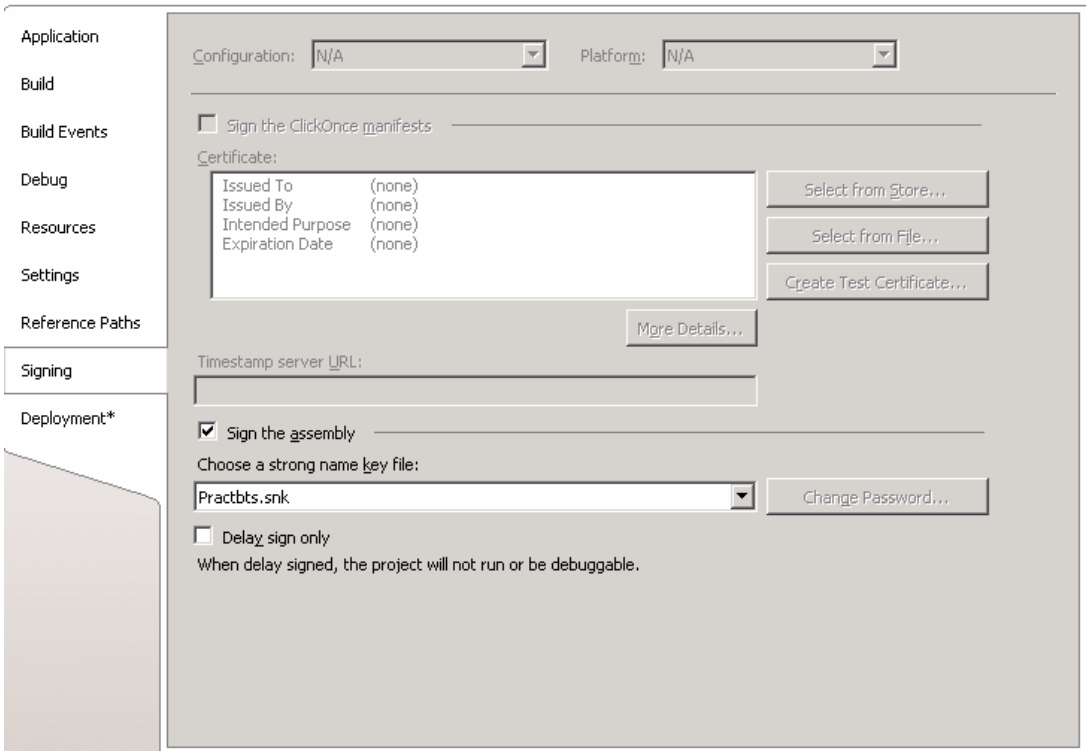

Figure 1.10: BizTalk Project Properties in Visual Studio 2005.

Now you are ready to add artifacts to your BizTalk Project. The artifacts can be *Schema, Pipeline, Map, or Orchestration*. You just right click on the project and select add new item and select one of the items in the dialog as shown in figure 1.11.

After you are finished adding the artifacts you can build or deploy the project with a single click. If you right clicked on the project and selected "*Deploy*", *Visual Studio* will

check the build status of the project and if the project is not built it will try to build it. If the project is successfully built *Visual Studio* will "*Deploy*" it to the *BizTalk Group* you set in figure 1.7 and 1.8. Deploy means that visual studio would *GAC* the assembly to the local computer GAC and add an entry into the *BizTalk Group Management* database.

