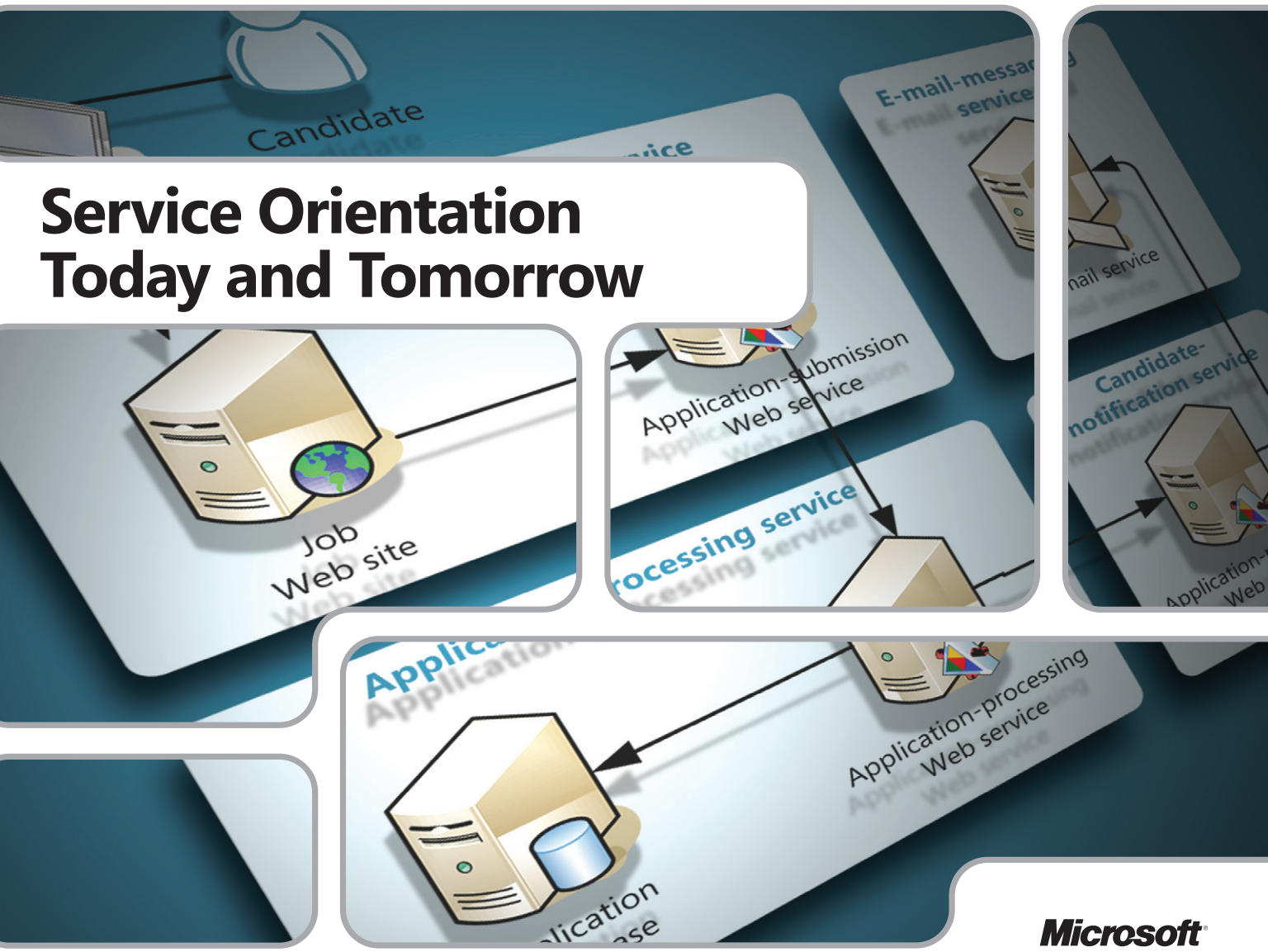
The Architecture Journal

Learn the discipline, pursue the art, and contribute ideas at www.architecturejournal.net

input for better outcomes

21



Service Orientation Today and Tomorrow

Microsoft

Design Considerations for Software plus Services and Cloud Computing

Model Driven SOA with "OSLO"

An Enterprise Architecture Strategy for SOA

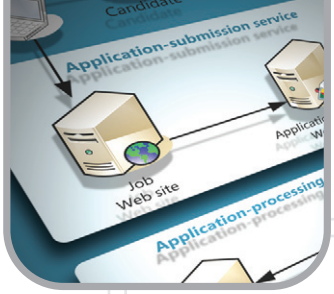
Enabling Business Capabilities with SOA

Service Registry: A Key Piece for Enhancing Reuse in SOA

How the Cloud Stretches the SOA Scope

Event-Driven Architecture: SOA Through the Looking Glass

Is SOA Being Pushed Beyond Its Limits?



Contents

Dear Architect,

When the first *Architecture Journal* was published in January 2004 (back then, called *Microsoft Architects Journal*), it contained an article, "Understanding Service-Oriented Architecture," in which two CBDI analysts explored a novel concept of application integration. At that time, most organizations ignored SOA and its implied concepts. Yet, in those companies that had some degree of SOA awareness, there were few or no ongoing projects that went in that direction.

A few years later, it was rather common to hear of a given organization thinking to realign its whole IT strategy with an SOA model. While they were clear on the goal, the business justification, and the benefits to be had, the initial attempts were still not clear on the right path to take or (even more CIO-enervating) the accuracy of initial cost estimations.

More than five years after that introductory article, thousands of SOA projects have been kicked off, and several of them are still being completed. These days, we have a better understanding about its ROI, low-hanging fruits, hidden costs, and other realities; and the basic, incipient SOA discussion is today specialized in a variety of threads—from service-related ones, such as versioning and discoverability, to broader, portfolio-level threads, such as business-process management.

As if everything about SOA hasn't already been said, other IT trends insinuate a sudden irruption that could twist the original course of SOA by using alternative techniques in noncritical scenarios. These include the modest, lightweight approach that REST offers, as opposed to the rich but heavier OASIS WS-* standards—the latter having matured and evolved with SOA as a purpose.

This issue of *The Architecture Journal* covers several of these debates. The main difference with the article that we published in the early days is that, this time, thoughts have emerged as a consequence of a *practice*; in 2004, thoughts had emerged as a consequence of a *vision*. Both perspectives are necessary: the vision, to understand the goal and what we intend to achieve, and the practice, to help us understand the real dimension of constraints and how to mitigate (if not avoid) them successfully.

These eight articles discuss aspects such as business alignment, service modeling, governance, federation, infrastructure, reusability, convergence, and coexistence with cloud computing and event-driven complementary approaches (among others), and they're accompanied by guest columns that analyze other tangential aspects. I'd like to stop here to thank Eliaz Tobias, IT architect and SOA expert from Microsoft Israel, for his collaboration as guest editor-in-chief.

As we promised two issues ago, we were in the process of leveraging more digital formats. In that sense, this set of articles and guest columns is complemented by a series of short videos on yet other aspects of this SOA topic.

Since our first issue, SOA has walked a long trail, and we still expect a long journey ahead. However, this checkpoint along our way is a great opportunity to share and reaffirm certain concepts that will be useful for the remainder of our journey. I hope that you enjoy these articles, dear reader. As usual, you may send us your comments at archjrnl@microsoft.com.

Diego Dagum
Editor-in-Chief

Design Considerations for Software plus Services and Cloud Computing

by Fred Chong, Jason Hogg, Ulrich Homann, Alejandro Miguel, Brant Zwiefel, Danny Garber, Joshy Joseph, Scott Zimmerman, and Stephen Kaufman

Summary

The purpose of this article is to share our thoughts about the design patterns for a new generation of applications that are referred to as *Software plus Services*, *cloud computing*, or *hybrid computing*. The article provides a view into S+S architectural considerations and patterns as they affect common architectural domains such as enterprise, software, and infrastructure architecture.

Introduction

Many enterprises have IT infrastructures that grew organically to meet immediate requirements, instead of following a systematic master plan. Organically grown enterprise systems have a tendency to develop into large, monolithic structures that consist of many subsystems that are either tightly coupled or completely segregated (sometimes referred to as a "siloed" system). Typically, these systems have arcane and inconsistent interfaces. Their complexity and inefficiency slows down business innovation and can force IT managers to focus on operational and firefighting processes instead of on how information technology can support the core business. Furthermore, some enterprise IT systems have partially duplicated functions that lead to fragmented and inconsistent views of business information, which affects the ability of an enterprise to make sound financial decisions.

Software plus Services (S+S) is an extension of Software as a Service (SaaS) that offers organizations more options for outsourcing development, management, deployment, and operational aspects of the technologies that run their businesses. S+S works in conjunction with principles of service-oriented architecture (SOA). S+S helps an SOA-enabled enterprise increase its technology choices by providing multiple modes of sourcing, financing, and deploying application software and services. To make informed decisions and take full advantage of the potential benefits of adopting an S+S model, IT architects and decision makers should weigh the business drivers and technical requirements against the economic, regulatory, political, and financial forces that are at work from both inside and outside the company.

This article is based on practical experience that was gained by the Microsoft Worldwide Services consulting organization during the design and delivery of S+S and Cloud-based applications. It provides a view into S+S architectural considerations and patterns as they affect common architectural domains, such as enterprise, software, and infrastructure architecture.

SOA, S+S, and Cloud Computing

During the mid-2000s, SOA practices were introduced to help bring sanity to enterprises that were imbued with complex IT infrastructures. Since then, SOA has gone from being a hot industry buzzword to recent pronouncements that SOA is dead. Regardless, SOA instigated key paradigm shifts that remain relevant today.

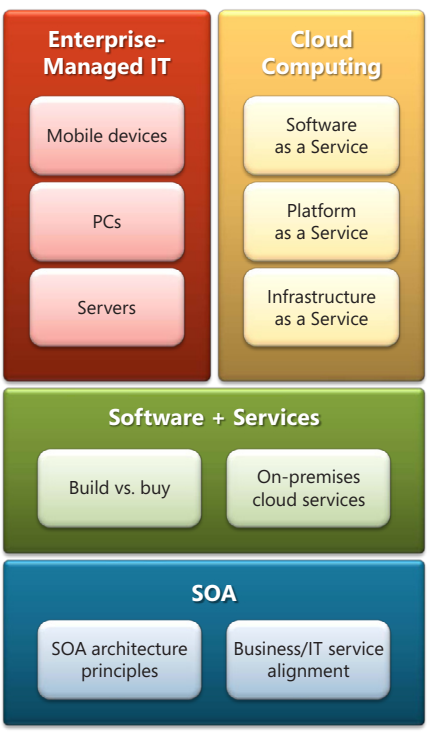
At its technical core, the key impact of SOA is the set of SOA principles, patterns, and analysis processes that enable an enterprise to inventory and refactor its IT portfolio into modular and essential service capabilities for supporting day-to-day business operations. The key objectives of SOA are to align enterprise IT capabilities with business goals, and to enable enterprise IT to react with greater agility as business needs demand. Some key SOA principles that promote agile IT solutions include loose coupling, separation of concerns, standards-based technologies, and coarse-grain service design.

While SOA helps the enterprise identify key service capabilities and architect its business and IT alignment for agility, S+S provides the computing model for organizations to optimize their IT investments through cloud computing and solutions that are deployed in-house. S+S does not invalidate the need for SOA; instead, it empowers an SOA-enabled enterprise to optimize its technology choices by making available multiple modes of sourcing, financing, and deploying application software and services.

The SOA, S+S, and cloud-computing stack relationship is shown in Figure 1.

Because there is not one universally correct IT portfolio for every organization, what is best for an organization depends

Figure 1: Optimizing IT with SOA, S+S, and cloud-computing stack



on its current set of business objectives and requirements. For this reason, the S+S computing model helps an enterprise optimize its IT portfolio by making specific technology choices that are based on decision filters such as cost, relevancy to the core mission, user experience and value for innovation, and business differentiation. S+S offers greater choices for the design of effective hybrid distributed architectures that combine the best features of on-premises software (for example, low latency and rich functionality) with the best features of cloud computing (for example, elastic scalability and outsourcing).

Cloud computing refers to a collection of service offerings. Currently, cloud computing includes vendor solutions for:

- **Infrastructure as a Service (IaaS).** IaaS usually refers to a computing environment in which dynamically scalable and virtualized computation and storage resources are offered as a service. This service abstracts the number of service consumers from the need to invest in low-level hardware, such as servers and storage devices.
- **Platform as a service (PaaS).** PaaS provides operating system and application platform–level abstractions to service consumers. PaaS provides system resource–management functions to schedule processing time, allocate memory space, and ensure system and application integrity within a multitenant environment. PaaS application-development tools enable service consumers to build cloud applications that run on the hosted platform.
- **Software as a service (SaaS).** SaaS refers to business and consumer applications that are hosted by third-party service providers. Service consumers might use Web browsers or installed desktop applications to interact with the hosted applications. In some cases, SaaS providers also offer headless (that is, without a UI) Web services so that enterprises can integrate data and business processes with SaaS applications.

Cloud-computing solutions compliment enterprise-managed infrastructures and offer various benefits to businesses, including the following:

- The ability to allocate resources dynamically, such as additional computing and storage capacity, enables an enterprise to adjust IT expenditures flexibly according to business demands.
- Transaction and subscription–based cloud platforms allow enterprises to develop innovative application solutions quickly for testing new business and operation models, without huge IT investments.
- Outsourced solutions reduce ongoing IT costs and the responsibilities of managing and operating nondifferentiating IT assets (which are opportunity costs to the company).

Design Considerations

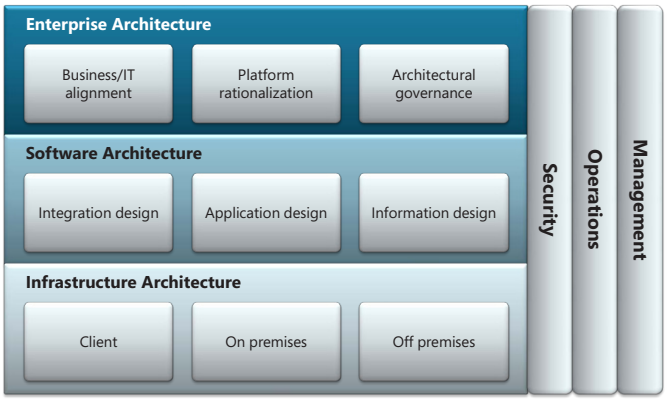
This section provides a summary of the business and technical challenges that a business should consider during the design or adoption of an S+S–based solution. Figure 2 illustrates the frame that is used to organize this document. The frame is organized around specific architectural perspectives and identifies the crosscutting concerns, to provide an end-to-end perspective on the types of scenarios, design considerations, and patterns to consider as part of an S+S strategy.

