APPLICATION INTEGRATION PROBLEMS: 6 HAZARDS YOU’LL ENCOUNTER WHEN RIDING THE ENTERPRISE SERVICE BUS
APPLICATION INTEGRATION PROBLEMS: 6 HAZARDS YOU’LL ENCOUNTER WHEN RIDING THE ENTERPRISE SERVICE BUS

Perhaps you’ve taken a bus to commute to work. Maybe you’ve gone on a bus trip for a weekend vacation. If you have used buses for transportation, you’ve likely experienced a breakdown, traffic jam, or missed connection at some point in time. It’s frustrating, but it happens. The same is true when you board an Enterprise Service Bus (ESB).

An ESB is a software architectural model designed to connect the disparate parts of a business. The ESB seamlessly links together multiple business applications, data repositories, and data consumers. These connections are made via a service layer that contains the rules and a common framework to enable communication between these various entities.

It sounds positively idyllic, with the ESB moving data from application to application, whenever and wherever it is needed to facilitate business functions. And it is wonderful... unless you happen to encounter one of these six hazards that can turn your pleasant bus trip into a travel nightmare.
#1 HAZARD #1: THE BUS IS NOT EASY TO ACCESS

When you need to take a bus, you want to have easy access to it — from anywhere in the city. The same is true for an ESB: you want your applications to be able to connect swiftly to the services layer and get the data they need without any delay.

To ensure this happens, your foundational architecture should be simple (complications always create time lags). It should also be designed in a way that can support change without requiring redesigns or long test cycles. For example, suppose that today your integrations are SOAP-based, but that next quarter you move to REST or JSON API, and the following year you have solutions that are even more abstract. Your ESB needs to provide flexibility in adding or obsoleting services or components of your system so that connections are always readily accessible.

Creating a simple yet flexible architecture requires IT to be “business aware” rather than “IT focused.” IT needs to understand the business environment and the business roadmap, assessing what changes have occurred in the past and what changes might occur in the future. This information will enable IT to add the necessary placeholders into the ESB to make it easy to incorporate future changes if and when they arise.

For instance, what would happen if the business acquired a new company and wanted to merge customer data between the two systems? An IT-focused architecture might be hard-coded with point-to-point connections, making a mashup of disparate systems all but impossible. A business-aware architecture would have built in the flexibility to respond readily to just such a scenario.

#2 HAZARD #2: THE BUS TAKES TOO LONG TO GET TO YOUR STOP

Suppose you take the bus to get to work each morning. Wouldn’t you be frustrated if the bus took one route on Monday and a different route on Tuesday and a third route on Wednesday? Different routes would create unnecessary complexity — for you, for the other passengers, and for the bus driver. Yet developers create this scenario quite often in ESB implementation, then wonder why the ESB doesn’t function efficiently.

Here’s what happens: developers build specific services for internal use and others for external use, even though the services are providing the same data points. Basically, all the different endpoints that consume
...developers need to leverage the ESB's features of reusability, governance, and meta-driven framework to build rich services. The data get their own unique services as developers get into the habit of saying, “Let’s build a new service for everything.” The result? “Services spaghetti”: multiple services with differing rules for various applications, all calling on the same data!

For example, developers might create different services to process purchase orders in EDI, JSON, SOAP, or IDoc — each with different endpoints and request parameters. This is a complete waste of time and effort, and bogs down the ESB. Instead, developers should review what they currently have in their services library to determine what can be re-used. That is the whole point to having a service-oriented architecture (SOA) like an ESB. In the purchase order example, there might be one service that could provide the same data in different formats.

There have been complaints that SOA has not lived up to its expectations when it comes to ESB. People say that it is too complicated, or developers don’t understand it, or it’s easier to address pain points with old-fashioned roll-up-your-sleeve coding. But the real reason why traditional ESB applications have failed in enabling SOA is because they have been treated like glorified code-based Eclipse platforms.

At the end of the day, if an ESB generates code, it’s not an ESB! It’s just another code generation platform. To have an ESB function as an SOA, developers need to leverage the ESB’s features of reusability, governance, and meta-driven framework to build rich services.

**HAZARD #3: THE BUS SKIPS YOUR STOP**

Don’t you hate it when the bus skips your stop, leaving you waiting by the side of the road? It throws your entire day off. Designed incorrectly, an ESB can do exactly the same thing.

