

# UDDI and Beyond

The Role of Service Registries and  
Repositories in Your SOA



**TABLE OF CONTENTS**

1	SERVICE REGISTRIES PROMOTE ASSET REUSE IN AN SOA.....	4
2	UDDI PROVIDES STANDARDS- BASED WEB SERVICES INDEXING.....	4
3	BENEFITS OF SOA DEPEND ON SYSTEM CONFIGURATION .....	6
4	MIDDLEWARE OFFERS MAXIMUM FLEXIBILITY FOR MAXIMUM BENEFIT.....	7
5	POINT-TO-POINT CONFIGURATIONS HAMPER PERFORMANCE, SCALABILITY AND RELIABILITY .....	9
6	TIBCO SOFTWARE PROVIDES CHOICE .....	10
7	ABOUT TIBCO .....	13

**Executive Summary** As the pace of business accelerates, enterprises are increasingly responding to customer demands in real time. To accomplish this feat, many IT departments have turned to service-oriented architecture (SOA). SOA is designed to help decrease development expenses and risk of application failures, increase asset reuse and enhance business agility.

SOA is a way of building enterprise systems out of reusable business logic that perform discrete functions and can be reused across the organization for different purposes, allowing developers to create composite applications by invoking and orchestrating multiple services, events and models so they collectively perform a higher-order business process.

The benefits of using service registries and repositories in an SOA environment are compelling:

- **Increased asset reuse.** Asset reuse can help increase the flexibility of a company's IT infrastructure by reducing reliance on monolithic applications, which are not easily modified or reused in new ways.
- **Enhanced compliance.** Many companies currently lack the visibility to determine their level of compliance with external regulations and internal business guidelines. However, because assets are listed in registries with the SOA approach, SOA can make it easier for administrators to keep an accurate inventory of existing assets, collect metrics and enforce operational policies.
- **Improved efficiency.** Web services can help enforce coding standards, facilitate communication and minimize the duplication of efforts within large, geographically dispersed organizations.

However, many of the potential benefits of an SOA depend on the way it is implemented. Because an SOA environment consists of multiple "services" that can be configured to interact in a variety of ways, it is critical that IT architects choose elements and mechanisms that maximize service performance, efficiency and scalability.

This paper will explore different ways to register and discover services, as well as manage the services lifecycle. It will outline two potential approaches for configuring an SOA environment and explain TIBCO's recommended approach.

## 1. Service Registries Promote Asset Reuse in an SOA

Because an SOA is based on discrete reusable services such as Web Services Description Language (WSDL) service descriptions, Business Process Execution Language (BPEL) process descriptions or Extensible Stylesheet Language Transformation (XSLT) maps, complexities of scale and distribution are introduced that extend far beyond integration of monolithic applications. This situation could result in chaos unless managed and organized correctly.

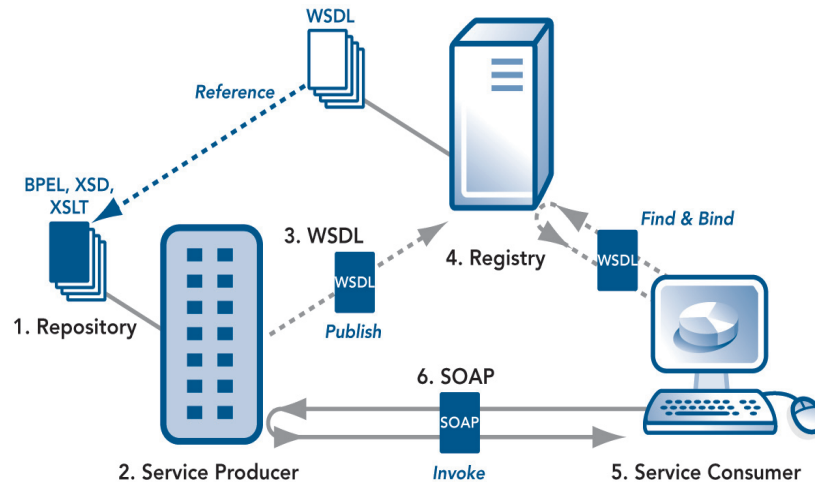
By creating centralized metadata registries, companies can make it easier for developers to find and reuse existing assets. A service registry is an index of service interfaces in the SOA. It functions as a link to concrete implementation definitions of the services but does not necessarily contain the definitions themselves. In its function as a catalog, the service registry must provide the means to publish new services to the catalog and browse and search the catalog for reusable services, as well as a mechanism for classifying services so they can be relevant to the usage. These registries help prevent duplication of efforts and promote efficiency. Universal Description, Directory and Integration (UDDI) is the Web services standard for SOA registries.