This information provides a basis for evaluating the end-to-end implications of adopting S+S strategies.

Enterprise Architecture

One of the most demanding aspects of the enterprise-architect role is to balance the constantly changing business needs with the ability of the IT organization to meet those needs consistently. S+S introduces new technology-deployment patterns that reduce operational expense by consolidating—and, in some cases outsourcing—IT platforms, applications, or application (business) services. In addition, S+S can enable organizations to integrate systems across the organization with less friction. Organizations can provide information services to existing business relationships, often by combining existing channels.

Figure 2: Architectural-perspectives framework



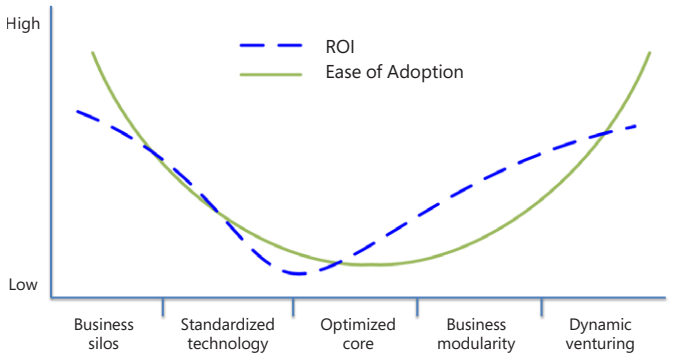
At the highest level, enterprise architects must establish a means of determining the core competencies of the organization and then establish a process for determining which applications support those core competencies, as well as which should perhaps remain in-house and which should not.

The following is a model that is used by several large organizations:

- **Proprietary and mission-critical systems**—Systems that are proprietary or mission-critical in nature or that provide competitive advantages are often considered too important to risk outsourcing to an off-premises service provider. As a result, these systems are usually designed, developed, operated, and managed by the existing IT department of an organization.
- **Nonproprietary and mission-critical systems**—Systems that are nonproprietary yet still mission-critical might be developed by another company, but might still be designed, operated, and managed by the existing IT department an organization.
- **Nonproprietary systems**—Systems that are nonproprietary and deliver standardized functionality and interfaces are often good candidates for outsourcing to a cloud-service provider if appropriate service-level agreements (SLAs) can be established with the service providers. E-mail, calendaring, and content-management tools are examples of such systems.

This model provides a starting point to evaluate applications and systems, but organizations should take into account their individual organizational differences. For example, if an organization is unable to manage its core systems effectively because of cost or missing expertise, it might consider outsourcing them. Likewise, putting some mission-critical systems in the Cloud might offer additional capabilities at little cost that can offset the drawbacks that are introduced. An example might be allowing access to the system by trusted partners or company branches, without having to build an in-house dedicated infrastructure.

Figure 3: S+S impact, depending on IT maturity



However, simply identifying opportunities for moving applications off-premises is insufficient. To leverage S+S opportunities, decision makers must have a clear understanding of the IT maturity of the organization. This understanding allows them to determine what changes in IT infrastructure and processes should be made to optimize the return on investment (ROI) or cost savings that can be gained through S+S adoption.

Figure 3 illustrates the ease of adoption for S+S at varying levels of IT maturity (maturity model based on "Enterprise Architecture as Strategy"[1]) and demonstrates that without determining the organizational maturity, the envisioned ROI might be incorrect.

Software Architecture, Integration Design

Few enterprise applications exist in isolation. Most are connected to other applications and form complex systems that are interconnected through a variety of techniques such as data integration, functional integration, and presentation integration.

In most cases, organizations use a variety of integration techniques, which results in tightly coupled systems that are difficult to separate and replace with off-premises capabilities. Typically, in such cases, the organization either establishes course-grained facades around subsets of functionality within its subsystems or adopts integration technologies that provide a bridge between legacy applications and services that could be hosted locally or off-premises.

When it integrates at the data layer and allows off-premises applications to use the same data as on-premises applications, an organization must consider a variety of factors, such as where the master data should reside. If the data is read-only or reference data, it might be possible to use push-or-pull replication techniques to keep the data synchronized. For business or transactional data, the organization must consider other techniques.

Organizations that use functional SOA-based business services can consider migrating these services to the Cloud, which is discussed in greater detail in the next section. However, in some cases, business applications cannot easily be partitioned into service contract–driven clients and service-provider components. This might be the case when the system involves complex legacy processes and human-driven workflows. In these cases, it might be possible to move the workflow into the Cloud and support a hybrid mode of operation in which the workflow can span both online and offline scenarios.

Traditional atomic approaches to managing transactions might no longer be possible, which would require the organization to examine alternative models that can ensure data consistency. The information-design section of this article describes such processes in greater detail.

Applications that have been developed more recently might use a service-oriented approach to functional integration, which can simplify the adoption of S+S. Applications that use a common service directory might be able to update the location and binding requirements of destination services within the service directory, and the clients might be able to reconfigure themselves dynamically. This can work where a service with the same contract has been relocated off-premises. However, in many cases, the client must interact with a service that has a different contract. In this case, the problem could be mitigated by using service-virtualization techniques[2] by which an intermediary intercepts and transforms requests to meet the needs of the new destination service.

As organizations move applications into the Cloud and become more dependent on services that are provided by multiple service providers, existing centralized message-bus technologies might also be insufficient and, thus, require an organization to consider Internet service bus[3] technologies.

Software Architecture, Application Design

Over the last decade, we have seen many organizations move away from centralized, mainframe-based applications to distributed computing models that are based predominantly on service-oriented and Internet-oriented architectural styles. Applications that are designed according to the principles of service orientation provide a solid foundation for the adoption or integration of S+S applications.

However, we should not assume that this alone is sufficient. Organizations that develop client applications must design these applications to be more resilient when a remote service fails, because remote services are usually outside the control of the consuming organizations. Techniques such as caching of reference data and store-and-forward mechanisms can allow client applications to survive service-provider failures. Additionally, traditional atomic transactions might not be appropriate when interacting with remote services—requiring developers to consider alternative mechanisms, such as compensating transactions.

As a result of services being moved outside of organizational boundaries, the time to access a remote service might also increase. Application developers might need to consider alternative messaging strategies, including asynchronous-messaging techniques. Service providers will almost certainly want to consider using asynchronous-messaging strategies to increase the scalability of their systems.

Alternatively, enabling a client application to interact with alternate service providers depending on their availability and response times might require a client application to resolve the location of such services dynamically and even modify the protocols that are used for interaction. For large organizations that have large numbers of client applications or services that depend on external services, the configuration information must be centralized to ensure consistent management.

Change management requires more attention for S+S applications, because the applications often support multiple tenants. This is further complicated if the application is required to run at or very close to 100 percent availability, which provides little room for upgrades. Rolling updates or application upgrades that use update domains[4] require careful application design, in addition to demanding that the service providers include support for high-availability deployment patterns.

Changes in the design or implementation of a service might inadvertently affect consumers of the services, and client applications might need to be updated to remain functional. Additionally, the

client might need to be modified in S+S scenarios in which services are provided by cloud-service providers; therefore, explicit versioning strategies are critical. In some cases, Web-service intermediaries might provide some ability to safeguard against such changes—allowing message transformations to occur seamlessly to clients. Service providers must have a good understanding of client-usage scenarios to ensure that changes in service contracts or behaviors do not result in unexpected changes in client behavior.

Testing of S+S applications also requires careful attention. Client applications must have the ability to simulate different environments—including development, user acceptance, and performance test environments—with the assurance that these environments are configured identically to the production environments.

Established principles, such as separation of concerns, can make it simpler to allow the security model of an application to change—allowing for new authentication and authorization mechanisms with minimal impact on the application.

Architects have traditionally focused on the design of the application and underlying data stores with the assumption that the organization is the only consumer of the application. During the design S+S applications, they can no longer make this assumption, and they must consider multitenancy and different approaches for scaling out database designs.

Software Architecture, Information Design

Information design is associated with the structure, management, storage, and distribution of data that is used by applications and services. Traditionally, enterprise applications have focused on data consistency, transactional reliability, and increased throughput. They have usually relied on relational data models and relational database-management systems that use the atomicity, integrity, consistency, and durability (ACID) principles as the measure of a reliable database design. S+S forces organizations to think about their information-design process very differently.

SOA adoption has lead to the notion of data as a service, where ubiquitous access to data can be offered independently of the platform that hosts the source data. This capability requires operational data stores that can verify and ensure the cleanliness and integrity of the data, while also considering the privacy and security implications for exposing the data.

Designing a service that will run in the Cloud requires a service provider to consider requirements that are related to multitenant applications. Multitenant applications require alternative schema

Design-Time Selection of Web Services for Composite Applications

by Shrikant Mulik and Manish Godse

The adoption of service-oriented architecture (SOA) in an organization helps the development of new composite applications by reusing services, instead of having to build the applications from scratch. The services could be available internally in the organization or could be offered by external agencies, such as Software as a Service (SaaS).

We propose a nine-step approach toward design-time selection of Web services:

1. Identify the candidate Web services.
2. Establish selection parameters.
3. Classify the parameters as eliminating parameters and evaluation parameters.
4. Collect data about all available services on the selected parameters.
5. Disqualify candidate services against eliminating parameters.
6. Assign weights to evaluation parameters.
7. Score each qualified candidate service on a scale of one to five.
8. Compute the composite score of each service as a weighted average.
9. Select the Web service that has the highest composite score.

Let us consider a case in which an application architect is developing a composite application for the quality-management system in the U.S.-based operations of a global enterprise. The architect has to select one Web service that provides audit-management functionality. As a first step, the architect has identified five candidate Web services. The architect has then established the following 19 parameters, which are grouped into the following five categories:

1. **Functional contract**—Names of operations, and details of input and/or output messages for each of the operations
2. **Quality of service**—Response time, throughput, reliability, availability, accessibility, and interoperability
3. **Security of service**—A security protocol (for example, WS-Security), use of a digital signature (yes/no), use of encryption (yes/no), and the type of security token that is used (for example, X.509 and Kerberos)
4. **Technical details**—A network protocol (for example, HTTP and JMS), a messaging style (RPC or document), an encoding style (SOAP-encoded or literal), and the nature of composition (atomic or composed)
5. **Commercials**—One-time cost, annual ongoing cost, and service-level-agreement (SLA) commitments

Out of these 19 parameters, the architect classified all parameters from functional contract and security of services categories as eliminating parameters. She noted that for eliminating parameters, the requirements are definitive and rigid and she can simply disqualify those candidate services that don't fulfill these requirements. She then went ahead and collected data for all the parameters for all candidate services. She needed to disqualify two candidate services, one for not using required WS-Security protocol and other one for not using digital certificate (as required). All other three services survived the qualification test against eliminating parameters.

Next the application architect assigned weights to evaluation parameters. She used Analytic Hierarchy Process (AHP) to prioritize the parameters from the stakeholders' point of view. Following up the last three steps led to one of the web services receiving the highest composite score. That web service was then selected during the development of Global Quality Management System.

Shrikant Mulik (Shrikant.Mulik@lntinfotech.com) is Lead Consultant at L&T Infotech, Mumbai.

Manish Godse (manishgodse@iitb.ac.in) is a Research Scholar at the Indian Institute of Technology (IIT), Bombay.

designs that must be flexible, secure, and versioned. In some areas, there has also been a movement toward generalized nonrelational data models that provide greater flexibility to tenant-specific schema changes but leave management of data redundancy and possible inconsistencies to the application. Increasingly, systems are processing semistructured or unstructured data (such as documents and media) that are not well-suited to structured relational data models and, therefore, require generalized data models such as name-value stores and entity stores instead. Helpful are supporting queues, blobs for large amounts of unstructured data, and tables with limited relational semantics.

Services and underlying data structures must be designed to support much greater volumes of transactions and/or they must manage much greater volumes of data than in the past. This makes changes to schema designs and data-partitioning strategies necessary. Partitioning strategies must support scaling out of the underlying databases, usually by functional segmentation or horizontal partitioning. Such strategies, however, might affect the ability to obtain optimal performance. This explains why some high-performance systems are moving away from ACID reliability[5] and toward Basically Available, Soft State, Eventually Consistent (BASE) consistency,[6] as well as toward decoupling logical partitioning from physical partitioning schemes.

Infrastructure Architecture

Typically, computing infrastructure represents a significant proportion of the enterprise IT investment. Before the era of cloud computing, an enterprise did not have many alternatives to spending significant amounts of capital to acquire desktops, server hardware, storage devices, and networking equipment to meet its infrastructure needs. Larger enterprises might have also invested in building and operating private data centers to host their hardware and service personnel.

Now, with IaaS, private cloud services, and various forms of virtualization technology, enterprises have alternatives and are able to reassess their IT-infrastructure investment strategy.

With IaaS, enterprises can choose to pay for computing resources by using transaction or subscription payment schemes. An infrastructure that scales dynamically enables an enterprise to adjust its infrastructure expenditure quickly, according to business needs. When the infrastructure budget evolves into an operating expense that can be increased or decreased as the demand for compute cycles and storage fluctuates, enterprises gain new financial flexibility. Such infrastructure services are useful for e-commerce sites that have spikes in computation needs during peak seasons or time of day, followed by dips in resource usage during other periods. IaaS can simplify the capacity-planning tasks for IT architects, because computing resources can now be added or retired without over- or under-investing in infrastructure hardware.

Additionally, IaaS enables enterprises to be more agile in launching and testing new online-business services. Traditionally, the business would need to weigh the business case of upfront investment in hardware for online-service experiments. Even more challenging were the struggles with internal IT to reconfigure corporate networks and firewalls to deploy the trial services. Internal policy–compliance issues have been a frequent cause of delay for new online-business ventures. Now, IaaS can help expedite the process and reduce infrastructure-related complications.

User desktops can be delivered as a service, and users can access them from any location that has network connectivity to the virtualization service. Server-virtualization technology helps reduce the server-hardware footprint in the enterprise. By using IaaS, an enterprise can derive immediate infrastructure cost savings by replicating virtual server instances to run on the cloud infrastructure as business needs require.

While cloud infrastructure–related services can bring many benefits that were not previously available to enterprises, the advantages do not come for free. IT architects must continue to weigh design considerations that concern availability, scalability, security, reliability, and manageability while they plan and implement a hybrid S+S infrastructure.

Infrastructure service disruptions will ultimately affect the availability of application services. Because a number of higher-level application services might be running on an outsourced cloud infrastructure, an outage at the service-provider infrastructure could affect more than one business function—resulting in the loss of business productivity or revenue in multiple areas. Therefore, enterprises should know whether their infrastructure-service providers can help mitigate such failures. Alternatively, an enterprise might want to use multiple infrastructure-service providers so that it can activate computing resources at the secondary provider in the event of service failure at the primary provider.

