INTEGRATING WITH SOA AND ESBS pdf

1: Key Differences between ESB, EAI and SOA

ESB, which is the abbreviation for "enterprise service bus," is a software architecture that provides integration of enterprise applications and services for complex architectures, such as middleware infrastructure platforms.

Clark Published on March 18, Nearly all enterprises have multiple applications that are the system of record for their key data, and business functions upon which the enterprise is built. It is no surprise, therefore, that we see organizations wanting to progressively surface these valuable assets in these operational systems to broader audiences within the enterprise or beyond. However, it has taken time. In this tutorial, we will chart the key stages in this evolution, help you to evaluate where you are as an enterprise on that journey, and consider what actions you might want to take to mature your integration architecture towards, or indeed beyond, API exposure. There is a clear evolution in integration architecture across the industry over the past few decades, with progressively greater degrees of exposure for a business function as shown in Figure 1. Progressive exposure of a business function View image at full size Our purpose is to understand the difference between SOA and modern business web APIs. In order to do that effectively, we need to have a clear picture on what SOA brings to the table. Getting users to re-key information from one system to another "swivel chair" integration was unsustainable for most circumstances. This introduced the need for direct point-to-point low level connectivity between siloed applications. Often, it was impossible to get real-time responses, so data was typically sent asynchronously via files or one-way messages. A new serialization and parsing code was required on each side for every interface as shown in Figure 2. Point-to-point integration View image at full size The different styles of lines between the applications depict that multiple different protocols were often required to achieve the different interfaces, further complicating the integration task. Application programming interfaces APIs , including transports, protocols, and data formats for real-time interaction, were introduced, where responses are gained directly back from the called system. We will clearly differentiate between the two in this tutorial by calling the original type as "low level APIs" and the new type as "web APIs". The techniques honed at each level within the model do not become obsolete when you move to the next level, they just get used more selectively. For example, even in a company implementing services at SIMM Level 4, there may still be a perfectly valid need for an occasional point-to-point or hub and spoke integration. These low-level APIs varied dramatically across platforms, which involve complex application-specific connectivity code to be written into in the applications on both sides of every integration. They knew how to perform the connectivity and provided a central hub through which all integration could be performed. This enabled a more "hub and spoke" architecture and significantly reduced the amount of proprietary integration code being written as shown in Figure 3. Hub and spoke architecture View image at full size This new tooling and techniques meant that you could reuse the connectivity within the scope of the integration hub - you only needed to work out how to connect to an application once. The same tools and runtime were always used for the job rather than the integration code in multiple languages and on multiple platforms. Due to the radically different interaction styles between applications, they were typically not connected in real-time. More typically, an inbound adapter draws data from a system into a file or message based store, then an integration flow manipulates the data and passes it on to the target systems. This inevitably results in a significant amount of data replicated across systems, when the data is only really required for reference purposes. A real-time interface to the original system can reduce this replication. Gradually, real-time interfaces to operational systems became more common, lessening the need for replication of data across systems. However, for a new system to use one of these real time interfaces, there was still some work needed to connect it to the hub. Indeed, many attempts at hub and spoke architecture only alleviated the point-to-point problem slightly, by bringing point-to-point coding into one runtime and one tooling. Unless the integrations were carefully designed for re-use, there was still significant new code needed to create a new interface. We need a more standardized way to expose functionality from the hub so it can be re-used without extra work. This meant that for services, it was possible for requestors, who understood those modern standards, to make use of the services with minimal effort. Direct re-use of these exposed business

functions now became possible. Any opportunity for re-use brings new benefits and also new challenges. Indeed, it leads to many challenges from unmanageable dependencies between systems to security exposures. From a service exposure point of view, SOA is much more complex than just the standardization of protocols and data formats. To expose services effectively, you need standardization of the following aspects too: The consumer must call a virtual service endpoint that hides the complexities of how and where it is ultimately implemented. Conversion to the standardized protocol and transport is part of the virtualization, but services also need to provide standard configurable aspects, such as routing and data translation while continuing to provide the same virtual service to the consumer to minimize the effects of change. If you are exposing core business functions, you need to manage and monitor them. To achieve effective monitoring at scale, it needs to be done in a standardized way across all services. To make the services easy to consume and easier to manage, you need to standardize access control, identity management, and other key security aspects. Security is complex, and yet you need your services to be consumable. You need to reduce the variation of security models exposed to the consumer. How can you ensure that priority consumers always have access to the services they require with acceptable response time? What if you need to sacrifice one consumer temporarily in order to sustain another? How do you manage outages, whether planned or not? You need some form of configurable operation control over the service exposure point, enabling you to make adjustments without going through code cycles. These are described in more detail at the beginning of this tutorial: To do all of the above, you need to formally separate the service exposure capability in the architecture as shown by the Service Exposure Gateway in Figure 4. It may not turn out to be a separate runtime component in the final physical architecture, but it at least needs to be clearly delineated in the design. It must be possible to perform the virtualization, visibility, security, and traffic management requirements in a first-class way. Service exposure View image at full size You will notice that we have deliberately not put the common SOA-related term, enterprise service bus ESB , in the diagram shown in Figure 4. This is because opinions vary quite considerably as to what the exact boundaries are for an ESB. ESB is, after all, an architectural pattern, not a component description. Some say it is just the service exposure gateway, others include the integration hub. Some would include the adapters too, and there are many variants in between. There is plenty of literature describing the subtleties of the ESB pattern , but we have ultimately found it better to be simply clear about the specific individual components and their responsibilities. Beyond the runtime components of SOA, there are also the governance aspects. If there are a large number of services, how do you decide and prioritize which functions to expose? How will people discover functions that have been exposed? How do you regulate the variations in the data models used? Some form of catalog of the candidate and current services must be kept to enable governance of the service lifecycle. All of these concerns mount to a recognition that exposing services is not trivial. Yet, to simply expose capabilities as generally available web services leave you wide open to failures in manageability and security. In short, re-use comes at a price, and with that, comes the question of how to fund an SOA? No project with tight deadlines and budget constraints wants to be the one to build a service for the first time â€” at least not properly. Couple with this with the fact that the standards required to enable the SOA concepts were being developed alongside the SOA initiatives, and were, therefore, immature. They were constantly changing while enterprises were attempting to implement them. It is easy to see why some SOAs struggled to gain traction. In many companies, SOA is confined to a specific domain of the business, or there are only a small number of core services actually in play. These took advantage of the maturing client-side scripting capabilities of browsers, and also their ability to perform background HTTP requests using techniques, such as AJAX, to retrieve data without interrupting the user experience with a page reload. It quickly became popular to make more granular data requests and, if possible, change to the JSON data format native to JavaScript as shown in red in Figure 5. Fine-grained exposure for rich browser applications View image at full size Free from the restrictions of the SOAP standards, these interfaces can consider alternative ways to simplify how the "actions" or "operations" to be performed on data were to be represented. In an interesting throwback to some of the earlier roots of the web, the original intent behind HTTP protocol was unearthed. From this, a more simplistic entity based style of interaction was derived. The worldwide web recognizes these verbs in a first class way in order to provide implicit benefits such as caching.

