BizTalk Integration Server Architecture

Satish R. Thatte

Microsoft Corporation
satisht@microsoft.com

Abstract: We begin with a brief explanation of how we view the current integration landscape. The paper then describes the mission and architecture of BizTalk Server, which is Microsoft's integration server product. We conclude with some thoughts on the future of the business integration domain, which we believe is still in the early stages of its technology lifecycle. The range of target problems, processing models and even architecture principles for integration solutions are likely to be evolving rapidly for some time.

1 The Integration Landscape

1.1 The Problem Domain

A vendor receives a purchase order from a customer, a physician orders a medication change for a patient, an investor transfers funds between financial institutions, a new employee joins the company, a retail outlet makes a sale. Every kind of business is driven by myriad events that occur in its environment. In the IT world today, these events are transmitted as business documents, each in its own idiosyncratic form based on its domain and origin: an EDI purchase order, an HL7 hospital pharmacy order, or a SWIFT funds transfer order. A large fraction of the business integration scenarios today are about highly reliable asynchronous straight-through processing of such business events in the form of documents flowing to and from partners, employees, business systems and devices such as point-of-sale terminals.

Over the next decade, the heterogeneity of document formats will evolve towards XML and the heterogeneity of communication modes will evolve towards a uniform web services protocol fabric. However, the core pattern of reliable asynchronous straight-through processing of business events will remain invariant because it reflects the way the world actually works. In some ways this pattern has not changed for decades since the days of nightly batch processing of the day's business updates in mainframe systems. The difference today is that business systems are more pervasive, more heterogeneous, more connected and involve many more aspects of business than ever before. And the tolerance for latency between business events (e.g., stock trades) and their reflection in systems of record (settlement) has gone down from days to seconds. The data in business systems must be kept consistent, often across enterprise boundaries; it must be updated in near-real-time in response to business events, and must be made available in useful aggregates for business users to view and act on. This is the systems integration problem as it is commonly understood today, and it has now become the biggest pain point and highest priority for investment in nearly every large IT shop.

At its lowest level, business systems integration is about normalizing and routing business documents. But nearly every non-trivial integration scenario of both basic types, asynchronous straight-through processing and synchronous portal access, goes beyond normalization and routing, and requires a programming model for integration, i.e., orchestration. In portals, for fetching and aggregating data from multiple backend systems, and for coordinating multi-system, multi-transactional update actions. In straight-through processing, for multi-document correlation, orchestrating document processing through multiple systems of record, and for driving business protocols with external partners in B2B scenarios. In all cases orchestration must provide capabilities for check-pointing and failure recovery across multiple ACID transactions to ensure consistency of business data.

1.2 Integration and Web Services

The bulk of integration today occurs with "legacy" protocols and formats. EDI over FTP is a common document transmission solution. This will continue and even grow for some time. Established legacy has a tendency to linger—mainframes are still with us despite numerous reports of their impending demise over the years. Nevertheless, the trend towards Web services as the integration fabric of the future is clear.

Web services are interesting because they promise to make integration of heterogeneous systems easier by standardizing the reliable communication protocols, interface contracts, and security and other policies across platforms. There are actually two orthogonal dimensions in the web services architecture that people have mixed up since the earliest days of SOAP [Bo00] and WSDL [Ch01]. There is a communication model and there is an interface model.

SOAP as a protocol framework has been used to develop an entire portfolio of WS-* wire protocols with the aim of creating an interoperable communication fabric. With WS-ReliableMessaging [Bi04] and WS-Security [WSS04] in place, it is possible to make the MSMQ and MQSeries category of enterprise communication services interoperable. Similarly, WS-AtomicTransaction [Ca04] provides a platform-neutral wire protocol for transaction managers to coordinate ACID transactions across platforms. This protocol can be expected to enjoy much broader support in the industry than previous attempts such as IETF's TIP protocol [LEK98] which Microsoft also helped define and supported in DTC [DTC04].

WSDL and WS-Policy [Ba04], on the other hand, are used to define interoperable service contracts that can be thought of as normalization of API rather than communication. Of course, SOAP/WS-* protocols and WSDL/WS-Policy are intimately linked in that the canonical access method for a normalized WSDL API contract is SOAP. But the key point is that with an interoperable API in the form of WSDL service contracts, applications can use each other's services in a loosely coupled but secure and reliable way so that application architectures become more flexible. The reality of a heterogeneous platform world is then no longer a high barrier to climb in building new application functionality based on existing enterprise application assets. Web service interfaces thus increase the reach of developers building new applications and help make them more productive in overcoming point-to-point integration problems in enterprise settings.