When desktop-virtualization technology is delivered through a centralized hosting infrastructure, it is important to consider scalability and performance of the solution. Often, desktop-virtualization services are at peak load during office hours when employees log on and perform office tasks. The load gradually tapers off after-hours. Combining virtualization technology with a dynamically expandable computing infrastructure service can be a good approach when the computational demands of an organization fluctuate.

Infrastructure security has always been part of the defense-in-depth strategy for securing the enterprise. For example, some enterprises rely on IPSec for protecting machine-to-machine communications within the intranet. This mechanism can add another layer of defense to protect against unauthorized information access by non-corporate-owned computing devices. To continue using existing infrastructure-level security mechanisms with cloud-infrastructure services, an enterprise might need to reconfigure its network, public-key, and name-resolution infrastructure.

When the server infrastructure is deployed to run as virtualized instances, IT architects should think about organizing the virtual instances into atomic units, where a service failure is isolated within each unit and does not affect other atomic collections of virtualized services. This infrastructure-design practice enables higher application services to be deployed as atomic units that can be swapped in if the virtualized instances fail to operate in another unit.

When higher-level applications are deployed across a hybrid S+S infrastructure, it can be difficult to debug application failures that occur because of infrastructure malfunction. Traditional network-monitoring and tracing tools that are used within the enterprise might cease to work across the boundaries of corporate and service-provider firewalls. Therefore, an enterprise can request that its cloud-infrastructure providers provide diagnostic tools that can help inspect cloud-infrastructure traffic.

Security

Security has been a key area of enterprise computing focus since the late 1990s, when businesses began using the Internet as a mainstream channel of commerce and customer service. In the current computing era of S+S, the security best practices and the technology developed to serve the business Web not only remain relevant, but are even more important to observe.

Speed Up to the Next Level of Enterprise Architecture

by Mario Fraiß and Erwin Zinser

In the future, the terms *adaptability*, *flexibility*, and *agility* should form the starting point for all considerations that concern the planning of integrated and automated IT infrastructure environments.

Furthermore, enterprise-architecture (EA) design concepts such as SOA and SOI have to be carefully considered as mandatory requirements for building a so-called *Intelligent Enterprise Architecture (IEA)*. The upcoming target is the creation of a mash-up between fully automated services and specifically placed decision points that are controlled by humans.

As a result of the permanently increasing automation level within automated IT infrastructure environments, it is necessary to discuss and scrutinize the advantages and disadvantages. Otherwise, we might end up in a fictional scenario in which computers could take control.

Primarily, we demand the intelligent use of available business knowledge, so that the intellectual capital that is thus obtained might subsequently be used to provide optimum support for the business and its corresponding business processes. Additionally, we have to cope with permanently changing requirements in the field of organizational and infrastructural management.

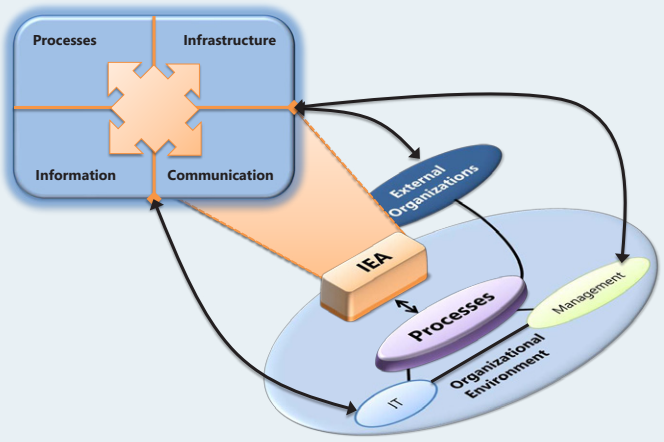
This new and changed focus of modern enterprise architectures has to be followed by realigned developmental techniques and methods and applied by future-oriented enterprise architects (see Figure 1).

We have to build solutions for complex event processing and the associated event and process monitoring. Moreover, these developments can establish the possibility of realizing independent, dynamic instantiation of problem-solving processes. To reach this fantastic vision, we introduce the approach of the *IEA*.

This approach was primarily developed on the basis of an analysis of current EA concepts and the consistent application of service orientation at each layer of the respective framework that was under consideration. In fact, various different views from the world of industry and technology were examined.

As shown in Figure 1, the core components of the IEA are *communication*, *information*, *infrastructure*, and *processes*. These subsystems have to concentrate on the central competencies of

Figure 1: Overview of an intelligent enterprise environment



the whole enterprise to guarantee a fully functional and business-supporting environment.

In the course of a research project, we set up a virtualized fictitious company that we used as a prototypical scenario to check out the validity of our concepts. In conclusion, we are now able to demonstrate that the approach to induce the IEA design paradigm works perfectly. These consolidated findings might be seen as a successful, pioneering step in the area of cutting-edge EA design principles.

Mario Fraiß (mario@fraiss.at, http://www.mariofraiss.com) is the founder and CEO of FRAISS – IT Consulting & Media Design, an Austrian technology-consulting company that specializes in developing innovative business solutions that are based on Microsoft technologies.

Erwin Zinser (erwin.zinser@fh-joanneum.at, http://www.entology.eu) is Professor of Enterprise Architecture Design at the FH JOANNEUM University of Applied Sciences, Department of Information Management, in Graz, Austria.

S+S security covers a broad spectrum of topics, ranging from the provisioning of identities and their entitlements, to enabling enterprise single sign-on between on-premises systems and cloud services, to protecting data in transit and at rest, to hardening application code deployed on cloud platforms against malware and penetration attacks.

User provisioning is a key task in the life-cycle management of user identities. When an enterprise adopts a cloud service, it must consider how its enterprise users are provisioned with the cloud-service providers. In addition, as a user's organizational role changes, the identity management processes should ensure that the user's application permissions are adjusted accordingly at the cloud service. When a user leaves the enterprise, access to the cloud service should also be deactivated. The user provisioning activities for S+S should be automated as much as possible to reduce manual provisioning errors and prevent loss of employee productivity that is due to service-access issues.

Enabling single sign-on (SSO) by using existing corporate identities is a key requirement and priority for many enterprises that adopt cloud services. The reasons are obvious. SSO provides convenience and better application experiences to end users and can reduce security issues that arise from having to manage multiple security credentials. Rationalizing and consolidating multiple identity systems within the enterprise is usually the first step in meeting the SSO challenge. New identity-federation technology can also improve the portability of existing user credentials and permissions and should definitely be a key part of the SSO strategy with cloud-service providers.

Data is critically important in every business. Therefore, an enterprise should set a high bar for ensuring that its business information continues to be secure in the S+S world. The key security issues that concern data are confidentiality and integrity when data is transmitted over the Internet and when information is stored at the cloud-service provider. Security mechanisms such as encryption and signing can help ensure that data is not being viewed or modified by unauthorized personnel.

New security threats, exposures, and mitigation approaches must be factored in to the security strategy for enterprise applications that are developed and hosted on Internet-accessible cloud-computing platforms. The potential security threats range from service disruptions that are Internet hacks to the risk of proprietary business logic in application code and trade-secret content being discovered and stolen. The practice of a secure-by-design security-review process becomes even more crucial during the delivery of applications to run on cloud-computing platforms.

Finally, IT decision makers and implementers should be mindful that any system is only as secure as its weakest link. Therefore, companies should always examine new risk exposures that arise from using cloud providers and take appropriate measures to mitigate those risks. If the weakest link is the outsourced provider, it can invalidate many security measures that the company has put in place.

Management

IT management deals with the end-to-end life-cycle management of software applications and services that are used by the enterprise to accomplish its business objectives. The life-cycle stages include planning, implementing, operating, and supporting an IT portfolio that consists of the hardware, network, infrastructure, software, and services that support day-to-day business operations.

Typically, IT management includes the following activities:

1. Defining policies to guide project implementation and operation procedures
2. Putting processes in place to systematize execution
3. Identifying organizational roles with clearly defined accountabilities
4. Implementing and maintaining the tools that automate IT-management operations

Best practices for the first three activities can be found in existing industry management frameworks such as the Information Technology Infrastructure Library (ITIL)[7] and the Microsoft Operations Framework (MOF),[8] while architecture principles and IT-management solutions are the key ingredients for automating IT operations.

S+S extends the enterprise IT environment beyond its corporate firewall—not only from a deployed technology perspective, but also from the perspectives of IT roles and accountabilities, operational procedures, and policies that govern the use and operation of deployed software and services.

For example, applications that are outsourced to an SaaS provider are now maintained by administrators and operators who are not employees of the enterprise. In the S+S world, traditional IT roles and accountabilities might need to be collapsed into a single service-provider role that is contractually responsible for the duties that are specified in an SLA. Legally enforceable liability clauses should also be clearly defined to mitigate any negative result that might occur because a service provider cannot perform its responsibilities satisfactorily. Similarly, IT-management processes for resolving user issues and technical problems are now handled by the service provider. Establishing clear escalation procedures and integrating effective communication channels into the end user–support process of the enterprise are vital for the minimization of service disruptions.

Although the enterprise no longer controls the implementation details of the outsourced services, the company should be aware of any of the mechanisms and procedures of the service provider that might affect the accountabilities and liabilities between the enterprise organization and its customers. Some service providers will provide documentation that complies with auditing standards such as SAS 70, which can help the enterprise determine if the IT-management practices at the service provider meet their business and industry requirements.

Enterprise organizations should plan to deploy IT-management solutions for monitoring services that run in the Cloud. The operation strategy should outline the performance indicators and management rules that are required to gain visibility into the performance and availability of the external services. Operational monitoring systems should raise management notifications and alerts, so that any service anomaly can be detected early.

Additionally, both outsourced service providers and enterprises that are developing cloud services should implement operation-related service interfaces to automate management tasks such as provisioning user accounts, setting user permissions, changing service run state, and initiating data backups.

In summary, IT management in the S+S world must continue to embrace the end-to-end strategy of planning, delivering, and operating the IT capabilities that are needed to support business operations. Existing IT-management frameworks are still relevant. Enterprises, however, should consider the impact that arises as they integrate external operation processes, personnel, and tools into the existing IT-management practices.

With external service providers taking responsibility for systems, organizations lose much of the direct control that they used to have over the people, processes, and technology. Instead, the organizations must provide effective management through clearly defined and mutually agreed-upon SLAs, policies and procedures, key performance indicators, management rules, and service-control interfaces. Design for operation is the architecture mantra for delivering manageable software and services. Ultimately, the outsourcing of operational details to cloud-service providers should empower existing IT staff to focus on new, high-priority computing projects that deliver greater business value.

Operations

Operations make up a very specific stage of the IT-management life cycle. It involves the day-to-day activities of monitoring software and services, taking corrective actions when problems arise, managing customer helpdesks to resolve user issues, performing routine tasks such as backing up data, and controlling and maintaining consistent service run states to meet the required quality of service. Operational procedures are governed by IT policies, and the outcome is measured by precise systems and applications health metrics such as availability and response times.

For example, the MOF outlines a best-practices model for these activities.

As enterprises adopt an S+S strategy, they must consider the business impact of outsourcing IT operational roles and responsibilities. Business continuity, liability, and employee and customer satisfaction are all key concerns that must be addressed by establishing clear SLAs with reliable cloud-service providers.

The enterprise should continue to play a proactive role in IT operations for its hybrid software-and-services environment. However, instead of focusing on execution details, enterprises should put monitoring systems in place that enable them to detect technical issues at the outsourced services. Enterprises should also establish operational procedures to ensure that problems are resolved by the service providers as quickly as possible.

Both enterprises and cloud-service providers can increase their S+S operational effectiveness by designing for operational best practices.

"Designing for operation" requires architecture and execution discipline over the course of planning, delivering, and operating software and services. Architects should be aware of the transition points in their applications when stability, consistency, reliability, security, and other quality factors are affected, and should include instrumentation features within the applications to notify monitoring tools of impactive events. Architecture concerns and patterns such as application health state, performance counters, management events, logs, and synthetic transactions can help provide operation-ready software and services.

During their evaluation of a cloud service, enterprises should determine if the service providers offer application-performance information and operation service interfaces that can be consumed by standard off-the-shelf IT monitoring solutions. Enterprises should also architect their systems so that the failure of a service is compartmentalized. Only the parts of the solution that are dependent on that service should be affected. This operational strategy helps maximize business continuity.

Conclusion

S+S brings new opportunities for everyone. It provides new options for optimizing business and IT assets, and enables organizations to save cost, increase productivity, innovate, and reach new markets.

There are three main ways to think about extending the current portfolios of on-premises technology with cloud computing: consume the Cloud, use the Cloud, and embrace the Cloud:

- **Consume the Cloud** is fundamentally about an enterprise outsourcing applications and IT services to third-party cloud providers. The key business drivers that push enterprises to consume online services are reducing IT expenses and refocusing valuable bandwidth on enabling core business capabilities. Cloud providers can usually offer commodity services cheaper and better because of their economy of scale. They can pass-on the cost savings and efficiency to enterprise customers. For Microsoft customers, some good examples are the Microsoft Business Productivity Online Suite (consisting of Exchange Online and Office SharePoint Online), CRM Online, and Live Meeting services.
- **Use the Cloud** enables enterprises to tap into cloud platforms and infrastructure services and get an unlimited amount of compute and storage capacity when they need it, without having to make large, upfront capital investments in hardware and infrastructure software. Such a utility computing model gives enterprises more agility in acquiring IT resources to meet dynamic business demands. In addition, by using cloud services, enterprises can avoid affecting the existing corporate infrastructure and speed up the deployment of Web-based applications to support new business initiatives that seek closer communication with customers and partners. For Microsoft customers, some good examples include Windows Azure and SQL Azure.
- **"Embrace the Cloud"** occurs when enterprises deploy technology that enables them to offer cloud services to their customers and partners. Service-oriented enterprises are best positioned to take advantage of this model by transforming their core business assets into information cloud services that differentiate them from their competitors. For Microsoft customers, on-premises technologies such as the BizTalk Server Enterprise Service Bus Toolkit can integrate data feeds and orchestrate workflows that process information exchange via cloud services.

Acknowledgments

The authors would like to thank the following reviewers: Tim O'Brien, Rob Boucher Jr., and Sharon Smith.

Resources

Meier, J.D., Alex Homer, David Hill, Jason Taylor, Prashant Bansode, Lonnie Wall, Rob Boucher Jr, and Akshay Bogawat. Microsoft patterns & practices Application Architecture Guide 2.0: Designing Applications on the .NET Platform. January 15, 2008. (See Chapter 13, "Service Layer Guidelines," and Chapter 18, "Services.")