It also uses the URL path to navigate the relationships between the data entities. To carry the same information in SOAP, it might have looked more like the following: However, SOAP has a broader set of standards , which can accomplish many things that these interface cannot. They are used by a different audience and not all of those standards are necessary in that space. There were, at least initially, some limits to the reusability of these new interfaces. Due to the same origin policy implemented by browsers, web page based applications found it difficult though not impossible to make HTTP requests to interfaces offered by other companies. What is a web API? Simplistically and loosely speaking, a "web API" typically refers to functions and data exposed in the following way: However, this definition is over simplistic on a number of levels: In theory, anything that HTTP can respond with could be valid. Not all web APIs are public: As we will see in a later section, APIs are not only exposed and used on the public Internet. However, it is fair to say that the Internet usage has driven much of the agreement on style, usage, and the supporting products and frameworks. They are probably less "RESTful" and more heavyweight to use. They are, therefore, often referred to as "REST" interfaces. However, in reality most only adhere to a subset of the REST recommendations described in the original material on the subject. HTTPS is strongly recommended: Payloads often contain private data, and credentials used to access the web API are usually confidential. So, a new, more lightweight protocol and interaction style is available, but this alone does not warrant a revolution in real-time data integration. What was the trigger for web APIs to come of age? The significant shift came around when smartphones with easy to access "app" stores became mainstream. Mobile application "app" development became commonplace and accessible to a huge audience of developers. With some notable exceptions, apps can rarely do much on their own.

INTEGRATING WITH SOA AND ESBS pdf

2: The Simple Solution to SOA is ESBs?

Blueprint for Successful SOA Integration. by Dain Hansen 03/12/ Abstract. SOA Integration has recently emerged as the de facto standard for successful IT integration; it leverages the benefits of Service Oriented Architecture (SOA) to solve one of the most fundamental challenges IT is facing today.

SOA has grown, changed and evolved, and we work with a number of technologies across industries. Fundamentally, service oriented architecture SOA is a model of infrastructure architecture and an approach to internal application development. Before SOA emerged in the early s, enterprise infrastructures consisted of multiple applications, typically developed in-house to provide a new business service or automate a particular business process. Often, applications for related business processes contained duplicate functionalities, with the same code existing in several internal programs. For example, if multiple programs required credit check information, each of those programs would duplicate the code needed to perform the credit check. These additional code bases resulted in multiple inefficiencies. Code was poorly reused, leading to wasted effort and money spent during development. For example, if a credit check process needed to be changed, multiple developers updated multiple applications, slowing down the entire modification process. These overly complex and poorly designed applications had substantial impact on the top and bottom lines of the organisation. In such an architecture, the emphasis is on creating components called services, which are small, discrete units of software that provide a specific functionality and can be reused in every application. In an SOA model, developers create new applications by orchestrating a collection of services instead of building out an entire software program, eliminating code redundancies across multiple applications. For instance, in SOA a simple bank loan application would be a composite of credit status check services, interest rate services, and customer data services. In short, SOA breaks down the islands of business logic and data that are scattered across multiple, disparate applications. In the New Enterprise era, these silos may exist in on-premise or cloud-based software, SaaS applications, or in devices brought from home by employees. SOA enables interoperability across all silos through integration, making it easier and faster to automate business processes. The benefits of SOA are plenty. By improving the agility of IT systems and business processes, enterprises can better respond to changes in the market and innovate new products to stay competitive. At the same time, they can reduce the bloat and complexity inherent in legacy systems, increase developer productivity by making software design more intuitive, and lower IT costs associated with maintenance and upgrades. Saves money â€” Integrating major applications is often expensive and SOA can save a fortune on integration costs. Improved visibility; improved customer satisfaction â€” Organizing internal software as services makes it easier to expose its functionality externally. This leads to increased visibility that can have business value as, for example, when a courier company makes the tracking of shipments visible to its customers, increasing customer satisfaction and reducing the costly overhead of status enquiries. Easier to make changes; increased agility â€” Business processes are often dependent on their supporting software. It can be hard to change large legacy programs. This can make it difficult to change the business processes to meet new requirements, for example if companies merge or if there are changes to legislation. A service-based software architecture allows for greater organisational flexibility, enabling it to avoid penalties and reap commercial advantage. Much like the traditional EAI tools, an ESB provides salient functionality for messaging, complex event processing, management, routing and mediation. We are experts in helping companies select an ESB. Read more about ESBs and download our whitepaper here. We are not tied to any particular technology, so we can help you find the best fit for your requirements. We partner with market leaders like InterSystems, Oracle and Mulesoft among others. Integrella specialises in IT integration, having delivered successful SOA and integration projects since Our UK-based consultants have experience of all the best-of-breed integration technologies. We live and breathe IT integration, so customers who work with us will have access to experts in this specialist field. Read more â€” click on the images below:

3: Understanding Enterprise Application Integration - The Benefits of ESB for EAI | MuleSoft

Integrella has been working with Service Oriented Architecture (SOA) integration since SOA has grown, changed and evolved, and we work with a number of technologies across industries.