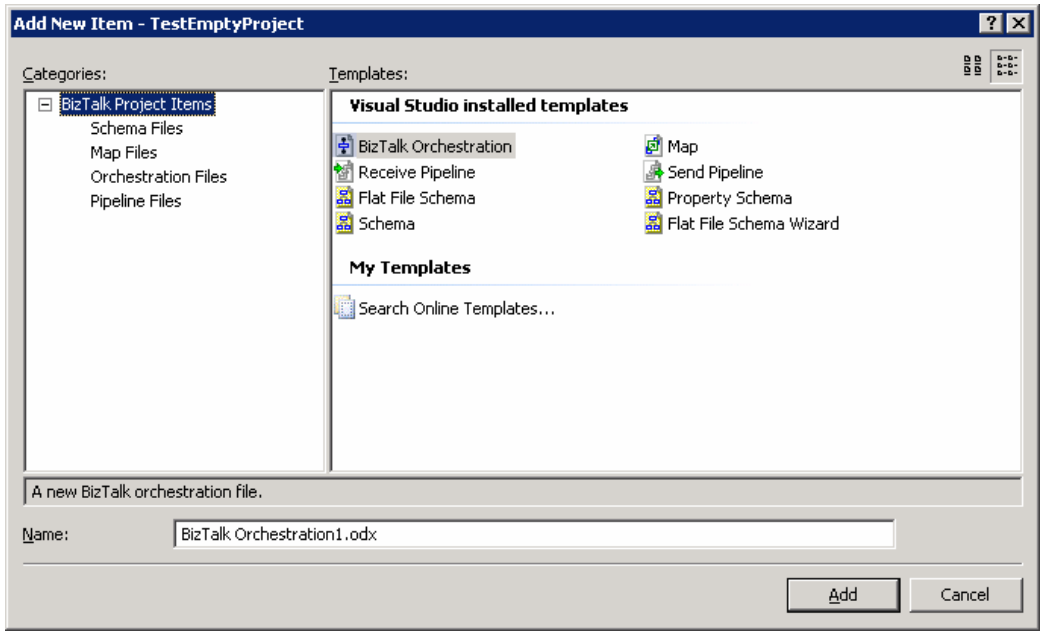

Figure 1.11: Add New Item for BizTalk Project in Visual Studio 2008.

If the build and deploy goes fine you will see in the output something similar to Listing 1.1. Note that I have edited the listing to keep it short.

Listing 1.1

```
----- Build started: Project: BTSPracCourse.Schemas.Sample, Configuration: Debug Any CPU ------
BTSPracCourse.Schemas.Sample                                                              ->
C:\Development\BTSPracCourse\PracBTS2009\BTSPracCourse.Schemas.Sample\bin\Debug\
------ Deploy started: Project: BTSPracCourse.Schemas.Sample, Configuration: Debug Any CPU ------
```

```
.
.
.
   : Deploy operation succeeded.

    : Deployed the following 1 BizTalk assemblies:
BTSPracCourse.Schemas.Sample.dll
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Deploy: 1 succeeded, 0 failed, 0 skipped ==========
```

1.6 BizTalk UI Tools

BizTalk comes with several user interface tools to facilitate the administration, configuration, deployment and development of *BizTalk Based Solutions*. The *UI Tools* are:

- *BizTalk Administration Console* is used to manage BizTalk applications, platform settings, artifacts and other *BizTalk* components. I will discuss it in more in the section 2.4.1.
- *Health and Activity Monitor* used to monitor the health of your *BizTalk Server* data. I will discuss *HAT* in more detail in section 2.4.2.
- *BizTalk Explorer* which is a *Visual Studio* add-in tool window that is similar to the familiar *Server Explorer* tool window in *Visual Studio*. It displays the contents of a *BizTalk Management* database. You use it to administer *BizTalk* from *Visual Studio 2005* without having to go to the *BizTalk Administration Console*. The *BizTalk Explorer* is still available with 2009 however it is depreciated and its use is not encouraged.
- *Business Rules Composer* used to create and modify policies and vocabularies. I will discuss the *Business Rules Composer* when we explore the rules engine in chapter 7.

- *Business Rules Deployment Wizard* used to deploy the policies and vocabularies to a *BizTalk* server. I will discuss the *Business Rules Deployment Wizard* when we discuss the rules engine in chapter 7.
- *Web Services Publishing Wizard* used to create and modify *BizTalk Web* service.
- *WCF Services Publishing Wizard* used to create and modify *BizTalk WCF* service.
- You use the *Tracking Profile Editor* to extract the data the business analyst requires and to perform the mapping between the specific business event data and the actual orchestration. I will discuss the *Tracking Profile Editor* in Chapter 8.

Throughout the book when I am discussing *BizTalk* features that apply to both 2006 R2 and 2009 I will refer to it as *BizTalk* only, and I will explicitly mention the version when discussing specifics that apply to one version only.

1.6.1 BizTalk Administration Console

The *BizTalk Administration* console (shown in figure 1.12) is a tool that you should get yourself intimately acquainted with. Though we think of ourselves as developers, architects and so on, there are many integration solutions that you can "develop" (or configure more accurately) without writing a single line of code.