A service repository, in contrast, holds the contents of the service implementation definitions such as Java code, BPEL processes, WSDL descriptions or XSLT transformations. In addition to the contents of the service definition, the repository frequently holds metadata such as dependencies between services (for example, XSLT maps invoked from a BPEL business process) and version management mechanisms.

## 2. UDDI Provides Standards-Based Web Services Indexing

Designed to locate, invoke and manage metadata about services, UDDI is probably the least known of the three core Web services standards. Along with its better known cousins WSDL and SOAP, UDDI enables loosely coupled architectures that are the hallmark of an SOA (see Figure 1). The UDDI specification also describes a set of Web services application program interfaces (APIs) for publishing, searching, change notification and replication between registries (see Figure 1).

Figure 1. UDDI as part of an SOA implementation



UDDI provides two key functions:

- **Location independence.** Instead of forcing applications to hardcode the API of their target services, UDDI provides a mechanism to dynamically look up the service interfaces and locations at runtime.
- **Taxonomy.** UDDI supports classification based on service categories and functions as a search engine for Web services. Well-defined service classifications can help deliver more relevant results than can searches conducted solely by name.

The basic UDDI information model consists of a hierarchy of four basic data types:

1. **businessEntity.** At the highest level of granularity is the businessEntity—for example, a bicycle repair shop.
2. **businessService.** The businessEntity can contain many categories of businessServices—for example, the bike shop may offer repair services, new bicycles and cycling gear.
3. **bindingTemplate(s).** Contained by businessServices, bindingTemplates contain technical information about the operation of a service.
4. **tModels.** These technical models are the UDDI structure for representing the namespace for finding and identifying a service, as well as a technical

fingerprint of descriptive information for invoking the service. tModels are referenced, but not contained, by bindingTemplates. An example of a tModel as a namespace is a “repair chain” service in a custom taxonomy. As a technical fingerprint, a tModel may be a WSDL specification that describes the interface of that service and its invocation mechanism.

### 3. Benefits of SOA Depend on System Configuration

There are two options for configuring an SOA environment. First, service consumers and producers can communicate directly in a point-to-point (or peer-to-peer) manner. In the second approach, a service mediator can act as an enterprise service bus (ESB) between service consumers and producers. TIBCO strongly recommends a mediated approach, in which all service requests are processed through an ESB, because this can be more flexible, scalable and reliable than a point-to-point configuration.

To understand the distinction between direct and mediated communication, it is helpful to consider a real-world example. Suppose the chain on your bicycle broke and you needed to have a new one put on. In this case, the service would be chain repair—and to access the service, you need to call the bike shop. In a direct lookup scenario, you might consult the Yellow Pages and find the shop under “Bicycle repair,” then call the shop directly to request service. This method corresponds to the point-to-point configuration described above. You are the service consumer, the bike shop is the service producer and the Yellow Pages is the registry that gives you information on how to request the service.

However, if the bike shop moves to a new location and gets a new phone number, its Yellow Pages entry is no longer useful. The same thing happens in an SOA environment when a service producer changes the location of a service without updating the registry—service consumers will be unable to access the service until its record is updated. Even then, consumers will find the updated record only if they check the Yellow Pages each time they make a service request. However, frequent checking of the registry does not scale well. For that reason, UDDI is not typically used at runtime.