ESBs provide valuable functionality for integrating existing systems and building composite applications, they are not well suited to providing enterprise SOA Federation capabilities. ESBs fall short in the following areas: Ease of use â€" most ESBs are driven by a application development model. They are designed to allow developers to create applications, and manage adapter frameworks to provide broad connectivity to legacy systems. To virtualize an existing service using most ESBs requires complex configuration steps, and in many cases programming tasks, especially if any mediation is required. An SOA Federation solution on the other hand is explicitly designed to facilitate simple federation and virtualization with declarative policy and technology impedance mediation. Policy silos â€" many ESBs provide their own, internally programmed policy models. They can quickly become policy silos creating and enforcing local policies through programming, rather than implementing and enforcing centrally defined enterprise policies. An SOA Federation solution understands standards-based policies for services it exposes and consumes, and offers declarative mediation between the policy capabilities of consumers, the policies enforced by virtual services, and the policy requirements of physical services. No service discovery â€" the current crop of ESBs are service runtimes with no discovery mechanism. SOA Federation solutions include a lightweight, extensible model allowing users to request the federation of a service, and administrators to approve or reject the request. They have no way to offer service consumers specific guarantees of service level, or even grant or deny access to their exposed services based on a contract approval workflow. An SOA Federation solution includes a lightweight, extensible contract workflow allowing potential consumers to request access with specific service level guarantees, and the service provider can grant or deny this request. They often support only a limited subset of the messaging, transport, and policy standards expected in enterprise SOA. Additionally, ESBs will generally not be well suited for DMZ type deployments, so cannot safely expose services outside the enterprise or regional firewall. An SOA Federation solution offers messaging, transport and policy mediation, as well as the ability to virtualize services into secure, DMZ resident intermediaries to remove these technology impedances. Semantic, not syntactic â€" most ESBs offer strong mediation solutions, but it is not the kind of mediation required for service federation. ESBs offer semantic i. An ESB can do a very good job of mapping from one business document to another, but it is not designed to map from one messaging style, standard, transport, or policy to another. SOA Federation solutions provide declarative syntactic mediation capabilities to minimize the impact of administrative, organizational, trust and technology impedances. To achieve this, the SOA Federation solutions offer a set of core capabilities: Uniform Policy Management - Uniform Policy Management ensures consistent policy definition, implementation, enforcement, validation, and audit through all stages of the lifecycle, and across all distributed and mainframe platforms. It ensures that services can be leveraged as first-class citizens throughout an enterprise SOA by complying with enterprise policies that are uniform across all platforms. Service Virtualization - Service Virtualization provides location-transparency, service mobility, impedance tolerance and reliable service delivery without requiring a re-platforming of existing platforms or introducing yet another service platform to support the required solution architecture. Trust and Management Mediation - Trust and Management Mediation ensures interoperability across disparate partners and platforms, trust enablement and trust mediation complementing threat prevention systems. It provides provide last-mile security, metric collection and reporting, SLA monitoring and management, to ensure that services are governed, managed, and secured, and policy implementation and mediation to allow consumers to communicate with a wide range of mission critical business services exposed from any platform. Consumer Contract Provisioning - Consumer Contract Provisioning provides offer, request, negotiation, and approval workflows for service access, capacity, SLA and policy contracts. It ensures that the service provides know which applications and users are consuming their services and allows them to treat different consumers with different priorities and service levels. We will

address the technology solutions required to address these capabilities later in this document. Application Composition â€" most ESBs include process design tools that allow developers to create composite applications tying together different services and applications. An SOA Federation solution is explicitly not in the business of creating new applications and services. The SOA Federation solution can create virtual services from sets of existing services, but it does not provide a mechanism for modeling business logic. They include comprehensive adapter frameworks providing connectivity to a wide range of legacy systems. SOA Federation solutions will not provide this connectivity â€" they will focus on service federation to turn existing services into governed service endpoints. Complex Transformations â€" One of the core capabilities of an ESB is the ability to map one message schema to another, performing complex transformation operations along the way. SOA Federation solutions will offer syntactic transformation, i. Stateful Orchestration â€" ESBs typically provide process orchestration engines built on top of the message-oriented middle backbone of the ESB. These orchestration engines are capable of orchestrating long-running business transactions through stored state. The SOA Federation solution should be able to route and virtualize long-running transactions, but will not offer any form of process orchestration.

INTEGRATING WITH SOA AND ESBS pdf

4: Evolutionary integration with ESBs

An enterprise service bus (ESB) implements a communication system between mutually interacting software applications in a service-oriented architecture (SOA). It implements a software architecture as depicted in the picture.