Figure 1.12 shows the *Administration Console* grouping logical items (called artifacts used in a *BizTalk Server* business solution) in what is called a *BizTalk* application. Artifacts are:

- *BizTalk* assemblies and the *BizTalk-specific* resources containing: orchestrations, pipelines, schemas, and maps.

- .Net assemblies that do not contain BizTalk-specific resources.
- Policies.
- Send ports, send port groups, receive locations, and receive ports.
- Other items that are used by the solution, such as certificates, COM components, and scripts.

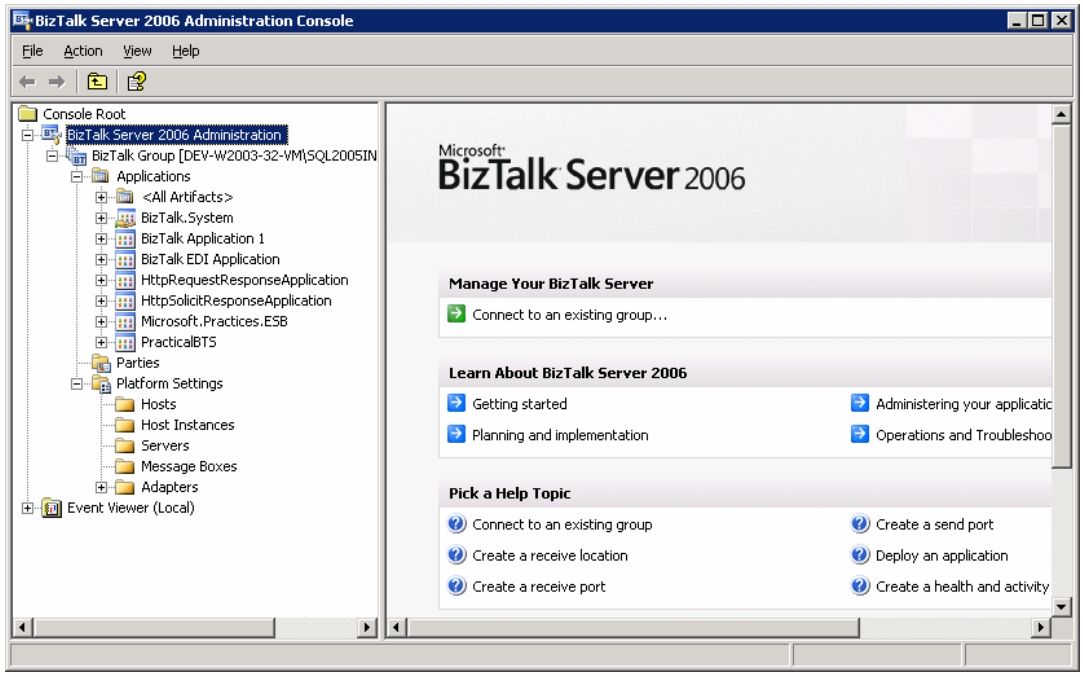

Figure 1.12: BizTalk Administration Console.

If you expanded the Platform Settings node in the Administration console, as shown in figure 1.13, you will see two important items: "*Hosts and Host Instances*" A *BizTalk host* is nothing more than a logical container.