But point-to-point integration is about the "last mile" connectivity in an integration solution. Thus Web services are a boon but not a panacea. Point-to-point solutions are inherently non-scalable due to combinatorial explosion in the presence of the many-to-many integration problems common in enterprises. Moreover, integration requires long-running coordination, e.g., BPEL4WS [An03] processes in the world of Web services. Thus there is an enduring need for a coordinating integration broker.

But there is another interesting reason to believe that the Web services fabric is only one dimension of the integration architecture. Web service based application contracts may be less tightly coupled at the wire protocol level than the client-server relationships enabled by DCOM, CORBA and similar distributed object technologies. But at the level of semantic coupling, Web service API contracts are actually quite similar to the older technologies. The client of a Web service is invoking a known processing function and is dependent in its own semantics on the semantics of that function. A much more loosely coupled model relies on reactive event-driven coupling in which the source and the target of an event know little or nothing about each other and are coupled through a coordinating broker with pub/sub routing capability. Event-driven architecture is very natural for many integration problems. The distinction and complementarity between service-oriented and event-driven business integration architectures is something that analysts like Gartner are now beginning to recognize and articulate as an industry direction [SN03]. Business events as key drivers of value are also beginning to feature in the vision of leaders of technology companies [Gi04]. The majority of asynchronous straight-through processing of business messages will continue to use the event-driven processing model even when the communication fabric used is based on WS-* protocols.

2 BizTalk Server Architecture

2.1 The Mission

BizTalk Server is an integration broker that is optimized for asynchronous straight-through transactional processing of business messages in all formats and carried over an extensible set of protocols. The messages may represent service invocations or business events. BizTalk Server is aimed at demanding scenarios requiring high throughput with extreme reliability. It includes capabilities for multi-message coordination, failure recovery, load balancing, and high availability. BizTalk Server provides a focal point for configuration, management, tracking and monitoring of integration solutions, avoiding the "spaghetti" of point-to-point integration links.

BizTalk Server supports both synchronous and asynchronous Web services. BizTalk Server orchestration provides reliability and recoverability for long-running process behavior with a declarative programming model for state logging and recovery, based on a generalization of multi-transaction Sagas [GS87]. The model is essentially identical to the compensation mechanism in BPEL4WS; in fact the BPEL4WS mechanism was modeled on the one provided in BizTalk Server orchestration. BizTalk Server also supports real-time synchronous information retrieval, transformation and aggregation for portal scenarios.

In the rest of this section we describe the two primary dimensions of the current BizTalk Server architecture. The abstract processing model describes the way integration solutions are composed through an asynchronous message-driven architecture. The deployment architecture describes the way the solutions are deployed in the server environment to ensure a high level of reliability, scalability and availability. The complete stack of current BizTalk Server technology is shown in Figure 1. The architecture description in this section focuses on the processing model and deployment architecture of the bottom three layers (runtime engines, service container and communication). Tools and business user services including Business Activity Monitoring (BAM) are described elsewhere.

2.2 Abstract Processing Model

We realized early on that the range of scenarios, design patterns, processing models and even architecture principles for integration solutions would be evolving rapidly for some time, given the low level of maturity in the domain as a whole. We therefore wanted to build a server architecture that would be resilient to change and would enable incremental investment to broaden its capability and reach. Since business integration involves high-value data that cannot be lost or corrupted without serious consequences, the use of database transactional technology as the base was obvious. But beyond that we wanted to keep the abstract processing model of the integration engine extremely simple and general. We believe we have largely succeeded in this goal.

