

Seven Showstopper Problems with BPEL Servers for Event-Driven SOA

Powering Next Generation SOA Networks

By Atul Saini, CEO and CTO, Fiorano Software, Inc.



Fiorano

Fiorano Software, Inc.
718 University Avenue, Suite 212
Los Gatos, CA 95032 U.S.A.
+1.408.354.3210
email: info@fiorano.com

Entire contents © 2007 Fiorano Software Technologies Private Limited and Affiliates. All rights reserved. Reproduction of this document in any form without prior written permission is forbidden. The information contained herein has been obtained from sources believed to be reliable. Fiorano disclaims all warranties as to the accuracy, completeness or adequacy of such information. Fiorano shall have no liability for errors, omissions or inadequacies in the information contained herein or for interpretations thereof. The opinions expressed herein are subject to change without notice.

How Limited Support for Event-Driven Architecture (EDA) Restricts the Value of BPEL products

Executive Summary

Most real-world integration scenarios in a typical enterprise involve multiple applications (or application components) that run on distinct physical machines across an enterprise network, are developed in different languages and run on different operating systems. As such, the typical scenario for integration involves the flow of events (data-flow, event-driven architecture, or EDA) and exchanges of requests (request/reply interactions) between service components distributed across a heterogeneous network.

A Forrester survey estimates that 64 percent of companies spend between 60 and 80 percent of their integration budget on consulting services, most of which go towards programming the infrastructure to connect and transfer events through the enterprise. The reason for these continued high costs lies in the fact that in real-world implementation conditions, current integration solutions based on BPEL suffer from some critical problems in the implementation of event-flows (EDA) resulting in rigid, complex implementations that are difficult to maintain and modify, leading to delayed projects, poor ROI and major cost overruns.

This whitepaper analyzes some critical implementation-level problems faced by current BPEL products in real-world implementations. Most of these problems occur because existing integration products do not seamlessly support request/reply and EDA within a single framework – a shortcoming that is addressed by ESB implementations based on data-flow, EDA and SCA (Service Component Architecture) concepts.

It is important to note that the problems discussed below apply not only to BPEL-centric platforms but to any integration platform that is based on a process-centric language similar to BPEL. Thus, many existing vendor products (including those from Oracle, WebMethods, TIBCO, IBM and BEA to name a few) based on older process-languages that precede BPEL also suffer from the problems analyzed in this paper.

Summary of Problems

The problems with current BPEL products include:

1. Inability to automatically create and deploy event-flows between distributed service components
2. Inability to easily create request/reply and event-driven SOA exchanges across multiple component-models and platforms
3. Inability to handle network-failures across Service components in a distributed environment

4. Inability to easily configure service components (adapters, services and applications) participating in a distributed integration
5. Inability to easily debug the flow of events and requests across multiple distributed service components.
6. Inability to provide equal citizen status to multiple programming languages (C, C++, VB, Perl and others)
7. Inability to easily map changes to abstract Business Process Flows to actual implementation-level event flows and request/reply interactions between distributed service components

The rest of this paper examines each of these problems in greater detail with specific explanation about why current BPEL products fail to address these issues

1: Inability to automatically define event-flows between distributed service components

Almost all integrations require events to be routed in some order between distributed service components/applications running on distinct machines in a heterogeneous network. Such integrations are surprisingly hard to set up on existing “process driven” BPEL products, since the process management tools provided by such products are not integrated with the underlying transport infrastructure to automatically create event-flows without programmer intervention. All event flows have to be manually configured, typically using JMS, MQSeries or other messaging middleware, increasing the time and complexity of the implementation.

For instance, in a typical scenario, queues or topics need to be configured for all data flows, and name-clashes need to be manually detected; developers have to understand low-level messaging concepts such as “Topics, Queues, durable subscriptions, persistent and non-persistent messages”, etc. Existing BPEL products provide no tools to isolate developers from these details, resulting in unnecessary complexity and increased time to deployment. The principal reason for this is that the underlying BPEL implementation of computation does not provide the address-indirection or intelligent routing capability that is needed to connect the event-flows between the top-level business components to the lower-level queues/topics that physically transfer the messages. In addition, to reduce development time, many BPEL tools simply simulate messaging via request/reply interactions, resulting in extreme inefficiency at runtime.

2: Inability to easily create event-driven (EDA) data-flow exchanges across multiple component-models and platforms

All current BPEL products are limited in their support for multiple component-models and platforms, being heavily biased towards development using particular component models such as EJB, J2EE (for most vendors) and COM/.NET for Microsoft based solutions. As such, current BPEL servers are unable to easily create event-driven SOA interactions across heterogeneous software components based on different models (EJB, J2EE, .NET, etc.) already deployed within an enterprise. Modern integrations require support for multiple component types, with added support required for legacy applications and protocols. The inability of most BPEL products to easily implement data-flow (event-driven/EDA) interactions across multiple component models and legacy systems leads to significantly increased costs of consulting and implementation.