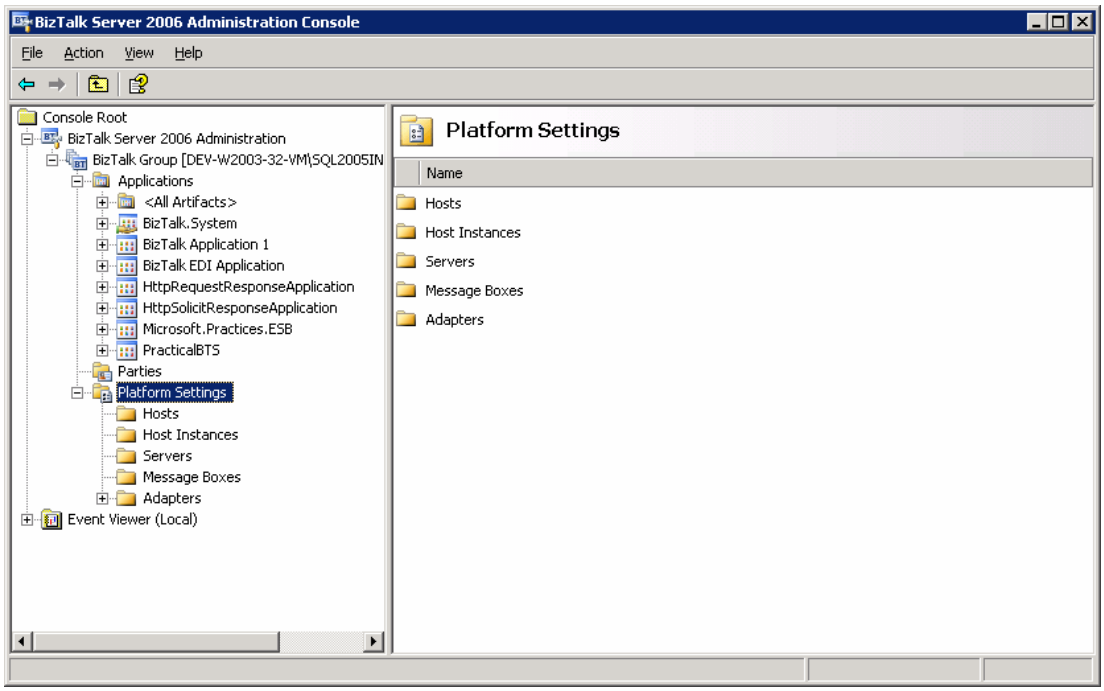

Figure 1.13: Platform settings.

A *host* defines which adapters, orchestrations, and ports are going to run in its context and under its security context. A host instance is just that, an instance of the host. The instance is actually just a service that runs on the machine called *BTSNTSvc.exe*. This process provides the *BizTalk* engine a place to execute, and allows for instances of different hosts to be running on one machine at a given time. Each host will end up being a separate instance of the *BTSNTSvc.exe* service from within the Windows Task Manager.

1.6.2 Health and Activity Tracking (HAT)

The *Health and Activity Tracking (HAT)* (shown in figure 1.14) is a GUI tool that provides utilities for reporting, analyzing, and debugging data and messages which are

archived in the *BizTalk Tracking databases*. For live data, use the *BizTalk Administration Console*. You can either use the included queries that appear on the *Group Hub page*, or select the *New Query* tab to create custom queries and reports.

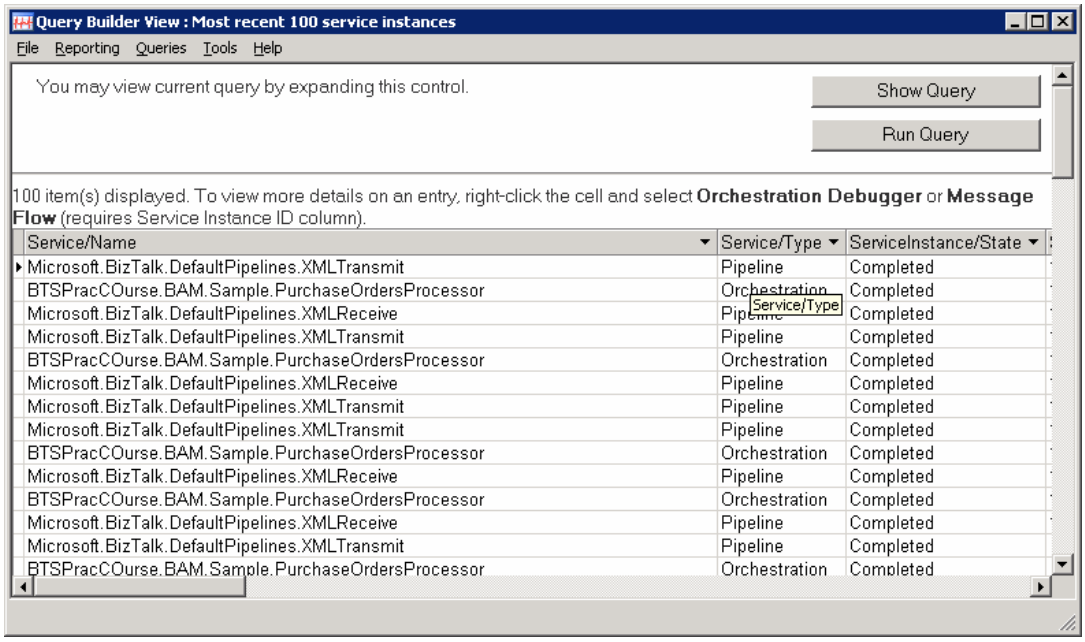

Figure 1.14: Health and Activity Tracking (HAT).