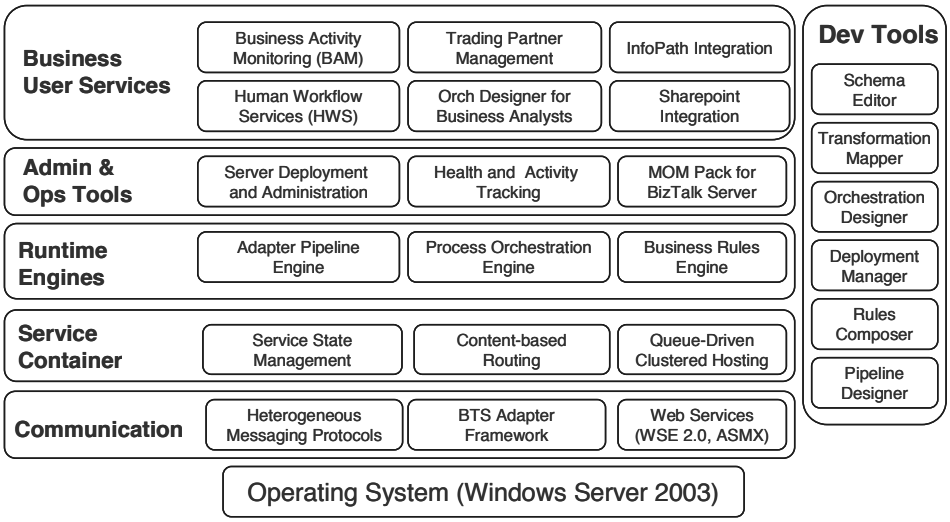

Figure 1: BizTalk Server Technology Stack

There are only a few functional elements that make up the BizTalk Server core, which is basically a database application.

- A durable message store where all messages are persisted. Messages are made of multiple parts in general, following the WSDL (and MIME) message type model. Messages and their parts, once persisted in the store, are treated as being immutable. Only references to them are used within the database in other functional elements.
- Every message has associated with it a set of properties of interest. These properties are stored in a separate table associated with the message store. Properties are of two kinds
    - Those which reflect the context of the message, e.g., the URI at which it was received, the identity of the source.
    - Properties within the message content that have been "promoted" to the surface because they are of special interest, e.g., the purchase-order-number in a purchase order confirmation document, which may be used to correlate the purchase order to a business process instance.
- A store for the state of long-running services such as business process instances. Instances of long-running processes are quiescent through most of their lifecycles, and the scalability of the server environment is maintained by "dehydrating" the state that represents the process context of quiescent instances. However, the process context of each instance is declaratively correlated with related messages through content-based routing rules. When a related message arrives, the routing rule associates the queue entry for the

message reference with the appropriate process context. Finally, when the message is picked up by a service host for processing, the quiescent instance is automatically "rehydrated" by picking up the associated process context in the same transaction. The server thus provides highly optimized support for building long-running processes and durable services in general.

- Application queues to which message references are routed by the content-based routing mechanism (see below). Application queues are the "internal endpoints" from which services "pull" their work.
- Content-based routing. This is the mechanism that ties the engine together. When a message, with its properties of interest, is persisted, the routing engine applies routing rules to determine the services which need to process the message and routes message references to the appropriate application queues.

Typical services configured to run in BizTalk Server today are

- Synchronous and asynchronous adapters for internet transfer protocols (FTP, SMTP, HTTP), Web services, and adapters for proprietary systems with their own peculiar formats and access methods.
- Orchestration process services similar to those expressed in BPEL4WS. BizTalk Server uses BPEL4WS as a *process metadata interchange* format and uses its own .Net based internal representation for actual running processes.
- Other stateful services such as drivers for long-running session-based message transport protocols (MSMQ) and batching services that encapsulate batching rules for specific trading partners.

Services in BizTalk Server are configured to run in specific abstract application hosts which provide security, scaleout, availability and other common support for the services hosted in them. Moreover, these hosts are the granularity at which the core server provides application queues, i.e., there is one application queue in the database per abstract host. This allows natural load balancing and throttling based on multiple clones of application hosts being tied to the same queue and pulling work as they are able based on their load.

The routing rules used by content-based routing fall into two categories

- Stateless rules that are relatively static—used for example to activate new service instances, or to always route certain types of messages to specific applications or partners.
- Stateful rules that are generated by long-running service instances, the most interesting example being rules that reflect correlation of messages to business process instances.

Custody of stateful instance data in an environment with scaleout based on cloned application hosts brings with it responsibilities for instance state locking and other correctness and performance requirements. Describing those aspects is beyond the scope of this paper. Suffice it to say that BizTalk Server provides efficient mechanisms for instance state hydration/dehydration, instance locking, lock release and service recovery in case of failures, and so forth.