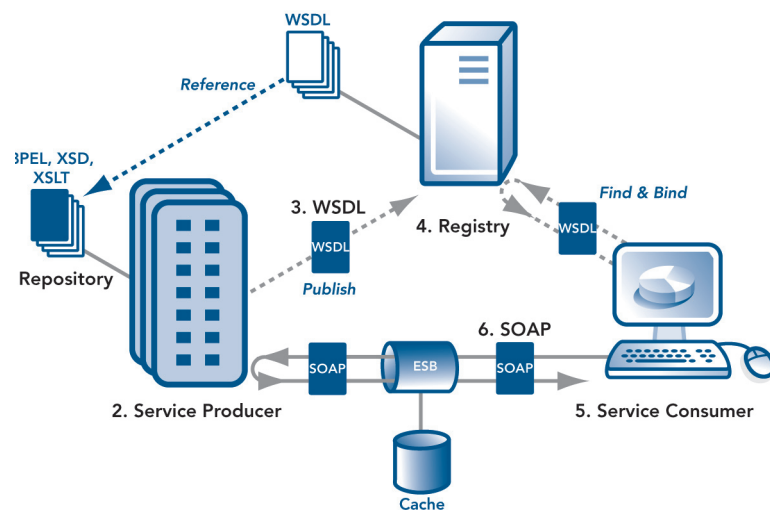
In a mediated approach, the bike shop would have an 800 number that is configured to forward calls to its local number (even if that number changes). In this example, the 800 number would act as an ESB to help ensure that service

consumers can find the appropriate services regardless of their location. Because 800 numbers change less frequently, it is more practical to remember (i.e., cache) this virtual number and not have to check the Yellow Pages as frequently.

## 4. Middleware Offers Maximum Flexibility for Maximum Benefit

To understand why a service bus is critical to a well-designed SOA environment, one must first understand the basic elements of such an environment, as shown in Figure 2.

Figure 2: Elements of an SOA environment



**1. Repository.** Metadata repositories can store, query and retrieve virtually any type of data elements or objects, including service definitions, policies, schemas, style sheets, namespaces, sample code and billing information. They can provide other components of the SOA environment with information such as security parameters, message schema information, enabling technology, translation information, rules and logic for message processing, design and architecture information, and object information (for example, properties, methods or inheritance data).

Repositories are designed to be aware of various roles such as registration authority, submitting organization and responsible organizations. By supporting

classification schemes, repositories can manage objects and provide external subscriptions, while flexible associations help facilitate impact analysis.

With a repository as a common reference point for all connected processes and databases, integrating data and methods is as straightforward as finding their equivalents and joining them together. Because the repository knows the schema of both the source and target systems, it also contains information for proper translation of messages flowing from source to target. Sometimes this translation can be automatically generated via semantic mapping without user involvement. Repositories also can help simplify IT administration by tracking all assets and dependencies, performing impact analysis and enabling collaboration and “hot” deployment without requiring system downtime or interruption.

- 2. Service producer.** The service producer creates services—code and configurations designed to execute specific business processes, such as credit authorization—and publishes directory information about those services to a registry. Reuse in an SOA means that multiple consumers share the same instance. Because applications can break when a shared service is shut down or moved, it is important to protect consumers from changes in the producer’s implementation or location.
- 3. WSDL.** Service producers use WSDL to publish service interface descriptions to a registry. Although WSDL documents contain information about the externally visible function of a particular service, they do not always contain all the additional information a developer may need when deciding whether to use the service—for example, WSDL documents do not contain data about pricing or when the service is available. New standards such as WS-Policy are emerging, which will help describe much more of the service contract than its simple WSDL interface—but even with WS-Policy, each implementer may describe attributes differently.
- 4. Registry.** Metadata registries are designed to manage public metadata about a subset of the data elements stored in the various other kinds of enterprise repositories. Most registries are used by developers at design time and are not consulted dynamically at runtime. The registry is designed to work with SOAP messaging and grant access to WSDL documents and other metadata that define interaction with the Web services listed in the directory.

In an SOA environment, registries are based on the UDDI standard. UDDI registries are open, XML-based and platform-independent to allow businesses to list themselves on the Internet and define parameters for interacting with other businesses, or for one group in a company to list its services for use by other groups.