References

1. Ross, Jeanne W., Peter Weill, and David Robertson. *Enterprise Architecture as Strategy: Creating a Foundation for Business Execution*. Boston, MA: Harvard Business School Press, 2006.
2. Skonnard, Aaron. "Service Virtualization with the Managed Services Engine." MSDN Magazine, May 2009.
3. Ferguson, Donald F., Dennis Pilarinos, and John Shewchuk. "The Internet Service Bus." The Architecture Journal, MSDN, October 2007.
4. Khalidi, Yousef A. "Architecting Services for Windows Azure." Professional Developers Conference 2008 (PDC2008), 2008, Slide 15.
5. Bain, Tony. "Is the Relational Database Doomed?" ReadWriteEnterprise, February 12, 2009.
6. Pritchett, Dan. "BASE: An Acid Alternative." ACM Queue, July 28, 2008.
7. ITIL®. "Information Technology Infrastructure Library." Official ITIL® Website, 2009.
8. Microsoft Corporation. "Microsoft Operations Framework." Microsoft TechNet, 2009.

About the Authors

Fred Chong is working on new and emerging market solutions, integrating mobile phones, software, and cloud services to improve the standard of living and productivity of citizens in developing countries.

Alejandro Miguel, Jason Hogg, and **Joshy Joseph** are architects inside the Solution Engineering team within the Worldwide Services Office of the CTO.

Ulrich Homann is the Chief Architect for WorldWide Enterprise Services.

Brant Zwiefel is a business architect in the Microsoft Services—Service Lines and Marketing team.

Danny Garber is a U.S. Azure Community Lead Architect within the Architecture Team of the CTO.

Scott Zimmerman is an SOA and BizTalk Solutions Architect in the U.S. Mid-Atlantic District who advises customers on S+S.

Stephen Kaufman is a delivery architect who focuses on middle-tier technologies with Microsoft Consulting Services.

Follow up on this topic

- Applications in the cloud with Windows Azure Platform

Model-Driven SOA with "Oslo"

by César de la Torre Llorente

Summary

Service-oriented architecture (SOA) must evolve toward a more agile design, development, and deployment process. Most of all, it must close the gap between IT and business. This article presents a map between modeling theory (model-driven development and model-driven SOA) and possible implementations with the next wave of Microsoft modeling technology, codename "Oslo."

Introduction

The following article presents a map between modeling theory (*model-driven development* and *model-driven SOA*) and possible implementations with the next wave of Microsoft modeling technology, codename "Oslo."

Microsoft has been disclosing much information about "Oslo" while delivering several Community Technology Previews (CTPs), or early betas. However, most of the information that is available on "Oslo" is very technology-focused (internal "M"-language implementation, SDK, and so on). This is why I want to present a higher-level approach. Basically, I want to discuss *Why modeling*, instead of only *How*.

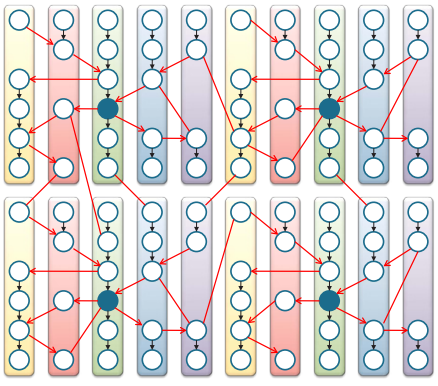
Problem: Increase in SOA Complexity

What is the most important problem in IT? Is it languages, tools, programmers? Well, according to researchers and business users, it is software complexity. And this has been the main problem since computers were born. Application development is really a costly process, and software requirements are only increasing. Integration, availability, reliability, scalability, security, and integrity/compliancy are becoming more complicated issues, even as they become more critical. *Most solutions today require the use of a collection of technologies, not only one. At the same time, the cost to maintain existing software is rising.*

In many aspects, enterprise applications have evolved into something that is too complex to be really *effective* and *agile*.

With regard to *service-oriented architecture* (SOA), when an organization has many connected services, the logical network can become extremely difficult to manage—something that is similar to what is shown in Figure 1, in which each circle would be a service and each box a whole application.

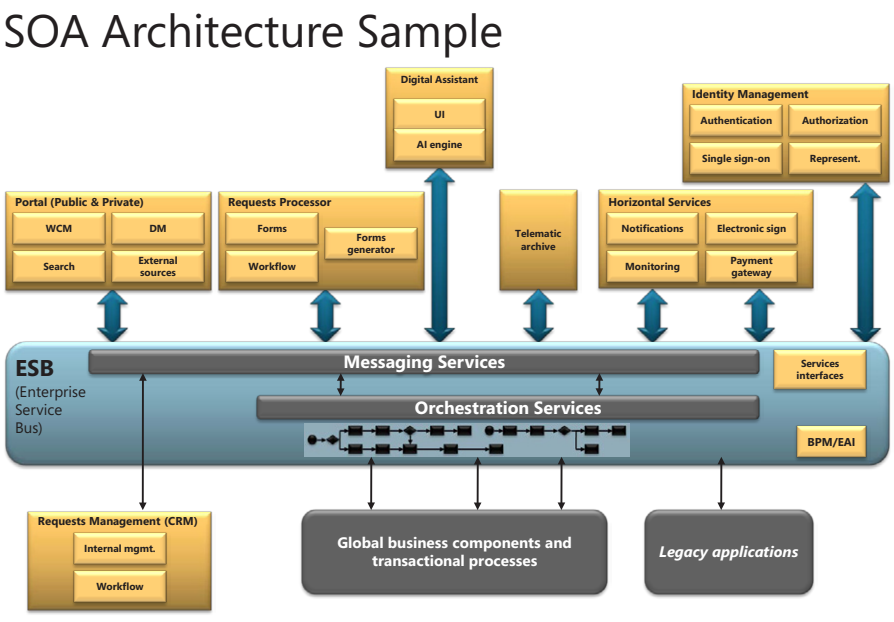
Figure 1: Point-to-point services network



The problem with this services network is that all of these services are directly connected; we have too many point-to-point connections between services. Using point-to-point services connections is fine, when we have only a few services; however, when the SOA complexity of our organization evolves and grows (having too many services), this approach is unmanageable.

Sure, you will say that we can improve the previous model by using an enterprise-service-bus (ESB) approach and service-orchestration platforms, as I show in Figure 2. Even then, however, the complexity

Figure 2: SOA architecture, based on ESB as central point



Language Is the Interface (Cloudly Speaking)

by Nagaraju Pappu

In traditional component development, the underlying platform services run in the execution space of the component; in a service-oriented architecture (SOA), however, the application component is hosted in a service container. The service container provides standards-based interfaces and service-level agreements (SLAs) that relate to performance, reliability, and availability, as shown in Figure 1.

Cloud computing and the emerging collaborative platform-oriented computing take this concept even further. We now require the ability for clients to control programmatically the SLAs that are provided dynamically by the component.

This is achieved easily if the primary interface of the component is a "programming language," instead of a set of APIs. In other words, the exchanges among components are scripts or programs, not requests and responses. Instead of a union or selection of all of the contained objects as its primary interface elements, a component should have as its primary interface a language interpreter that has an appropriate grammar, as shown in Figure 2.

The interface now consists of:

- The actions or methods that are provided by the constituent objects (basic functionality).
- The infrastructure and platform services—such as fault-tolerance services and connection pooling—that are provided by the framework and constitute the measurements and state-manipulation services of the component.
- A grammar that allows combining both services and functionality in a natural way programmatically.

Figure 1: From components to services

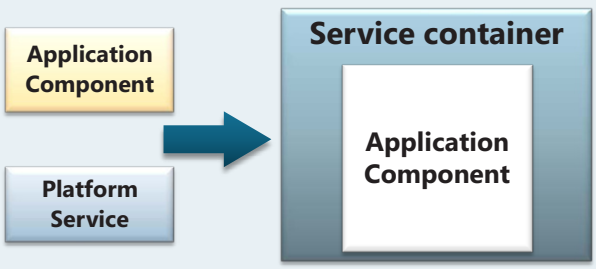
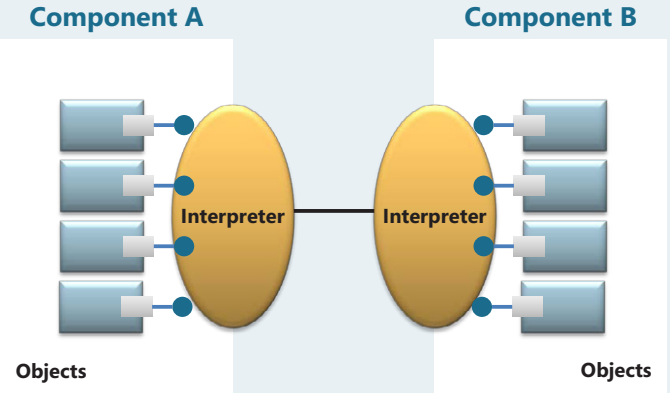


Figure 2: Communicating by using language as primary interface



Although it is highly appealing to build components in this manner, the major architectural challenge is to ensure reliability of the components and their services so as to prevent inherent failures of the component from percolating upwards. One way to ensure the reliability of programmable component architectures is to model the component in such a way that its SLAs are queried and controlled programmatically.

Architectural techniques that ensure this reliability include the design of components that:

- Offer programmatically controlled SLAs via a scripting engine as the primary API.
- Are state-aware and measure and publish their health and capacity to deliver against SLAs enabling graceful failover.
- Have longer design lifetimes, due to design that is centered on variables that change less often.
- Are resilient to environmental failures.
- Manage the effects of environmental dynamism by using design-time controls.

These architectural techniques are described in detail in our blog.

Nagaraju Pappu is the founder of and chief technology consultant at Canopus Consulting.

is very high: Implementation is based on a low level (programming in languages such as the Microsoft .NET languages and Java); therefore, its maintenance and evolution costs are quite high, although not as high as having non-SOA applications.

SOA foundations are right; however, *we must improve the work when we design, build, and maintain.*

Does SOA Really Solve Complexity, and Is It Agile?

On the other hand, SOA has been the "promised land" and a main IT objective for a long time. There are many examples in SOA in which the theory is perfect, but its implementation is not. The reality is that there are too many obstacles and problems in the implementation of SOA. Interoperability among different platforms, as well as real autonomous services being consumed from unknown applications, really are headaches and slow processes.

SOA promised a functionality separation of a higher level than object-oriented programming and, especially, a much better decoupled components/services architecture that would decrease external complexity. Additionally, separation of services implementation from services orchestration results in subsystems being much more interchangeable during orchestration. The SOA theory seems correct.

But, what is the reality? In the experiences of most organizations, SOA in its pure essence has not properly worked out. Don't get me wrong; services orientation has been very beneficial, in most situations—for instance, when using services to connect presentation layers to business layers, or even when connecting different services and applications.

However, when we talk about the spirit of SOA (such as the four tenets), the ultimate goals are the following:

> *In SOA, we can have many autonomous services independently evolving—without knowing who is going to use my services or how my services are going to be consumed—and those services should even be naturally connected.*

I think that this is a different story. This theory has proven to be very difficult; SOA is not as agile or oriented toward business experts as organizations would like.

Is SOA Dead?

A few months ago, a question arose in many architectural forums. It probably was started by Anne Thomas Manes (Burton Group) in a blog post called "SOA Is Dead; Long Live Services."

So, *is* SOA dead? I truly don't think so. SOA foundations are completely necessary, and we have moved forward in aspects such as decoupling and interoperability (when compared with "separate worlds" such as CORBA and COM). So, don't step back; I am convinced that service orientation is very beneficial to the industry.

The question and thoughts from Anne were really very interesting, and she really was shaking the architecture and IT communities. She did not really mean that service orientation is out of order; basically, what she said is the following:

> *"SOA was supposed to reduce costs and increase agility on a massive scale. Except in rare situations, SOA has failed to deliver its promised benefits... .*
>
> *"Although the word 'SOA' is dead, the requirement for service-oriented architecture is stronger than ever.*
>
> *"But perhaps that's the challenge: The acronym got in the way. People forgot what SOA stands for. They were too wrapped up in silly technology debates (e.g., 'What's the best ESB?' or 'WS-* vs. REST'), and they missed the important stuff: architecture and services."*

So, I don't believe that SOA is dead at all, nor do I think that Anne meant that. She was speaking out against the misused SOA word. Note that she even said that "*the requirement for service-oriented architecture is stronger than ever.*" The important points are the architecture and services. Overall, however, I think that we need something more. *SOA must become much more agile.*

Furthermore, at a business level, companies every day are requiring a shorter time to market (TTM). In theory, SOA was going to be the solution to that problem, as it promised flexibility and quick changes. But the reality is a bit sad—probably not because the SOA theory is wrong, but because the SOA implementation is far from being very agile. We still have a long way to go.

As a result of these SOA pain points, business people can often feel very confused.

As Gartner says, SOA pillars are right, but organizations are not measuring the time to achieve a return on investment (ROI). One of the reasons is that the business side really does not understand SOA.

To sum up, we need much more agility and an easier way to model our services and consumer applications. However, we will not achieve this until business experts have the capacity to verify directly and even model their business processes, services orchestration, or even each service. Nowadays, this is completely utopian. But who knows what will start happening in the near future?

Model-Driven SOA: Is That the Solution?

Ultimately, organizations must close the gap between IT and business by improving the communication and collaboration between them. The question is, "Who has to come up?" Probably, both. IT has to come closer to business and be much more accessible and friendly. At the same time, business experts have to reach a higher level to be able to leverage their knowledge more directly. They have to manipulate technology, but in a new way, because they still have to focus on the business; they do not have to learn how to program a service or application by using a low-level language such as C#, VB, or Java.

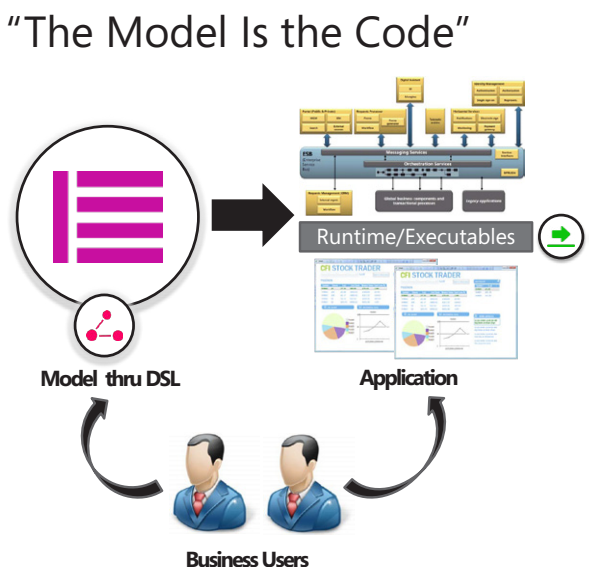
Someday, *model-driven SOA* might solve this problem. I am not talking only about services orchestration (by using platforms such as Microsoft BizTalk Server); I mean something else—a one-step-forward level—in which orchestration has to be managed not by technical people, but by business experts who have only a pinch of technical talent. This is something far from the context of today, in which we need technical people for most application changes. I mean a new point of view—one that is closer to the business side.