3: Inability to handle network-failures in a distributed environment

Since most BPEL products have a centralized hub-and-spoke design, any network failure results in end-point applications disconnecting from the broker. In the absence of any “store-and-forward” infrastructure implementation at the end-points of the network, the ability to handle network failures has to be pre-programmed into each service component, significantly increasing the complexity of any integration effort and decreasing the possibility of reuse of adapters and components (since implementations of fault-tolerance are different across all proprietary BPEL products).

For instance, in most current BPEL products, each service component has to have embedded support for multiple transport and security protocols (TCP, HTTP[s], SSL, etc.) and needs to be individually configured to connect to the centralized broker. Current BPEL brokers do not provide any tools or infrastructure-level support to ensure that service-component deployment is transparent to the underlying network topology, or to automatically route event flows across distributed business components. The lack of such support results in significantly increased time to deployment.

4: Inability to easily configure different business components (adapters, services and applications) participating in integration

Most current BPEL products do not provide adequate support to remotely configure the service components (adapters and applications) running at the network end-points to execute the integration workflow/process. Traditional BPEL broker suites require the configuration of each service component to be manually updated within a centralized repository, resulting in significant implementation complexity, as described below.

In a typical implementation of a service component, developers need to remember middleware concepts (such as topic or queue names), use a particular user interface for configuring an adapter, store the configuration and other meta-data formats in a centralized repository, and then manually load the corresponding meta-data formats into the workflow designer to compose the workflow. There are no tools to automatically fire a component configuration (which could be developed by a 3rd

party user) and automatically make the corresponding data-formats available to the workflow designer without significant manual intervention.

5: Inability to easily debug the flow of distributed events across multiple service components

Since most BPEL products are based on a centralized process-engine, they provide little or no support to debug the flow of events and requests across service components running at the end-points of the network. Current BPEL products require additional design and programming to ensure that events/requests are published on separate queues/topics for appropriate processing; they provide no support for dynamically changing the event-logging levels while the distributed flow is running, or to set breakpoints in a live distributed application after deployment to debug the flow of events across the network. Debugging is typically localized to the flow of control information within the broker only, which is very restrictive.

Since event-flows can get extremely complex in any non-trivial integration, the inability to debug distributed events can significantly increase implementation complexity and adversely affect delivery timelines.

6: Inability to provide equal citizen status to multiple programming languages (C, C++, VB, Perl and others)

Most current BPEL products are biased towards development in one or at most a few languages (typically Java, or C and C++). Modern integrations in a heterogeneous environment typically require integrations spanning multiple platforms, with applications written in variety of programming and scripting languages including Java, C, C++, C#, Visual Basic and Perl, to name a few.

Even the so called 'modern' BPEL products typically provide only a single language adapter/component development base, with wrappers for additional language support. For instance, several brokers support only native Java adapters; C adapters are invoked using JNI which is known to have several problems and limitations. As such, the lack of support for multiple programming languages can be a barrier to enterprise implementations.

7: Inability to easily map changes to abstract Business Process Flows to implementation-level event-flows across service components

Traditional BPEL products are based on the notion of creating information flows between abstract processes typically rendered using graphical process-design tools. However, the logical flow-diagram of the business process is completely distinct from the physical service components that execute across the network to realize the business process. This is because a single 'activity' within the logical BPEL business process may require the flow of data between multiple service components executing across different physical machines. As a result, what one sees logically on the process-management screen is not what happens physically at execution time.

A direct consequence of this fact is that a change to the "high-level" business process flow within a process design tool does not map directly to the routing of data between the relevant service components executing across the network. As a result,

changes to the high-level business process require significant manual coding to re-route the event-flows and request/reply interactions at the implementation level. In other words, existing BPEL products do not provide a single framework within which to handle request/reply and event-driven (data flow) interactions between distributed service components.

So What's the Solution?

The problems discussed above are inherent in the design of most current BPEL products and cannot be easily remedied without a significant rework of the basic architecture of such systems. Most BPEL products on the market today map well to synchronous request/reply interactions but do not map well to the implementation of other critical integration patterns including database consistency and multi-step processes, which require the flow of events between independently executing processes.

A solution to the problems of BPEL implementations requires

- An infrastructure platform that supports the intelligent routing of information, including both events and requests, across distributed service components over a network, and
- A framework (including APIs and tools) for creating business applications as a collection of modular service components

The first of these requirements is addressed by a distributed Enterprise Service Bus (ESB) architecture that allows service-components to run at the end-points of a network and communicate in a peer-to-peer fashion with support for store-and-forward messaging, without traversing a central broker.

The second requirement is addressed by Service Component Architecture (SCA). SCA unifies the notions of request/reply and event-flow (EDA) and turns the focus of development away from the notion of distributed computing (i.e. distributed request/reply and MOM) towards the creation of software modules called Service Components that follow the semantics of business functions. SCA tools allow the creation of request/reply and event-driven interactions between distributed service components, allowing processes to be created, deployed and modified dynamically in response to business requirements.

The Fiorano SOA 2007 platform is based on a distributed, event-driven ESB architecture, which implements the concepts of SCA. Applications deployed on the Fiorano platform overcome the seven problems outlined in this paper.