- 5. Service consumer.** Once a service is published to the registry, service consumers can “discover” it either through browsing or subscription. In a point-to-point configuration, consumers call services by making a runtime request to the service producer using SOAP. However, with an ESB, consumers call the ESB, and the ESB routes the request to the best available service producer.
- 6. SOAP request.** After performing a service lookup, either dynamically through the UDDI registry or through a middleware broker, the service consumer uses SOAP to call the service. These requests are commonly made using the HTTP protocol but may use other messaging protocols as well.

An ESB is critical to maintaining high performance, scalability and reliability in an SOA environment. By operating as a moderator between the service consumer and service producer, middleware acts as an information bus that connects each element of the infrastructure and transfers requests among them. This approach helps preserve reliability by creating service interfaces to legacy systems, allowing them to participate in service registries without hardcoding, while also helping to enhance performance by preventing service lookups each time an application calls a service. By preventing direct connection between the consumer and the producer, middleware also can enhance security by acting as a demilitarized zone.

## 5. Point-to-Point Configurations Hamper Performance, Scalability and Reliability

Enterprise applications often operate within extremely complex and inflexible IT environments consisting of monolithic application silos, point-to-point connections and hardcoded interfaces. It can be difficult to modify such environments to meet changing business needs. For this reason, service consumers in SOA systems without an information bus often hardcode their service endpoints as a workaround—a practice that precludes the benefits of dynamic service selection. This approach may boost performance in some situations, but it lacks the flexibility

to allow services to change locations without interrupting processes. If services are migrated to new servers, for example, hardcoded processes will fail at runtime when the service location cannot be found.

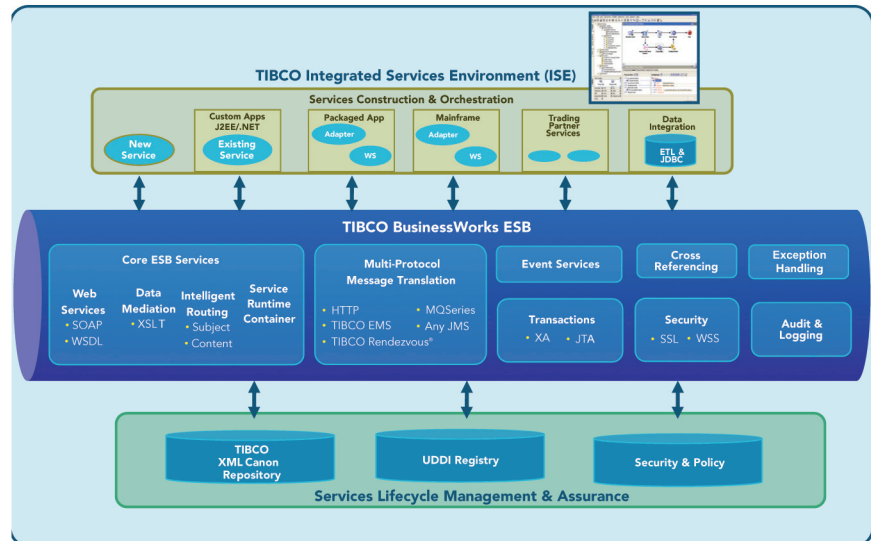
Some companies try to circumvent these problems by using dynamic service lookup—a tactic that can help reduce code maintenance but often causes performance and scalability to suffer. Dynamic service lookup is not realistic for most enterprises for several reasons. Security concerns loom large—companies often do not wish to simply trust a service without first reviewing its code. Performance and scalability are also problematic; if the service broker selects a service that is busy, it may make repeated calls and bog down the entire network.

## 6. TIBCO Software Provides Choice

TIBCO's approach to SOA nonintrusively integrates heterogeneous environments of multiple technologies (see Figure 3.) With a TIBCO-based SOA, enterprise IT organizations are free to choose their own hardware, software and messaging components. In addition, TIBCO-based SOAs give IT departments the option of using nearly any vendor's software for supporting functions such as transformation and administration, either within individual locations or across the organization.

TIBCO pioneered bus-based software architectures for messaging with the Information Bus™ architecture in the 1980s. TIBCO has been evolving toward a standards-based SOA over the years in the form of Java Message Service–based TIBCO Enterprise Message Service™ software and XML-based TIBCO BusinessWorks™ software. Beyond its core set of integration software, TIBCO offers a variety of metadata management tools designed to help IT departments implement SOA using the flexible middleware approach described previously.