Model-driven SOA will have many advantages, although we will have to face many challenges. But the goal is to solve most typical SOA problems, because *model-driven development* (MDD) makes the following claim: "The model is the code." (See Figure 3.)

The advantages and claims of model-driven SOA are the following:

1. **The model is the code. Neither compilation nor translation should be necessary.** This is a great advantage over code generators and finally drives us to maintain/update the code directly.
2. **Solutions that model-driven SOA creates will be compliant with SOA principles,** because we are still relying on SOA. What we are changing is only the way in which we will create and orchestrate services.
3. **Models have to be defined by business languages.** At that very moment, we will have achieved our goal: to close the gap between IT and business, between CIO and CEO.
4. **We will consequently get our desired flexibility and agility** (promised by SOA, but rarely achieved nowadays) when most core business processes are transformed to the new model. When we use model-driven SOA, changes in business processes have to be made in the model; then, the software automatically changes its behavior. This is the required alignment between IT and business. This is the new promise, the new challenge.

Figure 3: MDD: "The model is the code."



One Objective with "Oslo": Is "Model-Driven SOA" Possible?

Microsoft is actually building the foundations to achieve MDD through a base technology, codename "Oslo." Microsoft has been disclosing what "Oslo" is since its very early stages, so as to get customer and partner feedback and create what organizations need.

Of course, there are many problems to solve. The most difficult is to achieve "Oslo" modeling-specific domains—reviewed and even executed by business experts, and at the same time producing interconnected executables between different models.

Basic Concepts of "Oslo"

"Oslo" is the code name for the forthcoming Microsoft modeling platform. Modeling is used across a wide range of domains; it allows more people to participate in application design and allows developers to write applications at a much higher level of abstraction.

So, what is "Oslo"? It consists of three main pillars:

- A language (or set of languages) that helps people create and use textual, domain-specific languages (DSLs) and data models. I am talking about the "M" language and its sublanguages ("M," MGraph, MGrammar, and so on). By using these high-level languages, we can create our own textual DSLs.
- A tool that helps people define and interact with models in a rich and visual manner. The tool is called "Quadrant." By using "Quadrant," we will be able to work with any kind of visual DSL. This visual tool, by the way, is based on the "M" language as its foundation technology.
- A relational repository that makes models available to both tools and platform components. This is called simply the "Oslo" repository.

I am sure that Microsoft will be able to close the loop in anything that is related to those key parts ("M," "Quadrant," and the repository) for modeling metadata. In my opinion, however, *the key part in this architecture is the runtime-integration with "Oslo," such as integration with the next Microsoft application-server capabilities (codename "Dublin"), ASP.NET (Web development), Windows Communication Foundation (WCF), and Workflow Foundation (WF). This is the key for success in model-driven SOA (and MDD in general).*

Will we get generated source code (C#/VB.NET) or generated .NET MSIL assemblies? Will it be deployed through "Dublin"? Will it be so easy that business experts will be able to implement/change services? Those are the key points. And this is really a must, if "Oslo" wants to be successful in MDE/MDD.

In this regard, that is the vision of Bob Muglia (Senior Vice President, Microsoft Server & Tools Business), who promises that "Oslo" will be deeply integrated in the .NET platform (.NET runtimes):

> *"The benefits of modeling have always been clear; but, traditionally, only large enterprises have been able to take advantage of it, and [only] on a limited scale. We are making great strides in extending these benefits to a broader audience by focusing on [certain] areas. First, we are deeply integrating modeling into our core .NET platform. Second, on top of the platform, we build a very rich set of perspectives that help specific persons in the life cycle get involved."*

"Oslo" to Bring Great Benefits to the Arena of Development

In my opinion, it is good to target high goals. That is why we are trying to use MDD and MD-SOA to close the gap with business.

However, even if we cannot reach our business-related goals, "Oslo" will bring many benefits to the development arena. In any case, we architects and developers will get a metadata-centric applications factory. To quote Don Box (Principal Architect in the "Oslo" product group):

> *"We're building 'Oslo' to simplify the process of developing, deploying, and managing software. Our goal is to reduce the gap between the intention of the developer and the actual artifacts that get deployed and executed.*
>
> *"Our goal is to make it possible to build real apps purely out of data. For some apps, we'll succeed. For others, the goal is to make the transition to traditional code as natural as possible."*

MDD and "Oslo"

In the context of "Oslo," MDD indicates a development process that revolves around the building of applications primarily through metadata. In fact, this is the evolution that we had for all development languages and platforms. In every new version of development platforms, we had more and more developmental metadata and less hardcoded/compiled code (consider WF, WPF, XAML, and even HTML). However, with MDD and "Oslo," we are going several steps ahead—meaning that we are moving more of the definition of an application out of the world of code and into the world of data. As data, the application definition can be easily viewed and quickly edited in a variety of forms (even queried)—making all of the design and implementation details much more accessible. This is what the "Oslo" modeling technology is all about.

On the other hand, models (especially in "Oslo") are relevant to the entire application life cycle. The term *model-driven* implies a level of intent and longevity for the data in question—a level of conscious design that bridges the gaps between design, development, deployment, and versioning.

For instance, Douglas Purdy (Product Unit Manager for "Oslo") says the following:

Figure 4: Architecture of "Oslo" and related runtimes

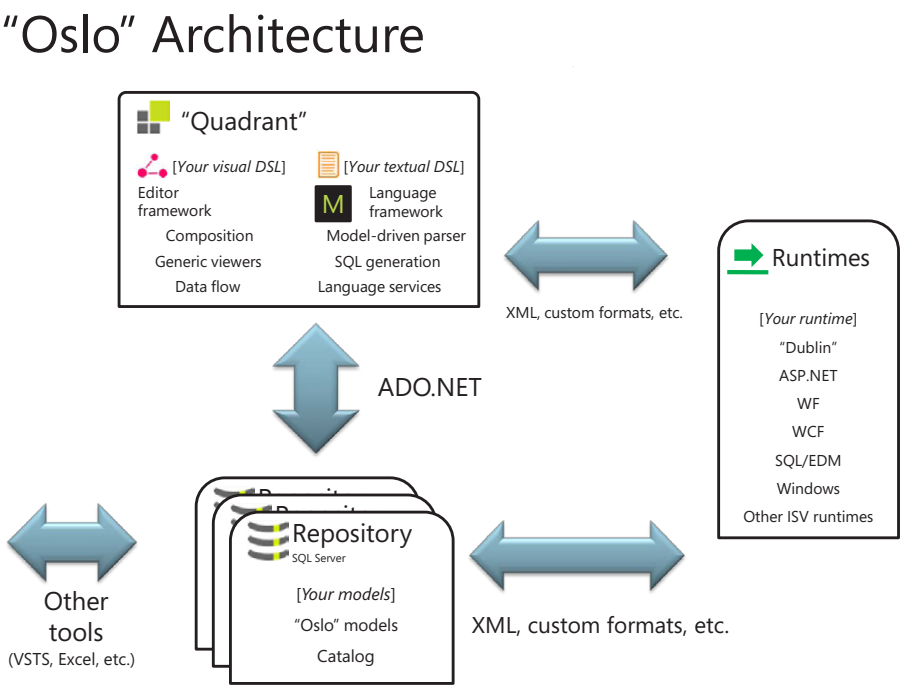
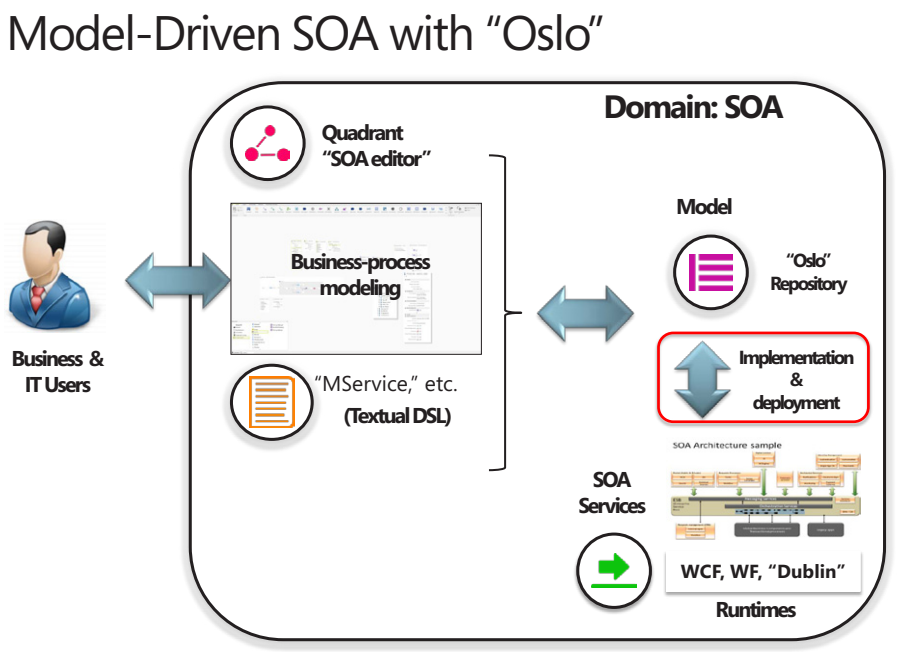


Figure 5: MDD with "Oslo"



"For me, personally, 'Oslo' is the first step in my vision to make everyone a programmer (even if they don't know it)." (See Doug's blog.)

This is really a key point, if we are talking about MDD; and, from Doug's statement, I can see that the "Oslo" team supports it. Almost everyone should be able to model applications or SOA—especially business experts, who have knowledge about their business processes. This will provide real agility to application development.

Model-Driven SOA with "Oslo"

Now, we get to the point: *model-driven SOA with "Oslo."*

Model-driven SOA is simply a special case within MDD/MDE. Therefore, a main goal within "Oslo" must be the ability to model SOA at a very high level—even for business experts (see Figure 5).

The key point in Figure 5 is the implementation and deployment of the model (the highlighted square that is on the right). "Oslo" must achieve this transparently. With regard to model-driven SOA (and MDD in general), the success of "Oslo" depends on having good support from base-technology runtimes (such as "Dublin," ASP.NET, WCF, and WF) and, therefore, support and commitment from every base-technology product group.

Possible Scenarios in MD-SOA with "Oslo"

Keep in mind that the following scenarios are only my thoughts with regard to how I think we could use "Oslo" in the future to model SOA applications. These are neither official Microsoft statements nor promises; it is just how I think it could be—even how I wish it should be.

Modeling an Online Finance Solution

In the future, one of the key functions of "Oslo" will be to automate and articulate the agility that SOA promised. So, *imagine that you are the chief executive of a large financial company. Your major competitor has started offering a new set of solutions to customers. You want to compete against that; but, instead of taking several months to study, analyze, and develop from scratch a new application by using new business logic, you can manage it in a few weeks—even only several days. By using a hypothetical SOA flavor of "Oslo," your product team collects the needed services (which are currently available in your company), such as pricing and promotions, to create a new finance solution/product.*

Your delivery team rapidly assembles the services; your business experts can even verify the business processes in the same modeling tool that was used to compose the solution; and, finally, you present the new competitive product to the public.

The key in this MD-SOA process is to have the right infrastructure to support your business model, which should not be based on "dark code" that is supported and understood only by the "geek guys" (that is, the traditional development team). The software that supports your business model should be based on models that are aligned with business capabilities that bring value to the organization. These models are based also on standards—based on reusable Web services that are really your software composite components. These services should be granular enough for you to be able to reuse them in different business applications/services.

Also, your organization needs tools (such as a hypothetical SOA flavor of "Oslo") to automate business-service models, even workflows—aligning those with new process models and composing existing or new ones to support the business process quickly. As I said, the key point will be how well-integrated "Oslo" will be with all of the plumbing runtimes (ASP.NET, WCF, .NET, and so on).

Finally, by using those model-driven tools, we could even deploy the whole solution to a future scalable infrastructure that is based on lower-level technologies such as "Dublin" (the next version of Microsoft application-server capabilities).

What Will "Oslo" Reach in the Arena of Model-Driven SOA?

The vision for model-driven SOA is that the software that supports your business model must be based on models and those models are aligned with business capabilities that bring value to the organization. Underneath, the models also must be aligned with SOA standards and interoperable, reusable Web services. Overall, however, business users must be able to play directly with those SOA models.

The question is not, "Will we be using Oslo?", because I am sure that we architects and IT people will use it in many flavors, such as modeling data and metadata—perhaps, embedding "Oslo" in many other developer applications such as modeling UML and layer diagrams in the next version of Microsoft Visual Studio Team System (although not the 2010 version, which is still based on the DSL toolkit); modeling workflows; and modeling any custom DSL—all based on this mainstream modeling technology. However, when we talk about model-driven SOA, we are not talking about that. The key question is, "Will business-expert users be using 'Oslo' to model SOA or any application?"

If Microsoft is able to achieve this vision and its goals—taking into account the required huge support and integration among all of the technical runtimes and "Oslo" (the "Oslo" product team really has to get support from many different Microsoft product teams)—it could really be the start of a revolution in the applications-development

What's Your Orientation?

by Pietro Nick Romano

Don't let your paradigm get in the way of seeing the best way to create great architectures.

Over the years, we have seen object orientation (OO), component-based development (CBD), service orientation (SO), and so on. We have OO analysis, OO design, OO haircuts... So, which paradigm is best? Should we be looking for objects, services, or both?

My essential message in this column is that "OO versus SO" is a false dichotomy that does not compare like with like, and that each paradigm has important things to offer.

Object Orientation: Is It Still a Valid Paradigm?

In their book *Object-Oriented Analysis and Design with Applications*, 3rd Edition (Upper Saddle River, NJ: Addison-Wesley, 2007), the venerable Grady Booch *et al.* do not even mention SOAs. So, what can OO offer today?

- **Analysis**—OO helps us manage complexity by hiearchically decomposing a system into entities and the operations that they perform.
- **Design/Implementation**—OO mechanisms such as encapsulation, inheritance, and polimorphism can facilitate reuse and future extension.

So, where might OO have limitations? I would underline two broad areas:

- **Requirements**—Requirements management should not be seen as an OO activity. Business people really are not interested in OO terminology. Trying to morph a list of requirements prematurely into an OO model is not helpful. Use cases complement—but should not replace—requirements (in UML, use cases are semantically defined as classfiers).
- **Distributed systems**—Strictly OO designs maintain state in objects and often hold chatty interchanges between highly coupled interfaces. This is inappropriate for heterogeneous environments, which are linked by unreliable communications. Martin Fowler's First Law of Distributed Object Design states it clearly: "Don't distribute your objects!"