What is SAP Integration? Companies each day are inundated with SAP application integration challenges. Businesses that do not integrate their SAP systems run the risk of failing to optimize critical business functions, resulting in reduced business agility and business process inefficiencies. Businesses, therefore, are continually looking for ways to integrate multiple stand-alone applications into a single application to reduce integration costs while concurrently focusing on ways to accelerate profitable growth. Also, the SAP landscape has been evolving over the past several years and companies looking to integrate their SAP systems have witnessed the burgeoning of SAP integration applications that have transformed the way businesses are controlling all aspects of their operations. This blog post, thus, highlights some of the common trials faced when integrating SAP and provides an overview of SAP interface and integration technologies, and discusses the many approaches for solving SAP integration challenges. This standard data format serves as a medium for transferring master data to and from SAP, and is typically used for asynchronous transactions. With IDOCs, you can retrieve a multitude of information such as information pertaining to suppliers, cost centers, and logistics such as bills of materials. While IDOCs are typically used for the asynchronous transaction, BAPI, on the other hand, is used in synchronous scenarios where two-way communication is required. For example, if an organization intends to manage or regulate its cost centers from an external application, they can do by obtaining a list of profit or cost centers or create new ones using a BAPI object. Approaches to SAP Integration This section illustrates the key approaches to integrating with SAP and also highlights the many positives and negatives of each approach. Developers are always on the lookout for ways to expedite integration with SAP systems and the point-to-point integration architecture is one of the approaches that offer rapid point-to-point integration between two specific applications. Moreover, this integration architecture can only support tight coupling with SAP, thereby making it a less flexible approach since the SAP landscape is prone to changes. As business processes evolve and new integration scenarios are required, the point-to-point integration architecture fails to offer additional touch points and tight dependencies. This significantly reduces business agility and is generally not the ideal long-term solution. SOA stacks comprise multiple products, including â€” but not restricted to â€” application servers, enterprise service bus, orchestration engines, and management and development tools. One of the many benefits of implementing the SOA components is that it creates a robust integration architecture that supports most use cases. Applications are loosely coupled in the SOA landscape and, in case changes are needed, this enterprise solution effectively addresses them. This integration architecture is also responsible for reducing application maintenance costs and is considered a more reliable platform than the point-to-point and hub-and-spoke approaches. Once the SOA integration framework is implemented, the cost of procuring additional applications for supporting new business processes and the complexity that comes with it is considerably reduced. The SOA approach, thus, allows organizations to create, consolidate, and deploy services, thereby catering to varying business needs. On the downside, SOA stacks involve a multitude of products which must all be deployed and configured. Also, implementation of an SOA can take multiple years and can involve higher upfront costs. This, in turn, can negatively impact the application development process. On the other hand, all developers need to be trained on proprietary software and recruiting new developers can pose a serious challenge to the requirement for specialist knowledge skyrockets. As a result, many organizations are still struggling to solve the SAP integration challenges. Stand-alone ESBs provide their own management tools or integrate with the management tools used by an organization. In other words, an ESB works as a unifier where it unites the numerous ways in which components can receive or send information to other application elements. Moreover, they also utilize development tool and technologies that users are already familiar with. Thus, the approaches outlined above allow organizations to meet their business objectives and accomplish business goals within a short span of time. HokuApps System Integration Platform: The SAP integration platform

offers businesses integration services that are secure, reliable, and delivered and managed by SAP. HokuApps SAP supports integration between processes and data between on-premise, Cloud applications, and other external third-party applications. This, in turn, can eliminate all challenges pertaining to training, staffing, and other development costs and enhance the speed at which organizations can develop and deliver applications and that will eventually streamline and automate business processes and also provide an interconnected landscape that will help organizations gain a competitive edge. When compared to other alternatives, HokuApps comparatively expedites the SAP-application integration process, thereby creating new business opportunities and simultaneously laying the foundation for future projects. With HokuApps, businesses can thus expect a higher long-term return on investment.

5: Integrella Expertise: Service Oriented Architecture (SOA)

These examples provide both working code as well as suggest a methodology of evolutionary integration which can be used to dramatically simplify and accelerate SOA integration. BT About InfoQ.

However, we still need a lot code to make things work. For example we need a Servlet bootstrapping the whole thing, some timers to poll the mail server and as mentioned before we need some data helper classes when converting email messages to POJOs and finally some property classes to store miscellaneous stuff like usernames and passwords. For now these components are left out, and provided for you to download see resources section. The same goes for the implementation of all the interfaces listed above. It is well organised, it uses Spring to do the wiring, it uses a handful of design patterns, etc, etc. Thus when we look at the layering of Powder Alert 1 it is a decent application. But when it comes to being flexible regarding its integration towards other systems, evolving new services, and scaling, it looks like the beginning of a Coral Reef. This is very typical situation which we have identified at numerous projects that are doing some kind of integration with other systems. The competence covering the system column is very good, but required resources due to distributed computing and integration are either not present or ignored. To be able to create applications that cope with changing requirements and evolving portfolios of services, we need a platform or tool that can act as the foundation that able to face these challenges. To be more precise, a collection of tools would be preferable, more like a LeathermanTM. Meeting Ross at the Afterski Ladies and Gentlemen, let us proudly present Mule is a messaging platform based on ideas from ESB architectures. In short, it is a light-weight messaging framework that uses your existing technology infrastructure Jms, Web Services, Email, etc to build composite service applications. A key characteristic of the Mule design is to be as flexible and extensible as possible, it can be thought of as the Spring framework for integration. The Mule framework provides a highly scalable service environment in which you can deploy your business components. It manages all the interactions between components transparently whether they exist in the same VM or over the internet and regardless of the underlying transport used. Your services and logic are not infected, and will not suffer a framework lock in. This means that it is well suited for testing and downscaling during development. This is what we call environment transparency, and is one of the main features of Mule. It enables the snowdudes to develop their entire application on a laptop on their way up the mountains, and when they finally get online, just dropping newly developed code onto a server, change the deployment configuration and run it in a distributed environment if needed. A complete feature list and lots of good introductory material can be found at the Mule site. Leveraging Mule Okay, we got hold of a powerful collection of tools, but what we need now is some kind of user manual that actually tells us what tool is the best for the job. We need to know that we are not literarily using a sledgehammer to hammer in nails. It should be of no big surprise that as other computer science disciplines somebody has faced the same challenges before, and has collected these best practices in the form of patterns. So to help us out with these Integration problems we have hold your breath.. It is highly recommended to read this book. It is really good. It lists all patterns from the book, along with other useful information like blogs and articles. We got the Integration patterns lined up between those two "Application"-bubbles at each end of the Architecture Overview figure. First we got something called "Channel". The next pattern coming up, for those who can read sideways, is hidden in the "Message Receiver" name. A Message Receiver is used to read or receive data from the Application. In Mule a Receiver is just one element of a Transport provider and Mule provides many transports such as jms, soap, http, tcp, xmpp, smtp, file, etc. The main purpose of a Message Endpoint is to connect an application to a messaging channel. By doing this encapsulation we achieve messaging transport transparency. A Message Endpoint is a specialized Channel Adapter that has been custom developed for and integrated into its application. The next thing coming up is a "Connector". The Message Receiver is coupled to this thingy to be able to communicate in the Mule-way of walking and talking. The connection takes channel specific requests in one end, and connects to Mule components in the other end talking in UMOEvents. Again, for those who can read sideways, we got the "Transformers" box. The main purpose of the Message Translator is to enable systems using different data