You use *HAT* to debug orchestration by running an instance of an orchestration then right click on it and select orchestration debugger shown in figure 1.15. You can also use *HAT* to see the message flow of a message by right clicking on the message and selecting message flow shown in figure 1.16.

1.7 BizTalk Project Architecture

How would you organize a BizTalk Based solution? In general, you should always strive to design the BizTalk solution to be layered as Figure 1.17 shows, designing the solution in a layered manner simplifies the project and makes it easier to maintain.

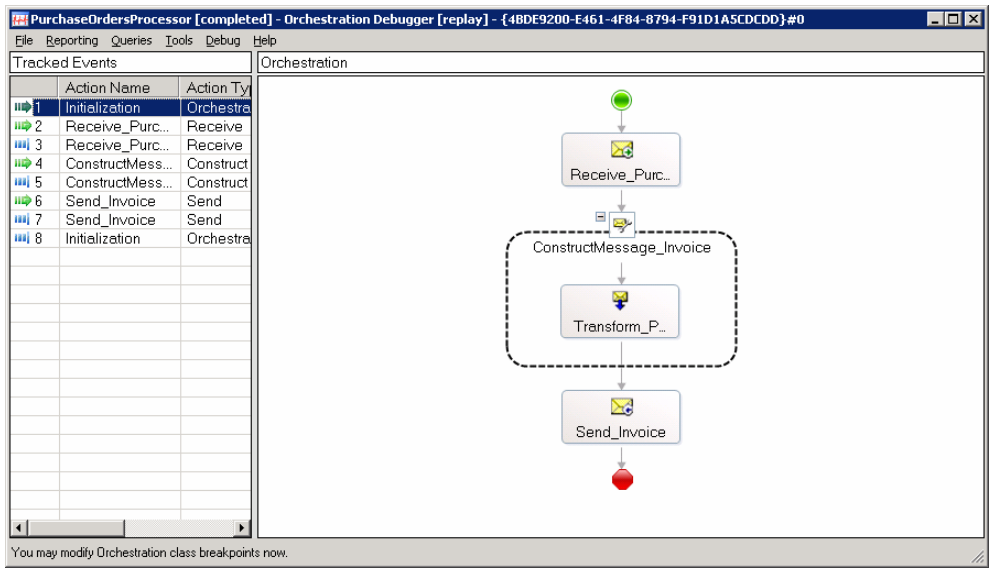

Figure 1.15: Orchestration Debugger.