The diagram in Figure 2 depicts the abstract processing model described above

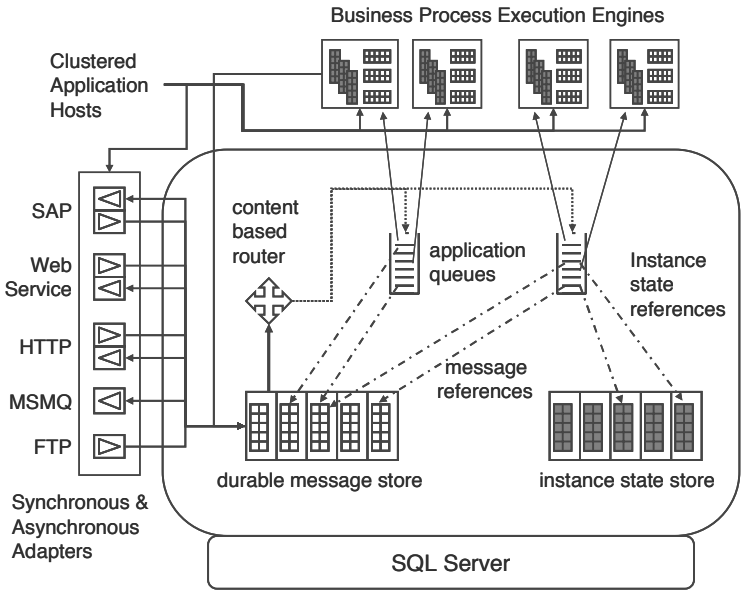

Figure 2: BizTalk Server Abstract Processing Model

2.3 Deployment Architecture

A high-end deployment of BizTalk Server uses a number of inexpensive server machines for application hosting. Application hosts are in effect stateless since all data is held in the database and is only used within the context of ACID transactions as the application services process the data and make internal state transitions. Thus the application host server machines are expendable and when application hosts are cloned across multiple servers, availability is ensured in the obvious way—if a server machine fails no data is lost and its work is transparently picked up by other servers that are running clones of the abstract application hosts that the failed server machine was running.

The database tier is obviously far more critical and in large-scale mission critical installations, servers in this tier are typically hosted in MSCS clusters with failover capability. Given the database-intensive nature of the processing model used, database capacity is the most common bottleneck in scaling up the performance of the server; database lock contention and other problems ultimately limit the degree of scaleout of the stateless application servers. BizTalk Server supports the use of multiple dynamically partitioned databases with identical abstract model structure (as described above) in order to scale the database capability in really large installations. The dynamic partitioning is based on the ownership of service instance state by individual partitions, and global routing rules ensure that the right messages are processed via the right database. Further details are unfortunately beyond the scope of this paper. The diagram in Figure 3 depicts a typical high-end deployment of BizTalk Server.