formats to be able to communicate with each other using messaging. The knowledge gained by reading this intro serves as good primer for the problem area. Back to Mule, Transformers are used to transform message or event payloads to and from different types. Mule does not define a standard message format though Mule can support standard business process definition message types. Data transformation is very subjective to the application and Mule provides a simple yet powerful transformation framework. Next in line is the "Inbound Router". A Message Routers main purpose is to consume messages from one Message Channel and republish them to different Message Channels depending on a set of conditions. When it comes to the concrete implementation of the Message Router, Inbound Router, it can be used to control and manipulate events received by a component. Typically, an inbound router can be used to filter incoming events, aggregate a set of incoming events or re-sequence events when they are received. Now we have done more or less all the secrets handshakes to be able to reach the soul of the Mule: The UMO Component Central to the architecture of Mule are single autonomous components that can interact without being bounded by the source, transport or delivery of data. These components are called UMO Components and can be arranged to work with one another in various ways. These components can be configured to accept data a number of different sources and can send data back to these sources. This is your client code that actually does something with the events received. Mule does not place any restrictions on your object except that if configured by the Mule directly, it must have a default constructor if configured via Spring or Pico, Mule imposes no conventions on the objects it manages. It is pretty obvious that UMO components are powerful and flexible stuff. Much like kids really. Full of vitality, creativity, and speed Imaging that when you are at work your have to put your kids in a store filled with Venetian glass from the 17th century. Guess you would be kind of nervous then So, with the UMO components as the kids, a kindergarten could be a good idea. They are shown in Figure 9 below. The Mule Manager sees to that all UMO components get there brotherly share of the resources, and ensures that there is no fighting amongst them. It manages how the kindergarten is conducted, how many kids on each department, the opening hours, etc, etc. In Mule terms, the Mule Manager manages the configuration of all core services for the Model and the components it again manages. Wow, that was a lot of management, folks! We must not forget the other nanny in the Mule kindergarten, The Model. The Model deals with our UMO kids on a day to day basis. It comforts them, feeds them, protects them from the hostile environment surrounding the kindergarten, and ensures that each of the UMO kids are playing with equals. The first control mechanism is the Entry Point Resolver. An Entry Point Resolver is used to determine what method to invoke on an UMO component when an event is received for its consumption. The Lifecycle Adapter is the second control mechanism used by the Mule Model. It is responsible for mapping the mule component lifecycle to the underlying component. As the last control mechanism we got the Component Pool Factory. This mechanism is responsible for creating proper component pools for a given UMO component. Thus departments for our UMO kids Back two our two lazy heroes. How and where should they apply Mule to their PowderAlert application? The Shift towards Mule Figure Mule-fied PowderAlert, first version Figure 10 above, shows a simplified sketch of how the first version of PowderAlert looks in a Mule costume. In this version, we use the VM option in Mule as a simple communication strategy. No messaging is involved The later articles in this series will introduce messaging as well. So, to explain how Mule is leveraged into PowderAlert we need to have a closer look at the architecture and the components of that makes up the application. Initially, you need to determine the services your application provides and consumes, and whether the services can be classified as external or internal. The rationale for doing this exercise is to determine options available for how the particular service should be integrated into Mule. In this context, the classification of the service determines to what extent it is possible to alter services in your architecture. One end of the scale we have internal services that you have developed yourself that can be altered with no impact on other services. The other extreme are external services that have given interfaces that must be accepted as is, with absolutely no possibility to be altered. By applying this definition to Mule it can be stated in general terms that the number of integration options of how to integrate a given service with Mule are larger with internal services than external services. Thus an internal service can be very tightly integrated with Mule, and if you fill out the right application form it can potentially be a UMO-kid in the Mule kindergarten! If we take a step back and have another look at

INTEGRATING WITH SOA AND ESBS pdf

Figure 1 above, we can, generally speaking, divide the PowderAlert application into three grain coursed services: The way they talk and walk are fixed. Thus they can be classified as external services;as a result, very tight integration with Mule would not really give us what we want. We need something that can connect to the services interface, and is able to transform the payload from component specific information elements to UMO information objects. If we take a closer look at Figure 7 above, the Endpoint component stands out as a candidate for integrating the mailserver and the SkiInfo site towards Mule. As the figure shows, the SkiInfo site communicates with PowderAlert by sending emails, thus only one Transport is needed to handle both the communication with the PowderAlert users and the SkiInfo site. We have in this case one technical integration point Email , but the semantics in the payload are different for SkiInfo and PowderAlert messages.

INTEGRATING WITH SOA AND ESBS pdf

6: Shaping Your Middleware Strategy To Benefit From ESBs

Part 1 introduces SOA and ESB concepts, and discusses the ESB capabilities of WebSphere Business Integration Message Broker V5 and WebSphere Application Server V6. It describes guidelines for determining when integration of ESBs is necessary, and describes patterns for integrating ESBs.