ESBs often must support long-running transactions that require waiting for application data to arrive before going to the next application “bus stop.” They may also need to integrate with human workflows where there are regular pauses while employees perform some necessary function. But, all too frequently, an ESB is not designed to take these breaks in activity into account. As a result, the ESB assumes that data is always correct and that responses to requests are always synchronous. It does not wait for more meaningful data to arrive, ignores context, and is indifferent to additional inputs. If the data it really needs is not there, too bad... the ESB skips that stop and moves on.
You can see this at work when an underwriter needs to write a policy for a commercial property. She requires supporting documentation to complete the policy, but that documentation may take several days to arrive. If the ESB she uses isn’t integrated with the business process itself, the ESB may move on without her, generating incorrect data along the way.

**HAZARD #4: THE BUS CANNOT BE CONVERTED INTO A HYBRID**

Buses have traditionally been diesel-fuel guzzlers, so it is a welcome change when they can be converted to more environmentally-friendly hybrids. Nevertheless, not all buses are built to accommodate such a conversion.

ESBs can face the same problem: typical ESB implementations are designed to be local in nature. They connect on-premise applications that exist within the company’s firewall exclusively; they do not connect internal systems with cloud-based applications.

But hybrid integrations are an important factor to consider when designing an ESB. Businesses are outsourcing more and more of their business processes to cloud applications (QuickBooks Online for accounting, Workday for HR, and Salesforce for managing customer relationships are a few popular cloud-based applications). So a key question becomes: “Can your ESB be converted into a hybrid to handle integrations with cloud applications?” That includes internal-to-cloud connections as well as cloud-to-cloud connections. Just as with physical buses, hybrid ESB designs are necessary to achieve optimal efficiency.

**HAZARD #5: THE BUS IS STUCK IN A RUT**

A bus stuck in a rut is a bad thing. Similarly, an ESB that has only static services will eventually prove to be a liability to the business.

For example, an ESB that provides a payload in one format will always deliver data in that format — such as an invoice being sent in XML. But what if you need the invoice in EDI or JSON? You could build an intermediary step in your application to transform the data into the desired format, but wouldn’t it be better (and easier) to submit your request to the ESB in a way that allows the target application to understand and convert the data to the format you want?
ESBs should offer dynamic services that can behave differently based on the type of request sent by the client. At Adeptia, for instance, an ESB is used as a dynamic service library. This includes a Mapping service where a client can send a request consisting of the type of mapping transformation required, schemas of the source and the target format, and where they want the response payload delivered. The Mapping service sends the result exactly how the client wants to the specified location.

**HAZARD #6: THE BUS SPUTTERS AND FAILS TO START**

A bad bus engine may be the result of multiple issues ranging from poor mechanical design to absentmindedly putting in diesel instead of gasoline or vice versa. In like manner, an ESB that sputters and fails to start may be caused by any of a wide range of issues. Fortunately, a solid SOA-based integration strategy that addresses the following points can avoid most if not all of the danger zones:

1. **Define the processes that need to be automated.** A process is defined as any operation that requires speed of data delivery with accuracy and that adheres to rules related to security.

2. **Map out all the applications, systems, and users involved with the data movement in the bus.**

3. **Determine how the data is going to be made available to the ESB.** This includes the ability to connect and receive data, and the architectural knowledge to make the necessary connections (i.e., to mainframes, databases, custom programs, cloud applications, etc.).

4. **Set the business rules for governing the services.**
5. Evaluate how systems are likely to change over time and build flexibility into the ESB to accommodate those changes.

6. Establish forms and web-based interaction portals to allow end-users such as business users or customers to interact with the information.

7. Define the owners of the data systems.

Having a clear understanding of these factors will allow you to design an ESB that meets your business goals.

YOU ARE IN THE DRIVER’S SEAT.

What happens during a trip on a physical bus is mostly out of your hands. The bus may skip your stop, experience a delay, or get stuck in a rut and there’s nothing you can do about it. But with an Enterprise Service Bus, you are not only the passenger...you are also the driver. All of these six hazards can be avoided with proper development of an SOA-based integration strategy. Do that, and your bus will take every user exactly where they want to go!

If you’d like to learn how you can create an efficient ESB solution that you can trust, call Adeptia at 1-312-229-1727 or visit adeptia.com.