Figure 3: TIBCO SOA architecture



**TIBCO BusinessWorks.** As TIBCO's ESB, BusinessWorks is designed to support several communication protocols, development platforms, operating environments and packaged applications. It offers all the functions and standards that are required in an ESB, plus additional features such as orchestration tools, adapters, workflow engines and transaction support. BusinessWorks provides maximum flexibility for creating a scalable, reliable, cost-effective SOA environment. The product is designed to help systems architects make their IT infrastructures more agile and adaptable in order to support constantly changing business requirements. By providing advanced integration technology in a rapidly deployable package, BusinessWorks can help IT administrators manage the entire lifecycle of integration projects, facilitate real-time information exchange and enable automation of business processes.

BusinessWorks is designed to act as both a service producer and a service consumer. By providing a platform for standards-based integration and transformation of legacy systems, this product enables IT departments to define and execute the orchestration of services as part of composite applications and processes. A graphical user environment enables model-driven creation, assembly, deployment and management of services.

BusinessWorks includes a Web-based graphical interface for creating and defining integration scenarios, assembling and deploying services, monitoring applications and tracking system resources. Providing an easy-to-use console and allowing

management by exception help enable rapid resolution of key business problems and increase efficiency.

BusinessWorks is fully integrated with TIBCO XML Canon™ software, an XML-aware repository. XML Canon facilitates the central storage and management of enterprise-wide metadata and other XML assets. BusinessWorks also embeds a runtime repository that enables users to interact with UDDI registries to publish and bind—further enhancing usability and accelerating deployment times.

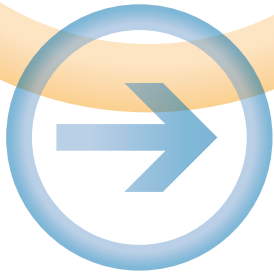
**TIBCO XML Canon.** Designed to store, manage and distribute XML assets, XML Canon is a design-time repository that allows organizations to gain control over their ever-increasing number of XML infrastructure assets and understand the structure and relationship of XML infrastructure objects—helping to unlock metadata for effective collaboration, comprehension and reuse. These XML assets can be reused in both SOA- and non-SOA-based projects in conjunction with both TIBCO and non-TIBCO products.

XML Canon uses a canonical data model to map each piece of metadata against a central repository. With a capability called smart impact analysis, XML Canon automatically flags how changes in one place will affect metadata used elsewhere, thereby helping users detect the impact of changes on the system without coding and debugging.

XML Canon enables companies to reduce development costs by managing XML assets at both the document and component levels from a centralized repository. An easy-to-use Web-based interface creates a virtual workplace and allows authorized developers around the world to collaboratively model, share and reuse common business objects and services throughout the enterprise—greatly reducing time to deployment.

The ability to integrate with the BusinessWorks platform allows XML Canon to store and manage integration projects, cross-referencing maps and other related data. XML Canon also automatically captures the relationship of BusinessWorks components and related metadata to provide a centralized view and access to critical enterprise resources.

Sophisticated revision control capabilities allow various development groups to work with the same development environment without creating redundant objects or services. Built on a strong security model, XML Canon grants access based on both the location of an asset and the status of the asset in its project lifecycle.



## 7. About TIBCO

**TIBCO Software Inc.** (NASDAQ: TIBX) is a provider of infrastructure software for companies to use on-premise or as part of cloud computing environments. Whether it's optimizing claims, processing trades, cross-selling products based on real-time customer behavior, or averting a crisis before it happens, TIBCO provides companies the two-second advantage™ – the ability to capture the right information at the right time and act on it preemptively for a competitive advantage. More than 4,000 customers worldwide rely on TIBCO to manage information, decisions, processes and applications in real time. Learn more at [www.tibco.com](http://www.tibco.com)



Global Headquarters  
3303 Hillview Avenue  
Palo Alto, CA 94304

**Tel:** +1 650-846-1000  
+1 800-420-8450  
**Fax:** +1 650-846-1005

**[www.tibco.com](http://www.tibco.com)**