In a nutshell, EAI is an approach, or more accurately, a general category of approaches, to providing interoperability between the multiple disparate systems that make up a typical enterprise infrastructure. Enterprise architectures, by their nature, tend to consist of many systems and applications, which provide the various services the company relies upon to conduct their day to day business. A single organization might use separate systems, either developed in-house or licensed from a third party vendor, to manage their supply chain, customer relationships, employee information, and business logic. This modularization is often desirable. In theory, breaking the task of running a business into multiple smaller functionalities allows for easy implementation of the best and newest technological advancements in each area, and quick adaptation to changing business needs. However, to gain the benefits of this kind of distributed, modular system, an organization must implement technologies that deal with the problems presented by this architecture: Robustness, Stability, and Scalability: Because they are the glue that holds together a modular infrastructure, integration solutions must be highly robust, stable, and scalable. In a point-to-point integration model, a unique connector component is implemented for each pair of applications or systems that must communicate. When used with small infrastructures, where only two or three systems must be integrated, this model can work quite well, providing a lightweight integration solution tailor-made to the needs of the infrastructure. However, as additional components are added to an infrastructure, the number of point-to-point connections required to create a comprehensive integration architecture begins to increase exponentially. A three-component infrastructure requires only three point-to-point connections to be considered fully integrated. By comparison, the addition of just two more components increases this number to 10 connectors. This is already approaching an unmanageable level of complexity, and once an infrastructure includes 8 or 9 component systems, and the number of connections jumps into the 30s, point-to-point integration is no longer a viable option. Remember that each of these connectors must be separately developed and maintained across system version changes, scalability changes, and more or, in some cases, even purchased at high cost from a vendor , and the unsuitability of point-to-point integration for complex enterprise scenarios becomes painfully clear. The EAI Approach To Integration To avoid the complexity and fallibility of integrating complex infrastructures using a point to point approach, EAI solutions use various models of middleware to centralize and standardize integration practices across an entire infrastructure. Rather than each application requiring a separate connector to connect to every other connector, components in an EAI-based infrastructure use standardized methods to connect to a common system that is responsible for providing integration, message brokering, and reliability functionalities to the entire network. In other words, EAI solves the problem of integrating modular systems by treating integration as a task for a system, like any other task, rather than a snarled mess of brittle connections. EAI systems bundle together adapters for connectivity, data transformation engines to convert data to an appropriate format for use by the consumer, modular integration engines to handle many different complex routing scenarios simultaneously, and other components to present a unified integration solution. EAI loosens the tightly coupled connections of point-to-point integration. This allows for a more flexible architecture, where new parts can be added and removed as needed, simply by changing the configuration of the EAI provider, and simplified modular development, where a single service can be re-used by multiple applications. Many modern EAI approaches also take advantage of the opportunity presented by adding a central integration mechanism to further consolidate messaging tasks. Traditional EAI The first EAI solutions on the market took the idea of unifying integration literally, and incorporated all the functionality required for integration into central hubs, called brokers. The Broker Model In a broker approach to EAI, a central integration engine, called the broker, resides in the middle of the network, and provides all message transformation, routing, and any other inter-application functionality. All communication between

applications must flow through the hub, allowing the hub to maintain data concurrency for the entire network. Typically, implementations of the broker model also provide monitoring and auditing tools that allow users to access information about the flow of messages through their systems, as well as tools to speed up the complicated task of configuring mapping and routing between large numbers of systems and applications. Advantages Like all EAI approaches to integration, the broker model allows loose coupling between applications. This means that applications are able to communicate asynchronously, sending messages and continuing work without waiting for a response from the recipient, knowing exactly how the message will get to its endpoint, or in some cases, even knowing the endpoint of the message. A broker approach also allows all integration configuration to be accomplished within a central repository, which means less repetitive configuration. Disadvantages Like any other architecture model that uses a central engine, is that the broker can become a single point of failure for the network. Under heavy load, the broker can become a bottleneck for messages. A single central destination for all messages also makes it difficult to use the broker model successfully across large geographical distances. This can present problems if your integration scenario involves products from several vendors, internally developed systems, or legacy products that are no longer supported by the vendor. The lack of clear standards for EAI architecture and the fact that most early solutions were proprietary meant that early EAI products were expensive, heavyweight, and sometimes did not work as intended unless a system was fairly homogenous. The effects of these problems were amplified by the fact that the broker model made the EAI system a single point of failure for the network. A malfunctioning component meant total failure for the entire network. In , one study estimated that as many as 70 percent of integration projects ultimately failed due to the flaws in early broker solutions. While it still used a central routing component to pass messages from system to system, the bus architecture sought to lessen the burden of functionality placed on a single component by distributing some of the integration tasks to other parts of the network. These components could then be grouped in various configurations via configuration files to handle any integration scenario in the most efficient way possible, and could be hosted anywhere within the infrastructure, or duplicated for scalability across large geographic regions. The Enterprise Service Bus Is Born As bus-based EAI evolved, a number of other necessary functionalities were identified, such as security transaction processing, error handling, and more. Rather than requiring hard-coding these features into the central integration logic, as would have been required by a broker architecture, the bus architecture allowed these functions to be enclosed in separate components. The ultimate result - lightweight, tailor-made integration solutions with guaranteed reliability, that are fully abstracted from the application layer, follow a consistent pattern, and can be designed and configured with minimal additional code with no modification to the systems that need to be integrated. Despite their differences, most ESBs include all or most of the following core features, or "services": A way of centrally configuring endpoints for messages, so that a consumer application does not require information about a message producer in order to receive messages Transformation: The ability of the ESB to convert messages into a format that is usable by the consumer application. Similar to the transformation requirement, the ESB must be able to accept messages sent in all major protocols, and convert them to the format required by the end consumer. The ability to determine the appropriate end consumer or consumers based on both pre-configured rules and dynamically created requests. The ability to retrieve missing data in incoming messages, based on the existing message data, and append it to the message before delivery to its final destination. The goal of ESB is to make integration a simple task. As such, an ESB must provide an easy method of monitoring the performance of the system, the flow of messages through the ESB architecture, and a simple means of managing the system in order to deliver its proposed value to an infrastructure. ESB security involves two main components - making sure the ESB itself handles messages in a fully secure manner, and negotiating between the security assurance systems used by each of the systems that will be integrated. If an organization knows that they will need to connect additional applications or systems to their architecture in the future, an ESB allows them to integrate their systems right away, instead of worrying about whether or not a new system will not work with their existing infrastructure. When the new application is ready, all they need to do to get it working with the rest of their infrastructure is hook it up to the bus. Unlike broker architectures, ESB functionality can easily be dispersed across a geographically distributed

network as needed. Additionally, because individual components are used to offer each feature, it is much simpler and cost-effective to ensure high availability and scalability for critical parts of the architecture when using an ESB solution. This means that an organization seeking to migrate towards an SOA can do so incrementally, continuing to use their existing systems while plugging in re-usable services as they implement them. At first glance, the number of features offered by the best ESBs can seem intimidating. The large number of modular components offers unrivaled flexibility that allows incremental adoption of an integration architecture as the resources become available, while guaranteeing that unexpected needs in the future will not prevent ROI. For this reason, making informed decisions about your EAI strategy is vital to the success of your integration initiative. How many applications do I need to integrate? Will I need to add additional applications in the future? How many communication protocols will I need to use? Do my integration needs include routing, forking, or aggregation? How important is scalability to my organization? The article includes an expanded version of the checklist above, to help determine whether or not ESB is a good match for their integration needs. Want to know more about the future of Mule as an ESB? The Mule development team has released new Development features to the platform, including:

INTEGRATING WITH SOA AND ESBS pdf

7: 5 reasons why ESB-led integration is out and API-led integration is in | MuleSoft Blog

Applying the laws of physics to SOA, then, it stands to reason that enterprise service buses (ESBs) are the simplest path to SOA within organizations -- and perhaps in many cases, the right path.

In order to create the new products, applications, and services that businesses need to confront the digital revolution, you have to connect systems, data, and devices. Enterprises struggled with point-to-point integrations , which created a spaghetti-like mess of code, and then, for many years, companies found a better integration solution with ESBs Enterprise Service Bus and an ESB integration approach. In addition, ESB integration provides a way to leverage your existing systems and expose them to new applications. For now, here are the top 5 challenges with an ESB-Led approach. ESB Integration presents organizational challenges ESBs face procedural and political bottlenecks as different parts of the business need to move at different speeds. Gartner refers to this as PACE layering; they recognize that business capabilities â€" together with their supporting applications â€" support innovation, differentiation, and operationsâ€"â€"all of which need to change at different speeds. An ESB can often become the bottleneck to implementing changes, while maintaining service to the rest of the business. For example, HR needs to implement new legislation updates, Finance wants to implement new tax measures, and Asset Management needs to acquire and onboard new warehouse capacity. ESB Integration encourages a monolithic architecture with high cost ESBs can be very complex; they comprise multiple products, require significant hardware resources, have many different tools, and require specialist skills and experience which are hard to source and maintain. The cost of ESB infrastructure, implementation, and ongoing costs is high; so high, in fact, that very few customers will be able to afford multiple ESB instances. In addition, ESBs can be a single point of failure and a single point of outage, especially when upgrades are required. These are all capabilities which ESBs were not designed to deliver and, until today, still struggle to provide. Purchasing additional cloud integration tools is an option, but not a good one. This can prevent your business from taking advantage of cloud and SaaS applications. ESBs treat all integrations as equals, but they should be built with specific purposes in mind Integrations fall into three broad categories: Device integrations â€" to support multiple channels; for example, mobile, web, and partners. Process integration â€" these integrations provide orchestration logic to tie together integrations that span multiple back-end systems. System integration â€" these provide connections to our applications. All of these integration categories behave differently and have different requirements. While ESBs will often support process and system capabilities, they rarely provide direct support for device integrations. Customers must, therefore, purchase additional products such as an API Gateway , increasing costs and complexity and, in turn, introducing monitoring, management, and security gaps as integrations flow backwards and forwards between multiple products. Mixing integration categories also causes problems with ESB integrations. For example, including channel specific logic with the process integration makes it difficult to introduce support for new channels and devices. Mixing process and system logic impede easy access to backend applications and sharing of orchestration logic. ESB integration treats all integrations as equalsâ€"â€"reducing agility and flexibility, while increasing governance overhead. ESB integration does not deliver agility and innovation ESBs were designed before the cloud, before the mobile explosion, before social media, and before the Internet of Things. Most importantly, ESBs were designed before the enormous pressure to innovate and move faster aroseâ€"â€"triggered by new technologies, rising consumer expectations , and business disruptors. ESBs have had their time, but evolving enterprise technology requires a different integration approach. Take a look at how businesses in every industry have met their digital transformation goals by adopting an API-led approach to connectivity.

8: Virtual ESB - SOA Federation

ESBs are built into the SOA "middleware" to overcome integration problems between incompatible systems. One of the systems may be a slow receiver, another may need messages in binary format.

Using business intelligence BI oriented ETL processes, businesses extract data from highly distributed sources, transform it through manipulation, parsing, and formatting, and load it into staging databases. From this staging area data, summarizations, and analytical processes then populate data warehouses and data marts. Historically, the primary use for ETL tools has been to enable business intelligence. Pulling databases, application data and reference data into data warehouses provide businesses with visibility into their operations over time and enable management to make better decisions. Data integration allows companies to migrate, transform, and consolidate information quickly and efficiently between systems of all kinds. ETL tools reduce the pain of manually entering data and allow dissimilar systems to communicate, all the while supplying a unified view. Its success can be attributed to its cross functionality, reusable components, and automatable processes. Optimized for moving large amounts of data in a batch-oriented manner, PowerCenter and similar ETL tools have been used to integrate enterprise applications across heterogeneous environments. These databases provide the foundation for the operational systems and applications that run the business. As operations increasingly required these systems to integrate with each other, existing ETL tools provided an obvious solution. Already supporting data-level connectivity to many popular databases and applications, ETL tools became a quick and seemingly simple means of connectivity and data movement. ETL Tools Get Complicated ETL tools indeed provide a method of communication between databases and applications, but pose significant challenges over time. Because creating this type of connectivity requires an comprehensive knowledge of each operational database or application, interconnectivity can get complicated as it calls for implementing very invasive custom integrations. Over time, this approach grows increasingly complex, and the greater the number of interconnected systems, the more complicated things become. Moreover, with such tight coupling, interdependencies create the potential for big, unpredictable impacts when even the slightest changes are made. As the IT landscape transitions to the cloud, lack of visibility into the internals of cloud databases and applications often make it impossible to implement ETL-based integrations. Moreover, the transition to the cloud means greater value is placed on real-time integration updates, something ETL tools cannot easily deliver, as they are primarily batch-oriented. With APIs, developers can access endpoints and build connections without having in-depth knowledge of the system itself, simplifying processes tremendously. As ETL tools remained focused more towards BI and big data solutions, and as traditional operational data integration methods become outdated with the rise in popularity of cloud computing, ESBs become better options to create connectivity. Unlike traditional ETL tools used for data integration, an ESB isolates applications and databases from one another by providing a middle service layer. This abstraction layer reduces dependencies by decoupling systems and provides flexibility. Developers can utilize pre-built connectors to easily create integrations without extensive knowledge of specific application and database internals, and can very quickly makes changes without fear of the entire integrated system falling apart. Shielded by APIs, applications and databases can be modified and upgraded without unexpected consequences. In comparison to utilizing ETL tools for operational integration, an ESB provides a much more logical and well defined approach to take on such an initiative. The next generation platform consists of a set of products that help businesses connect SaaS, cloud, mobile, and on-premises applications and services as well as data sources. Working with numerous other components, Anypoint Platform delivers a robust integration solution for businesses: A graphical interface eliminates the need for intricate manual coding. Simply use the graphical drag and drop interface of Anypoint Studio to accelerate development productivity. MuleSoft offers numerous solutions to help businesses overcome integration challenges and transform themselves into an efficient New Enterprise in which numerous disparate systems and applications can seamless communicate Typical ETL tools are limited in their ability to serve the requirements of the New Enterprise, but an enterprise service bus can provide real-time, high-performance, scalable operation