Contrasting OO with SO

Pat Helland identifies the core SOA tenets in his seminal article "Data on the Outside vs. Data on the Inside." Let us contrast these tenets with the OO approach, as follows:

1. **Boundaries are explicit.** Services should have interfaces that are more coarsely grained than objects, with a view to reducing coupling and the number/volume of message exchanges.
2. **Services are autonomous.** The tighter interface coupling in OO implementations reduces the autonomy of objects.
3. **Services share schema and contract, not implementation.** Objects that are collaborating within a system generally do share implementation technology.
4. **Service compatibility is based on a policy.** Object compatibility is traditionally binary in nature—depending on hardcoded implementation.

Implementing SOAs by Using WCF (While Not Forgetting OO)

The message of this column is that OO can still help us implement SOAs. Windows Communication Foundation (WCF), which is part of the Microsoft .NET Framework, takes the best of both worlds and allows us to build SOAs while not discarding the still-useful mechanisms that OO provides for internal implementation—for example, the following:

- **Contract first development**—We define interfaces as in OO and decorate them later with the appropriate **ServiceContract**, **OperationContract**, and other attributes.
- **Structured OO error-handling**—WCF service operations can be decorated with one or more **FaultContract** attributes, which allows us to throw a strongly typed, generic **FaultException(TDetail)**.
- **Extension mechanisms**—The fundamental OO mechanisms of inheritance and polimorphism allow us to extend existing code efficiently through subclasses and interfaces.

For more information, visit my blog.

Pietro Nick Romano (nickr@microsoft.com) is an IT Architecture and Planning Advisor with Microsoft Services, Spain.

field. It will be like changing from assembler and native code bits to high-level programming languages—even better, because, as Doug says, every person (business user) will be, in a certain way, a programmer, because business-expert users will be creating applications. *Of course, this has to be a slow evolution toward the business, but "Oslo" could be the start.*

Conclusion

SOA has to evolve toward a more agile design, development, and deployment process. Most of all, however, SOA must close the gap between IT and business.

Model-driven SOA can be the solution to these problems. So, with regard to MDD implementations, "Oslo" is Microsoft's best bet to reach future paradigms for both MDD and model-driven SOA (a subset of MDD).

About the Author

César de la Torre Llorente is an Architect in the Microsoft Developer and Platform Evangelism organization. Before joining Microsoft two years ago, he cofounded Renacimiento Sistemas S.L. (an IT consultancy company and Microsoft Gold Partner) and spent his career architecting, designing, and developing solutions for big customers in the IT industry. César is happy to receive feedback at cesardl@microsoft.com.

Follow up on this topic

- Microsoft Codename "Oslo" Developer Center

An Enterprise Architecture Strategy for SOA

by Hatay Tuna

Summary

This article discusses key concepts and methods that CIOs, CTOs, architects, and IT leaders can put into action pragmatically to help their organizations evolve their IT architecture and solutions, so as to meet ever-changing business needs and demands through service-oriented architecture (SOA). It presents a robust architecture foundation—built on proven and already successful methods and practices—to implement tangible service orientation in a consistent and repeatable fashion, with a focus on business priorities and alignment, to deliver predictable and measurable business value while maximizing value of IT investments.

Introduction

- Where to start and how to begin SOA
- How to align services to business needs and priorities
- How to identify and scope services
- How to enable agility and innovate through services
- How to implement SOA

These are the fundamental challenges that organizations who demand a realization of business value through SOA—whether it is in the form of innovation, growth, revenue, or operational costs—are facing today.

An inconsistent understanding of promises, disconnected approaches to implementation (even among departments within the same organization), and unrealistic expectations of products and technologies become impediments on the road from aspiration to success.

This article introduces and discusses key *concepts*, *principals*, and *methods* that architects can practically put to work immediately to help their companies overcome these challenges and lead their organizations in their journey through SOA implementation for a better outcome.

The concepts and the methods that this article presents are technology-agnostic and compatible with widely available methodologies; they are designed to complement existing capabilities for implementing service orientation on-premise, in the Cloud, or both. Samples are selected from a recent case study for a leading global-recruitment consultancy firm in which these concepts and methods were implemented successfully.

Concepts

SOA is not an *end*; it is not a production environment that organizations can buy or deploy, after which some singing-and-dancing technologies magically adapt to change and deliver out-of-this-world value.

In fact, SOA is an incremental and iterative *journey* that organizations need to experience through *ways* and *means* so as to achieve predictable *ends*.

Industry has done a fairly good job in terms of defining the *means*: standards-based Web services, advancements in middleware products, and integration patterns.

On the other hand, because of the bottom-up and technology-centric perspective that is predominantly driven by software vendors, organizations who want to adapt SOA are left in the dark with poorly defined or no *ways*—methods and practices that will help them navigate though their SOA implementations.

The following are key, foundational concepts that organizations can embrace as part of their *ways* to tackle the key challenges that they face and build a robust enterprise architecture (EA) strategy for successful SOA:

1. Abstracting the *stable* from the *volatile*
2. Encapsulating the *volatile* within *services*
3. Managing *change* through *services*—the *journey*

Abstracting the Stable from the Volatile

Starting SOA is difficult, not to mention figuring out how and where to start implementation in an environment that is, in fact, a web of closely coupled people, IT systems, and business processes. This tight coupling makes change very difficult for organizations that are desperate to move forward and evolve.

Therefore, architects must acquire a new way of thinking and a view of the problem domain, so that they can slice and dice problems into stable and consumable parts that can be addressed through service orientation.

Let us look at the following scenario from the recruitment industry, in point:

> *A recruitment consultant sends a notification e-mail message to a successful candidate.*

The fundamental problem with such requirements is that they do not differentiate between *what must be done* and *how it should be done*. In other words, the problem and the solution are closely coupled.

How many times have architects been given problems in such a form and expected to develop solutions to change quickly and cost-effectively?

Capability-oriented thinking can help.

Capabilities Are Abstract

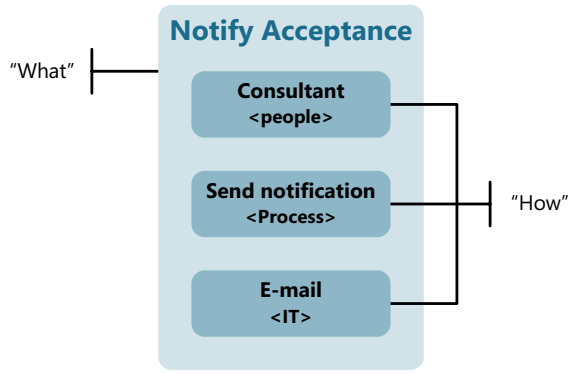
Capability is an abstract view of what an individual business function does; it defines the "what" of the function and is not concerned with the "how." Capability hides the bewildering details of the:

- *People* who perform the capability.
- *Information technology* that supports the capability.
- *Processes* that describe how the capability is performed.

Abstraction enables organizations and helps architects concentrate on the problem and, then, focus on solutions in terms of people, IT systems, and business processes.

Figure 1 illustrates the capability-oriented thinking about the problem.

Figure 1: Capability perspective



Capabilities Are Stable

Compared to business processes, organizational structures, and IT systems, capabilities are very stable in terms of change. People, IT, and processes are often very volatile and change over time. Capabilities, however—because they represent the abstract view—have a life span of centuries.

Let us go back to our scenario:

> *A recruitment consultant sends a notification e-mail message to a successful candidate.*

In fact, sending an email message is just one way of doing it. "How" it is done varies from organization to organization. However, "what" a recruitment company does has not changed much over centuries; they have always notified candidates of their successful job applications, and they will always notify candidates of their job-application status—even 100 years from now. How they do it, however, may change; in fact, it probably will change.

For example, as Figure 2 shows, a consultant might choose to leave an automated recorded voice message on a candidate's telephone or send an SMS message.

The nature of such evolution makes capability-oriented thinking an ideal foundation for service orientation to embrace change.

This is the promise of SOA.

Capabilities Have Clear Boundaries

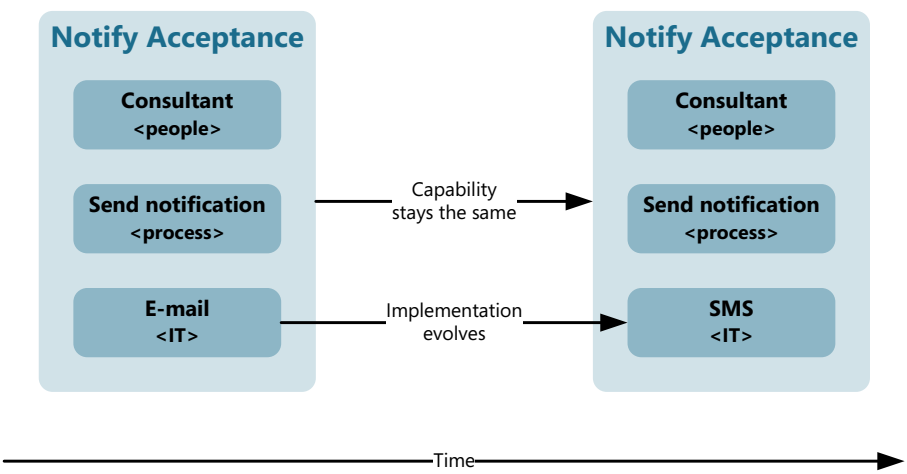
As a consequence of their abstract and stable nature, capabilities form clear boundaries that help architects define the implementation scope of their solutions. Such boundaries enhance the understanding of the relationships between capabilities and dependencies and between their implementations.

A *capability architecture*—also known as a *business-capability architecture* or a *business architecture*—is the decomposition and exposure of the business model that is represented as a *capability map* that is composed of *capabilities*. Capability maps represent different granularities of detail, which are also known as *levels*.

Let us look at the following capability-map sample that a recruitment firm might have:

- Develop products and services
- Generate demand
  - Sales
    - Generate new jobs
    - ...
  - Marketing
  - ...
- Deliver products and services
  - Recruitment
    - Generate new candidates
    - Manage applications
      - Submit application
      - Process application
        - Parse CV [curriculum vitae, or résumé]
      - Notify candidate
        - Notify candidate of application acceptance
        - Notify candidate of application rejection
        - Assess candidates
    - Manage shortlists
    - ...
- Plan and manage enterprise
  - Financial management
    - Billing
    - ...
- Manage collaboration

Figure 2: Change happens; evolution is inevitable.



Capability architectures have three inherent qualities in which architects will be very interested. A capability architecture:

- Abstracts the problem from the solution, where the solution is expected to change.
- Decomposes the problem into different levels of granularity of detail, with clear boundaries.
- Establishes a common language and a vocabulary that are shared between the business and the IT teams.

Building a Capability Map

It is the foremost duty of an architect to embrace this view, and to lead and drive conversations with business representatives—in terms of workshops, interviews, and so on—to develop a map that represents the "what" and not the "how" of the business.

It is important that each level in a capability map represent the same intensity of granularity, so that organizations can drill-down and gather details of a particular capability. For example, organizations might be looking for opportunities for automation. They might be investigating why a particular capability is not performing well. In other words, is it the people, the IT, or the process that is letting down the organization?

Prioritizing and Contextualizing

Capability maps help organizations proactively carry out purposeful analysis and assessment of capabilities to identify and prioritize capabilities that do or do not need attention, based on what is important to business.

Let us look at a sample. Which low-performing capability should the organization address first?

Match candidate and job Business value: High Competitive differentiator: Yes Performance: Low	Or	**Pay employees** Business value: Low Competitive differentiator: No Performance: Low

The answer most likely is "Match candidate and job." In fact, it delivers high value to business and is a competitive differentiator in the marketplace, when compared to "Pay employees."

This is how an organization might know where to start and align its SOA initiatives with business needs and priorities.

An analytic assessment of capabilities through their attributes provides benefits immediately:

- It enables better, rational, and defensible decision-making to help prioritize the problem.
- It provides contextual information to architects for a better understanding of the problem at hand (what is important to business, what is not performing well, why not, and so on).

Enterprise SOA Critical-Success Factors

by Adrian Grigoriu

Service-oriented architecture (SOA) is an enterprise-integration approach that consists of service definition, orchestration (BPMS/BPEL), description (WSDL), registration, discovery (UDDI), and distribution (ESB) technologies. Nevertheless, SOA is more than IT, although its origins are in IT. From a business viewpoint, it is a way of structuring a business as clusters of loosely coupled services.

Besides agility and reusability, SOA enables the following:

- Business specification of processes as orchestrations of reusable services
- Technology-agnostic business design, with technology hidden behind service interfaces
- Contractual-like interaction between business and IT, based on service SLAs
- Accountability and governance that are better aligned to business services
- Untangling of application interconnections by allowing access only through service interfaces, which reduces the daunting side effects of change
- Reduced pressure to replace legacy and extended lifetime for legacy applications via encapsulation in services
- Cloud-computing paradigm that uses Web services technologies that make possible service outsourcing on an on-demand, utility-like, pay-per-usage basis

Nonetheless, SOA harbors developments that are in the scope of enterprise architecture (EA). It does not specifically address IT alignment to business and strategy, documentation of the as-is enterprise state, or guidance for the development program (as EA frameworks do).

Because both SOA and EA are usually initiated by IT, the lack of both engagement by business stakeholders and support of top management can foil the success of SOA.

SOA does require a large enterprise reengineering effort that has consequences at all EA layers: business, applications, infrastructure, and organization.

The enterprise SOA critical-success factors (CSF) should be primarily approached as a business development, a business architecture, a way to structure the enterprise, and a style of target EA—and only then as an IT-integration technology. Also, the enterprise SOA CSF might succeed only if it is developed inside an EA development, because SOA does not cover the enterprise transformation process.

Driven only by IT, both SOA and EA are prone to fail; the engagement of the business stakeholders and the support of top management are keys to their success. The enterprise SOA CSF will demand business-process reengineering, new governance around services, and (ultimately) reorganization.

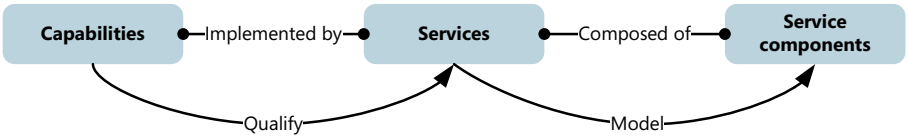
Enterprise SOA was pronounced dead, because the preceding CSFs were not really met. Application-level or Web-domain SOA, however, are alive and well.

When it has been implemented, an enterprise-wide SOA will become a competitive asset that is based on business services that are accessed independently of technology and geography, orchestrated agilely for change, and ready for outsourcing in the Cloud.

Adrian Grigoriu is author of the book *An Enterprise Architecture Development Framework: The Business Case, Framework and Best Practices for Building Your Enterprise Architecture* (Victoria, BC: Trafford, 2006). For more information, visit his blog.

Let us remember the basic principles: The "what" is very *stable* and the "how" is *volatile*. So, what is next? Why not hide the "how" behind *services*?