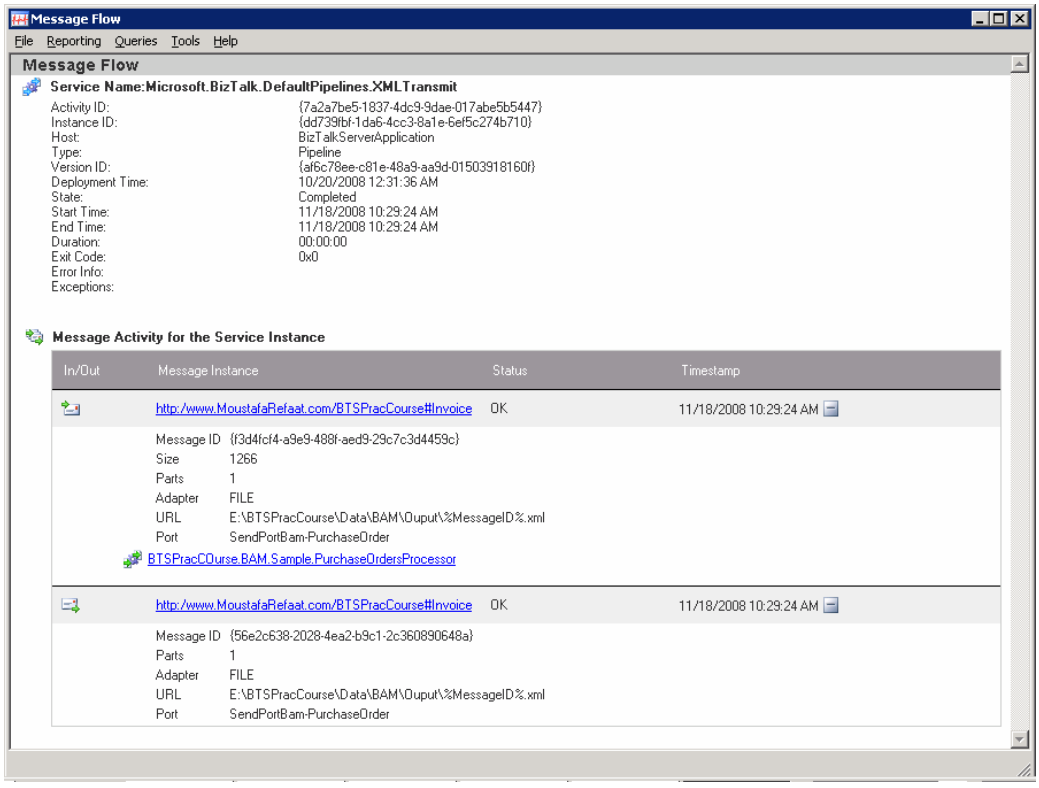

Figure 1.16: Message Flow.

You can design the solution to consist of at least the following three layers:

1. *External Interface Layer*: this layer contains all different maps, pipelines, schemas that would consume external data feeds and convert them to a unified internal schema that is internal to your enterprise. This can be implemented in a DLL or several DLLS, one DLL per feed, or several DLLs per feed. It depends on your solution, future changes (new third parties, external systems etc).
2. *The Business Logic Layer*: this is where all the business logic gets implemented in orchestrations, maps, business rule polices, etc.
3. *Enterprise System Interface*: This layer is similar to the External Interface layer except it is for internal systems in the enterprise.

You should note that we are always changing to a unified or a simple schema internal to the system. I have seen many BizTalk solutions where they had so many orchestrations or maps performing the same logic only because the third party supplied different schemas. On any BizTalk solution you should strive to:

1. Maximize the use of the BizTalk out-of-box features, and utilities,
2. Minimize the amount of code (this includes, custom pipeline components, custom adapters, custom libraries, etc), and
3. Prefer to buy components that are not out of the box from BizTalk than developing them.

As for structuring the code, there is no single answer to fit every solution. There is a tendency by many BizTalk developers to put the schemas, maps, orchestrations, and pipelines in separate projects. This is not suitable for large, medium and sometimes not even for small solutions. When you are structuring the solution code, you should

consider deployment of the DLLs and other artifacts, as well as the modularity of the solution.

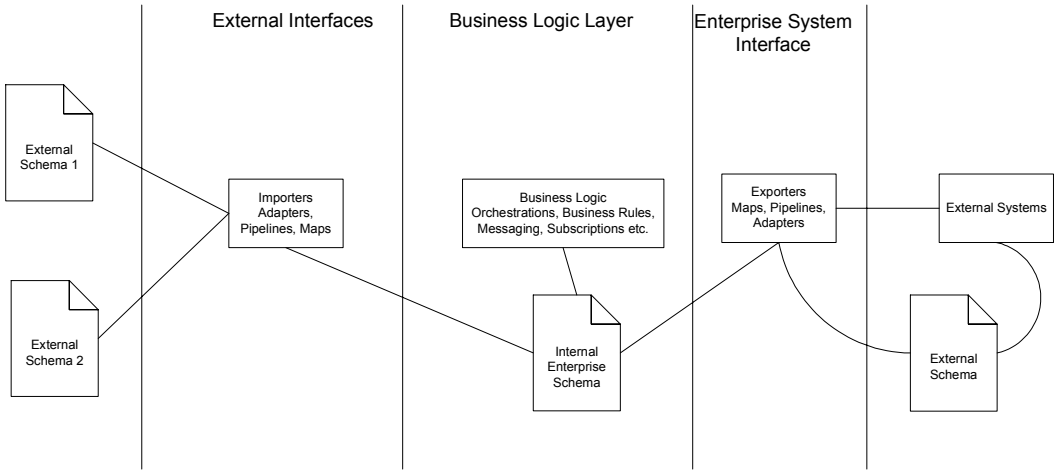

Figure 1.17: General BizTalk Based Solution.

For example, adding new partners and additional functionality should be made easy. You should also consider minimizing the dependency between developers. For example, when all they developers working on the same project, they may hinder each other's progress. Also if one developer is specialized in schemas, then other developers become dependent on her and waiting for her to finish the schemas so that they could start.

1.8 Design Patterns

In this section I will review a couple of the most used design patterns with *BizTalk* implementations.