capabilities.

INTEGRATING WITH SOA AND ESBS pdf

9: From ETL Tools to ESBs | MuleSoft

SOA Integration: Service-Enable Your Business Logic for Greater Agility. by Dain Hansen 05/16/ The BEA SOA Integration solution connects applications, processes, and data sources into reusable services that can help organizations improve customer service, add new revenue streams, and streamline operations.

Overview[edit] The concept is analogous to the bus concept found in computer hardware architecture combined with the modular and concurrent design of high-performance computer operating systems. The motivation for the development of ESB was to find a standard, structured, and general purpose concept for describing implementation of loosely coupled software components called services that are expected to be independently deployed, running, heterogeneous, and disparate within a network. ESB is also a common implementation pattern for service-oriented architecture. Working[edit] An ESB applies the design concept of modern operating systems to independent services running within networks of disparate and independent computers. Like concurrent operating systems an ESB provides commodity services in addition to adoption, translation and routing of client requests to appropriate answering services. The primary duties of an ESB are: Route messages between services Monitor and control routing of message exchange between services Resolve contention between communicating service components Control deployment and versioning of services Marshal use of redundant services Provide commodity services like event handling, data transformation and mapping, message and event queuing and sequencing, security or exception handling , protocol conversion and enforcing proper quality of communication service Ambiguous use of the term ESB in commerce[edit] There is no global standard for enterprise service bus concepts or implementations. The implementations of ESB use event-driven and standards-based message-oriented middleware in combination with message queues as technology frameworks. History[edit] The first published usage of the term "enterprise service bus" is attributed to Roy W. It should be noted that Schulte asserted that: Service - denotes non-iterative and autonomously executing programs that communicate with other services through message exchange Bus - is used in analogy to a computer hardware bus Enterprise - the concept has been originally invented to reduce complexity of enterprise application integration within an enterprise; the restriction has become obsolete since modern Internet communication is no longer limited to a corporate entity ESB as software[edit] This section does not cite any sources. Please help improve this section by adding citations to reliable sources. Unsourced material may be challenged and removed. October Learn how and when to remove this template message The ESB is implemented in software that operates between the business applications, and enables communication among them. Ideally, the ESB should be able to replace all direct contact with the applications on the bus, so that all communication takes place via the ESB. To achieve this objective, the ESB must encapsulate the functionality offered by its component applications in a meaningful way. This typically occurs through the use of an enterprise message model. The message model defines a standard set of messages that the ESB transmits and receives. When the ESB receives a message, it routes the message to the appropriate application. Often, because that application evolved without the same message model, the ESB has to transform the message into a format that the application can interpret. A software adapter fulfills the task of effecting these transformations, analogously to a physical adapter. If the message model does not completely encapsulate the application functionality, then other applications that desire that functionality may have to bypass the bus, and invoke the mismatched applications directly. Doing so violates the principles of the ESB model, and negates many of the advantages of using this architecture. The beauty of the ESB lies in its platform-agnostic nature and the ability to integrate with anything at any condition. Therefore, the challenges and opportunities for EAI vendors are to provide an integration solution that is low-cost, easily configurable, intuitive, user-friendly, and open to any tools customers choose. ESB hive of commodity components Most observers[who?

INTEGRATING WITH SOA AND ESBS pdf

Forms of college writing Israeli-Palestinian conflict National Geographic Investigates: Ancient India Topics in palliative care Traditional Korean Furniture Searching for design concepts Chapter 44-Principles of Pain Management 327 The revolution will not be funded full German Demystified A locally most powerful rank test for the location parameter of a double exponential distribution. Approaches to Modern Judaism II Physical activity and obesity Savanna Plants of Africa Tempesta di William Shakespeare Nella Popl Annual Symposium on Principles of Programming Languages (ACM Sigplan Notices,) I: KEY OBSERVATIONS ON GALACTIC COSMIC RAYS Turn scanning: experimental and theoretical approaches to the role of turns Business to business marketing definition Payment systems in the us book Water resources engineering ralph a wurbs wesley p james Reliable acute care medicine. Exploring the Christian way Child welfare problem Per and the Dala horse Submission and the non-believing husband Sacred places and sacred space Vba for modelers developing decision support systems Lean, Long Strong The role of ethics in business curricula Duane Windsor The adventures of Tom Sawyer and the great brain : liminality, ritual, and race in the construction of th 31st IEEE International Symposium on Multiple-Valued Logic: Proceedings The amazing world of James Hector Amazing Nellie Bly. 11th Cat, Vol. 5 (11th Cat) Straight Down from Heaven Honda odyssey 2003 service manual U00a7 63. The French Declaration of Faith, A.D. 1872 498 Yes Is Better Than No Spices and herbs health benefits Healthy, active and outside!