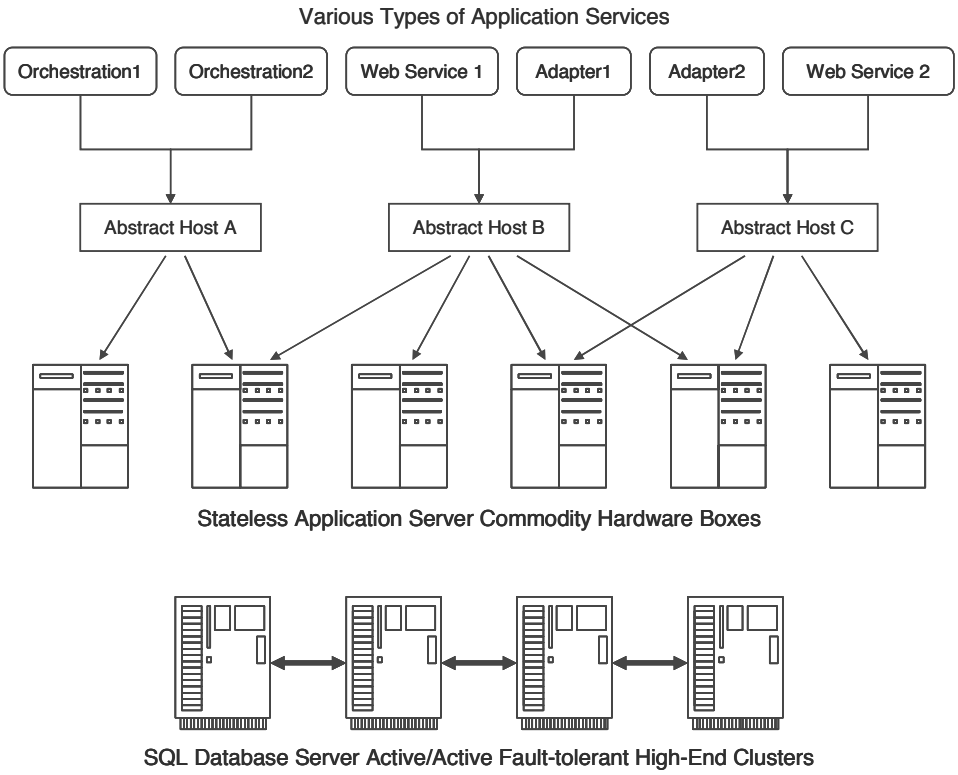

Figure 3: BizTalk Server Deployment Architecture

3 The Future: We Have Only Scratched the Surface

In the broadest sense, business integration and business process are synonymous. If one takes a comprehensive view of the notion of business process, then "real" business processes in domains such as supply-chain, healthcare, financial services, etc., are today embedded in line-of-business applications in the form of the process logic attached to business entities that represent the data of record. Both enterprises and application vendors recognize that the processes associated with business applications need to evolve far faster than the applications themselves to meet business needs. In the absence of this kind of process agility, business applications as repositories of process can go from being business enablers and organizers to millstones around the neck of an enterprise. Given the current slowdown in technology spending, this is becoming a key driver of the search for "business value" in the entire area of business application and integration software. Nobody has yet (at least publicly) articulated an adequate approach to separating the business application logic into different modes based on the "clock-speed" of its evolution, but it is clear that traditional process models are not going to be sufficient for restructuring business applications, although they will do perfectly well in their current sphere of message-driven service-composition and long-running conversation management processes.

The limitations of current models are partly due to the fact that the origins of many of the modeling metaphors of the day are in models of communication (pi and join calculi) and synchronization (Petri nets) rather than in models of data-centric behavior which predominates in business applications. Business communication may trigger or be triggered by processes that manage business data, and the management of such communication is of course part of the overall business process embedded in applications, but not the core of it. Process calculi are very good at representing the logic of long-running business messaging protocols. But for that very reason, their internal elements tend to have deeply interconnected semantics and it is very difficult to take such a process and evolve it piecemeal without running into a host of semantic issues. But incremental evolution is a critical requirement for process improvement. Moreover, the processes embedded in business applications are frequently open-ended—for instance, a supply-chain process may need to be able to deal with recalls, disputes, returns, etc., long after the product is shipped. In other words, a large part of the process logic deals with contingencies rather than long-running conversations. Processes may even be simultaneously very dynamic and very long-lived, so that the process model may need to change not only for new instances of a process but even for instances that are in progress. The ability to support this category of application-embedded agile processes is an essential requirement for the business integration platform of the future. In some sense, this requires a deep restructuring of the very notion of business application. The challenges make this an exciting area for both academic research and commercial innovation. We have a long and interesting road ahead of us.

4 References

[An03] Andrews, Tony et al: Business Process Execution Language for Web Services, version 1.1; Link: http://msdn.microsoft.com/library/en-us/dnbizspec/html/bpel1-1.asp, 2003.

[Ba04] Bajaj, S. et al: Web Services Policy Framework; Link: http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policy.asp, 2004.

[Bi04] Bilorusets, R., et al: Web Services Reliable Messaging Protocol; Link: http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-reliablemessaging.asp, 2004.

[Bo00] Box, Don et al: Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000; Link: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, 2000.

[Ca04] Cabrera, L. F. et al: Web Services Atomic Transaction; Link: http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-AtomicTransaction.pdf, 2004.

[Ch01] Christensen, Erik et al: Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001; Link: http://www.w3.org/TR/wsdl, 2001.

[DTC04] Microsoft Product Documentation: Using TIP with DTC; Link: http://msdn.microsoft.com/library/en-us/cossdk/htm/pgdtc_dev_48kz.asp, 2004.

[Gi04] Gillmor, Steve: SAP's Agassi Lays Out Business Event Network, eWeek enterprise news and reviews article; Link: http://www.eweek.com/article2/0,1759,1563262,00.asp, 2004.

[GS87] Garcia-Molina, Hector; Salem, Kenneth: Sagas, Proceedings of the 1987 ACM SIGMOD international conference on Management of Data, 1987.

[LEK98] Lyon, J; Evans, K; Klein, J.: Transaction Internet Protocol, version 3.0. ; IETF Network Working Group RFC 2371; Web Link: http://www.ietf.org/rfc/rfc2371.txt, 1998:

[SN03] Schulte, Roy W.; Natis, Yefim V.: Event-Driven Architecture Complements SOA, Gartner Research Note DF-20-1154, 2003.

[WSS04] OASIS Web Services Security Technical Committee; Link: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, 2004.