Figure 3: Connecting business to IT



Encapsulating the Volatile Within Services

Service-oriented modeling is a method that connects capabilities to their implementation via services. Capabilities represent an abstract view of what a business or business function does or intends to do. People, IT, and processes collectively make up the implementation of a capability. Such implementations are grouped logically as services.

Service-oriented modeling helps architects productively:

1. Qualify (or map) capabilities to services.
2. Model services, including the components in which they are composed.

Figure 3 illustrates the key concepts of service-oriented modeling.

Services are composed of service components that implement the capabilities. In other words, a service is a logical group of components that make up the implementation of a capability. A service represents a logical association between capabilities and service components; by doing so, it loosely couples what a capability is and how it must be implemented.

Consider the "Manage applications" capability from the previous recruitment capability-map sample. This capability is concerned with allowing candidates to send job applications with their CVs. Processing includes parsing candidate CVs, extracting candidate information, and matching this information to a job description. If there is a match, candidates are notified of their successful application submission. This saves valuable time for a consultant, who can actually focus on candidates whose profiles fit the job at hand. Otherwise, a rejection notification is sent to candidates whose profiles do not match the job.

After a candidate and job are matched, the recruitment cycle continues (interviews, offers, and so on):

- Manage applications
  - Submit application
  - Process application
    - Parse CV
  - Notify candidate
    - Notify candidate of application acceptance
    - Notify candidate of application rejection

Figure 4 illustrates not only the granularity of these capabilities, but also the interactions among them.

With this example in mind, let us look at how these capabilities might be mapped to services.

Qualifying Capabilities for Services

Qualification is the process of mapping capabilities to services that are composed of service components that will provide the implementation of the capability.

There are no predefined qualification criteria or contexts that fit all organizations that operate in all industries, or a magic formula that satisfies each and every business scenario. However, the characteristics of capabilities—such as their clear boundaries, purposes, and other attributes—come in really handy when capabilities are being qualified for services.

Figure 5 represents a possible mapping between the "Manage application" capabilities (illustrated as *lollipops*) and the services that encapsulate their implementation.

Moreover, because of the associating behavior between capabilities and their implementation, services also form a logical entity to define the expectations—such as performance, reliability, service-level expectations, quality-of-service (QoS) requirements, or key performance indicators (KPI)—of the capability implementation.

Thus, the performance of capabilities can be monitored and assessed against business objectives.

"Parse CV" from our recruitment scenario is a good example, in point:

- Ninety-percent accuracy (in terms of extracting quality information from CV documents)
- One-thousand CVs per hour

Such attributes and requirements provide architects with vital information and context in terms of selecting the right products and technologies, as well as developing successful architectures. In this case, for example, architects should be looking at technologies that can hit 90 percent accuracy, as well as perform (Parse 1,000 CVs per hour) and scale.

Qualification is a *rational*, *logical*, and *purposeful* activity. When it has been completed, architects can start thinking about developing service-oriented models.

Figure 4: "Manage applications" capability

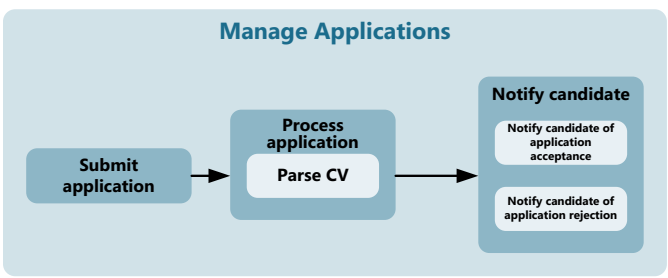


Figure 5: Capabilities mapped to services

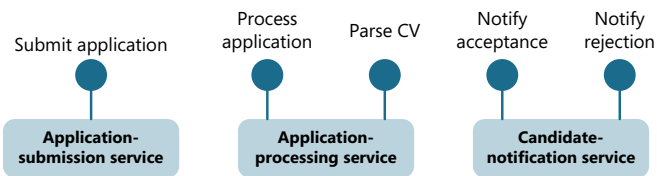
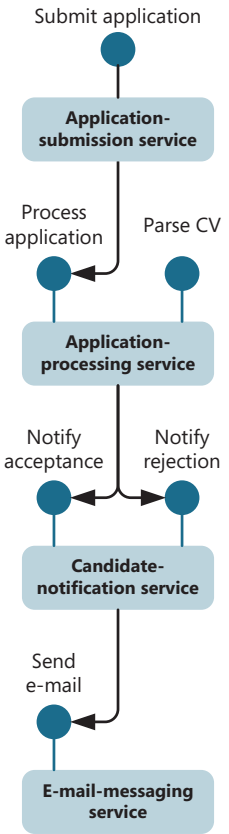


Figure 6: Structural model



Modeling Services

Modeling is the process of capturing and illustrating relationships, as well as the interactions between services. Service-oriented models include structural and behavioral models to capture and detail different aspects of the services.

For example, the "Candidate-notification service" depends on the "E-mail-messaging service" to send notifications via e-mail, as shown in Figure 6.

Structural models represent the decomposition of services in relation to other services—thus, allowing architects to capture dependencies between services, and understand and analyze the impact of change before it happens.

Behavioral models, on the other hand, represent interactions between services within context in terms of business processes, scenarios, use-cases, and so on.

Figure 7 illustrates a scenario in which a candidate applies for a job; the CV is parsed and successfully matched to the job; and the candidate is notified via e-mail.

Models that are developed at the service level, as opposed to detailed technical models, provide an easy-to-understand illustration and help architects capture the business requirements with clear boundaries, while hiding the complexities of the implementation.

Figure 8 illustrates the logical composition of these services in terms of service components that implement the "Manage application" capability.

Table 1 presents the connections from capabilities to services to service components.

Service components can be a combination of devices—for example, mobile phones and printers; hardware components, such as networks and servers; and (last, but not least) software components, such as applications, Web services, and databases. All principals seamlessly apply, regardless of whether a service component is a singleton commercial off-the-shelf (COTS) application, a batch file that runs overnight, a set of Web services that run in the Cloud, or a legacy COBOL application.

Figure 7: Behavioral model

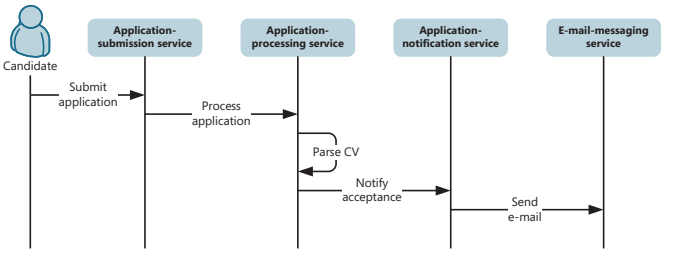


Figure 8: Services composed of service components

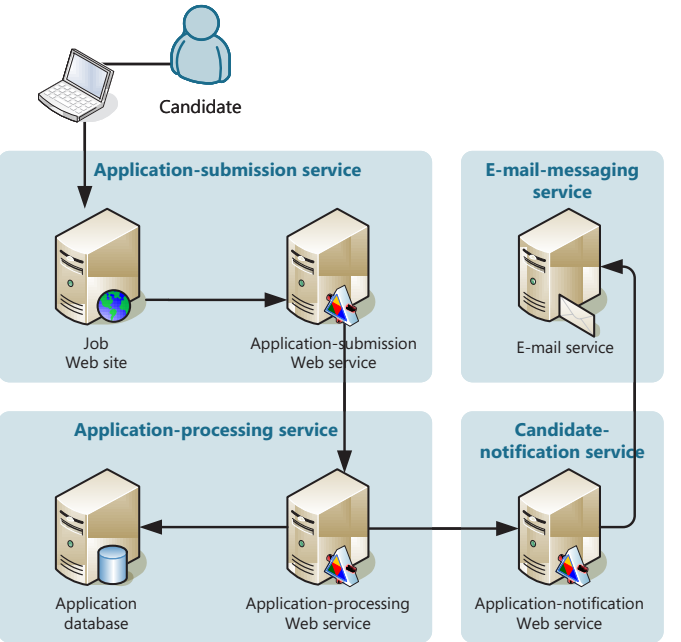


Table 1: Composition of services and the capabilities that they implement

Capability	Service	Service component/Behavior
Submit application	Application-submission service	*Job Web site:* Allows candidates to browse and apply to jobs through their Web browsers via the Internet
		Application-submission Web service: Receives applications via the Web site, but also ensures that other job boards can submit applications in the future
Process application	Application-processing service	*Application-processing Web service:* Processes applications, and parses candidate CVs to extract and match candidate information to job description
		Application database: Stores applications and candidate details
Notify candidate	Candidate-notification service	*Application-notification Web service:* Notifies candidates of the status, whether acceptance or rejection of their applications
E-mail messaging	E-mail-messaging service	*E-mail service:* Sends e-mail messages

Four Pillars of Successful SOA Adoption

by Dennis Smith

Service-oriented architecture (SOA) has been adopted successfully across a variety of domains. However, a number of high-profile failures have caused some to say that "SOA is dead." In reality, the proponents of this statement are saying that SOA as an architectural style makes a lot of sense but, unfortunately, it has been poorly adopted, and expectations have been poorly managed.

The way in which the four pillars of SOA adoption are managed can make the difference between success and failure:

1. **Create the right adoption strategy, and set realistic targets.** Develop a strategy to meet critical business goals that are related to SOA adoption. For example, creating intuitive portals or services that are related to customer information can satisfy a goal to increase the information that is available to customers. Combine this strategy with a plan for initial pilots, followed by systematic expansion to larger parts of the enterprise. This approach reduces risk, enables gradual education and buy-in, and provides for midcourse corrections.
2. **Create effective SOA governance.** Develop and enforce organizational policies for identifying, developing, and deploying services. Maintain consistency and standard processes for development of services, SOA infrastructure, and applications that consume services. Develop runtime-governance policies and rules for deploying and managing service-oriented systems, monitoring the use of services, and enforcing rules and constraints.
3. **Evaluate standards and technologies for SOA implementation in the appropriate context.** An SOA implementation might use common technologies in novel contexts. Evaluate *in advance* whether a specific technology is appropriate for the task at hand. For example, T-Checks[1] provide a hands-on, contextual evaluation for determining the fitness of a technology and associated tools for a specific need. Focused, extremely simple experiments can validate specific technology claims early in the life cycle and at low cost.
4. **Recognize that development in SOA environments requires a mindset that is different from traditional development.** In service-oriented systems development:
   - Service consumers, usage patterns, and requirements might potentially be unknown to service providers.
   - Service providers, service consumers, and infrastructure developers might have distinctly different goals and needs; in fact, they might belong to different organizations. As a result, there is no central control over a single system.

These differences have a profound impact on the way in which software is funded, managed, and developed. They affect all life-cycle activities—from requirements through architecture and design, development, and system testing.

Dennis Smith is a Senior Member of the Technical Staff and Lead of the System of Systems Practice Initiative (SoSP) at Carnegie Mellon University's Software Engineering Institute (SEI). For more information, visit his Web page.

[1] Lewis, Grace A., and Lutz Wrage. "A Process for Context-Based Technology Evaluation." CMU/SEI-2005-TN-025, June 2005.

Often, it is not an option to ignore existing investments and perform a fresh start; architects must also ensure that existing assets are mapped to capabilities through services, so that they can be managed and reused—in this case, the "E-mail-messaging service" for sending e-mail messages.

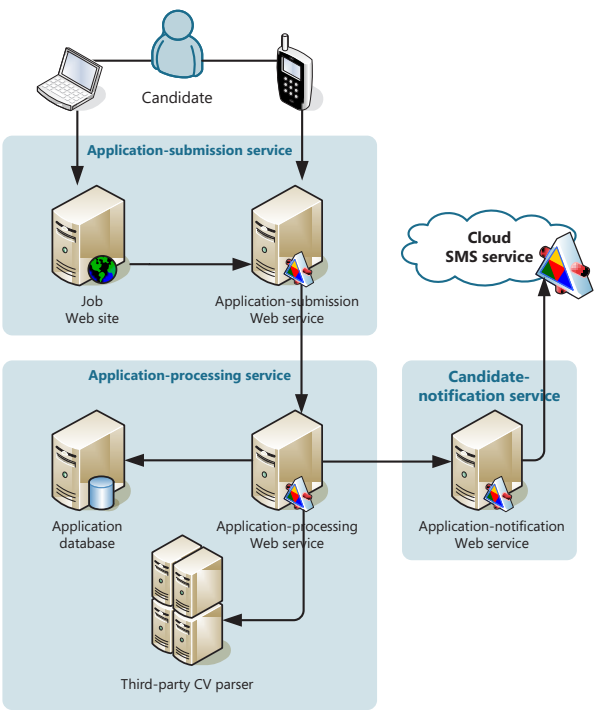
Let us build on our recruitment scenario and allow the change to happen. Consider that this organization is looking to transform its "Manage applications" capability by implementing the following changes that the business requires:

- Improvement in the accuracy of CV parsing from 90 percent to 95 percent by using a COTS specialist application
- Use of SMS messaging instead of e-mail messaging
- Extension of reach by allowing candidates to apply for jobs through a mobile application for smart phones

Figure 9 illustrates such a change.

Note the boundaries of services; although the implementation has changed to meet the latest business demands, the scope and boundaries that are inherited from the capabilities remained stable—helping the organization minimize the impact of change and enable room for evolution and innovation.

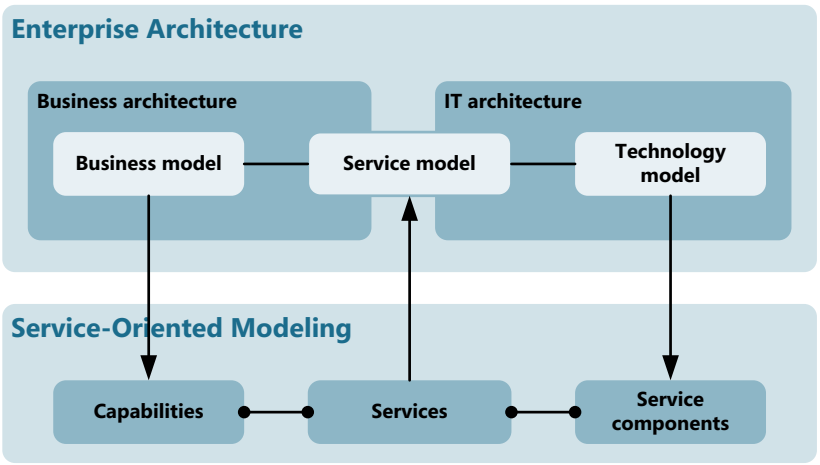
Figure 9: Boundaries are stable.



Mind the Gap!

Service-oriented modeling plays a vital role in bridging the gap between the business and IT—between business architecture (represented as business capabilities) and IT architecture (represented as service components)—by connecting an abstract view of what is

Figure 10: The missing link in EA—SOA alignment



done or what must be done to how it is done or how it must be done (see Figure 10).

Also, service-oriented modeling enables loose coupling between business models and technology models to embrace change and, ultimately, enable agility.

Service-oriented modeling has three qualities in which architects will be very interested:

- It connects and loosely couples business models to technology models via services.
- It hides complexities and transformation for IT behind services.
- It accelerates the delivery of services with robust service modeling.

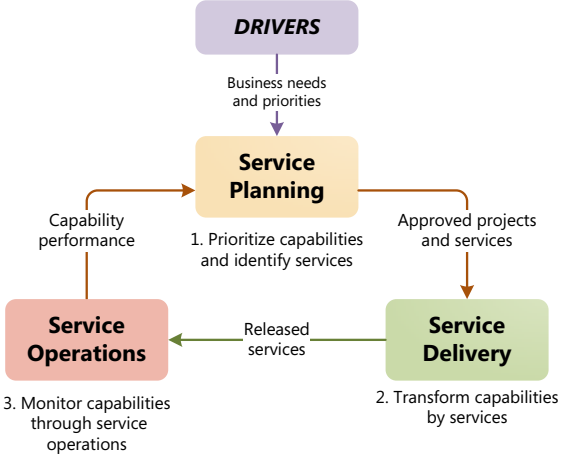
By now, the immediate benefits and practicality of *capability architectures* and *service-oriented modeling* might have become clear. Now, let us put these into context and discuss SOA implementation from a life-cycle perspective, and see how these methods can be performed in interactively and incrementally.

Managing Change via Services

Every change has a life cycle, from identification of the business need to implementation to verification of the success after the change. Figure 11 illustrates the key phases through which organizations must go during the life cycle of such a change—transformation of capabilities through services.

Note that it is better to incorporate SOA-implementation phases (the journey) to existing IT functions, such as Microsoft Operations Framework (MOF), the Information Technology Infrastructure Library (ITIL), the Control Objectives for Information and related Technology (COBIT), and the International Organization for Standardization (ISO). It greatly helps organizations manage the life cycle of services in a structured and systematic fashion with which most IT teams are familiar and comfortable. Also, it allows organizations to leverage their existing people and processes and optimize their thinking for service orientation, without creating unnecessary organizational and political barriers.
Let us have a brief look at these phases.

Figure 11: The journey—from vision to value



Step #1: Prioritizing Capabilities and Identifying Services

Service planning is the key phase in which architects must perform *capability architecture* and *service-oriented modeling* to prioritize capabilities and model services. The purpose of this phase is to:

1. Identify and prioritize capabilities that require attention, based on business context.
2. Qualify priority capabilities for services and, thus, for service orientation.
3. Model identified services and their key components.

This is the phase in which fundamental challenges are addressed: where and how to start SOA, how to identify and scope services, and so on.

The primary deliverables are *approved projects* to deliver *services* that represent the desired *change*.

Step #2: Transforming Capabilities via Services

Service delivery is concerned with transforming capabilities—implementing desired change via services. Attributes and boundaries of capabilities, as well as service-oriented models, provide vital input to architects and into the delivery phase.

Because service-oriented modeling connects implementation all the way up to business, it enables end-to-end visibility and traceability throughout the delivery phase, and provides teams with a context for the work that they are carrying out.

Obviously, the ultimate purpose of service delivery is to deliver to production reliable services that implement the capability and represent the desired change.

Step #3: Monitoring Capabilities via Service Operations

Service operations focus on operating and monitoring services and their components that are deployed in production. They are concerned with the operational health and performance of services (composition of service components) that represent capabilities.

Organizations must ensure that key service characteristics—such as SLAs, QoS requirements, and KPIs—are continually monitored.

The key outcome of service operations is that the *health and performance of services*—representing the success of the service, whatever its shape or form—are key inputs and requirements for healthy service planning.

Changes in economy and competition, pressures to cut operational costs, the desire to innovate, and an ever-changing

business landscape all drive organizations to focus on priorities and realize value from each and every service that is deployed through increments and iterations. Therefore, here is where you go back to Step 1—not only to drive the next set of services, but also to measure the success of the services that you deployed in Step 2 and that you have been monitoring in Step 3.

Conclusion

Solutions to key challenges in SOA are not hidden in products or technologies; in fact, they can be unlocked by applying a framework of methods and practices (collectively illustrated in Figure 12) to help organizations navigate through SOA implementation.

Let us sum up by remembering the key concepts:

1. Abstract the *stable* from the *volatile*.
   Capability architectures help organizations build an abstract and stable view of the business and expose areas that require attention. They help architects abstract and decompose the problem into consumable elements that have clear boundaries.
2. Encapsulate the *volatile* within *services*.
   Service-oriented modeling helps organizations connect and align IT to business. It helps architects map capabilities to services—and, then, to IT implementation—and model these services, while hiding the complexities of technology.
3. Manage *change* through *services*. SOA is the journey, not the end.
   Organizations must go through incremental and iterative cycles to implement SOA with business needs and priorities in mind.

Whether it is a business capability or an IT capability—and whether solutions are implemented on-premise, in the Cloud, or both—capability architecture and service-oriented modeling both greatly help an organization navigate through its SOA journey. They can dramatically increase the chances of an organization to succeed in delivering a required change—from innovation to optimization to outsourcing to automation to the Cloud—via service orientation, so as to realize the desired value and benefits of SOA.

The *people*, *processes*, and *IT* that make up implementation of a capability will change; they always do and always will. Loosely coupling what is done (capability) and how it is done (service components) via logical associations (services) provides a foundation for alignment, innovation, productivity, and agility.

Loosely coupling business from IT first and, then, loosely coupling systems will enable innovation and productivity that business desperately demands and which cannot be achieved by loosely coupling systems only.

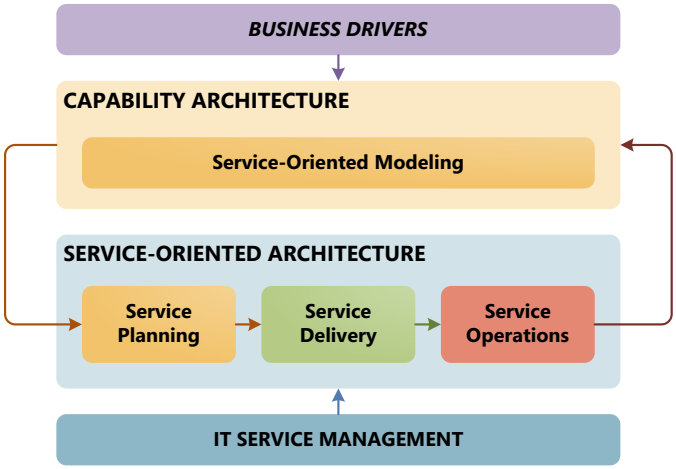
Resources

Microsoft Corporation. "Michael Page International MPI." Microsoft Case Studies, July 2009.

Microsoft Corporation. "Service-Oriented Modeling." Architect Insight Conference 2009, May 2009.

Microsoft Services Business Architecture (MSBA) and Microsoft Services Service-Oriented Modeling (MS SOM) are solutions that are offered by Microsoft Services.

Figure 12: The big picture—an EA strategy for successful SOA



Acknowledgements

Special thanks to my brother, Kutay Tuna (Microsoft Services), for tirelessly reviewing everything that I threw at him and helping to bring out the lousy author in me.

About the Author

Hatay Tuna (hatayt@microsoft.com) is an award-winning Principal Architect in Microsoft in the UK. He specializes in Software + Services, Service Oriented Architectures, Business Architectures, Modeling, Model-Driven Development and Architectures, Software as a Service, Service Bus Architectures, Process Automation and Optimization, Event-Driven Architectures, and Software Factories. Hatay has a strong passion for innovation, cutting-edge technologies, and future industry trends. He has successfully delivered several solutions in the Government/Public Sector, the Financial Services space, and the Health and Retail industries. Also, he regularly presents at internal and external events, such as TechReady, SOA & Business Process, and Architecture Insight Conferences. Besides work, Hatay enjoys spending time with his family, playing on Xbox 360, watching movies, and reading strictly technical books.

Follow up on this topic

- Microsoft Services. Architecture and Planning.

Enabling Business Capabilities with SOA

by Chris Madrid and Blair Shaw, Microsoft Consulting Services

Summary

This article describes some methods and technologies for enabling an SOA infrastructure to realize business capabilities and track dependencies between the two, so as to gain increased visibility across the IT landscape.

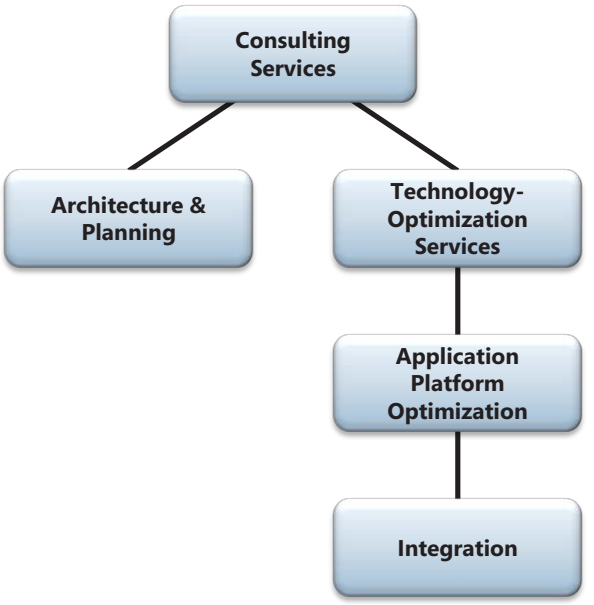
Introduction

Evolving from business-process reengineering (BPR), business-process management (BPM) was an established discipline well before service-oriented architecture (SOA). Initially, enterprises viewed the two as distinct—often, establishing separate teams that leveraged disparate technologies. Since then, SOA is less about leveraging Web services for faster integration and more about providing an abstraction of information-technology (IT) resources to support business activities directly. This maturity in SOA thinking brings it closer in alignment with BPM.

Enterprises now see this alignment and frequently combine new SOA and BPM projects; however, challenges still exist. While activities in a business process can be implemented as discrete services, there is usually no direct connection between BPM model artifacts and SOA model artifacts, which makes traceability difficult. Enterprise architects have a strong desire to see which processes would be affected if a given service were modified. At the same time, business owners want to see how their investments in SOA are faring. Ideally, they would like to gain visibility into which processes and services support a given business capability such as "Order fulfillment."

This article explains how Microsoft Services might provide enterprises with the visibility that they want through the application of service offerings (see Figure 1) that would leverage any existing Microsoft Services Business Architecture deliverables. Specifically, the article focuses on how Architecture and Planning services might feed directly into technology-optimization services. By weaving existing Integration (SOA) offerings—part of the Application Platform Optimization (APO) model under Business Technology Optimization (BTO) services—with emerging concepts that are used to drive future requirements, we hope to paint a picture of how you can enable business capabilities today on our stack and how this will only get easier over time.

Figure 1: Services offered by Microsoft Consulting



The Integration offerings are built around the concept of enterprise layers that describes relationships among processes, services, and data in the enterprise. The specific mesh of enterprise-layer items and their relationships in a given environment is referred to as an *enterprise service model (ESM)*. To deliver this ability, we start by mapping items in our ESM to capabilities in a capability model, as described by Martin Sykes and Brad Clayton in their article titled "Surviving Turbulent Times: Prioritizing IT Initiatives Using Business Architecture" in the July 2009 issue of *The Architecture Journal*. When the mapping is complete, we have a dependency map that we can use to identify which IT resources support a given capability and to understand which capabilities are affected by IT resources. In this case, an IT resource might be a Web service; but it might also be a message queue (MQ) or a mainframe Customer Information Control System (CICS) transaction.

We will continue to use the fictional retail bank, Contoso Bank, which was previously introduced to provide a context for discussing the concepts of enterprise layers and the ESM. As with all businesses, Contoso Bank must perform employee onboarding. The onboarding of an IT resource begins when a start is established after a candidate has accepted an offer and cleared the background check. Onboarding ends when the employee has a telephone number, e-mail address, laptop computer, and cubicle; has registered for HR benefits; and is set up in payroll. Several multistep processes are executed by different groups in the organization to fulfill employee *onboarding*. In an effort to reduce costs and increase productivity, Contoso Bank has a strong desire to reduce the time that it takes to onboard employees, so that they can start at their jobs more quickly.

Figure 2: Enterprise layers

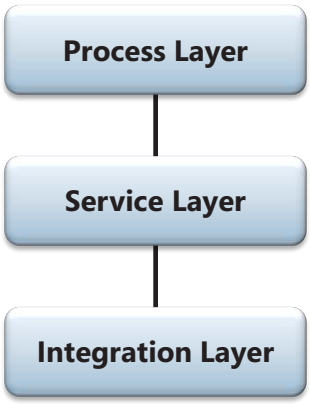
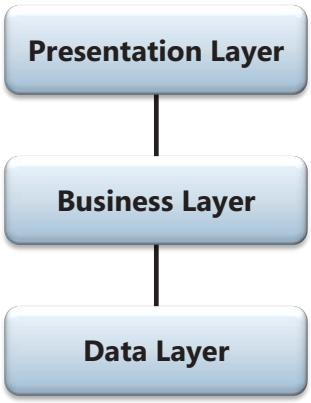


Figure 3: Application layers



Enterprise Layers

The EA team at Contoso Bank structures its decisions and activities around an enterprise-layer model, as shown in Figure 2. The model that Contoso Bank adopted consists of the following:

- Process layer
- Service layer
- Integration layer

The model that is shown in Figure 3 is an evolution of the distributed- (or *n*-tier–) application architecture paradigm of the following:

- Presentation layer
- Business layer
- Data layer

The enterprise-layer concept extends these areas of concern across multiple applications. Whereas the presentation layer in a single application represents the actionable interface to invoke business logic, it is often asynchronous events in a process that invoke business logic in an enterprise ecosystem. The business layer encapsulates business logic to a specific application. Frequently, enterprise services must coordinate interaction with multiple business services to fulfill an activity step in a process.

The integration layer is where traditional data jobs and enterprise application-integration (EAI) activities take place. The enterprise-layer concept provides enterprise architects with an opportunity to prescribe policies across a given layer and gives them a framework in which to think about dependencies across the entire IT landscape.

After several years, the application portfolio of Contoso Bank is more consistent in approach and architecture, but it still struggles to align with the business. An additional relationship is missing: *capabilities*.

Business Capabilities

Capabilities do not represent an IT resource or a group of IT resources. They are purely business abstractions that can be accomplished or that are wanted. A given capability might depend on additional capabilities that are to be delivered. The Contoso Bank capability model includes the Onboarding capability. As shown in Figure 4, this capability depends on the following child capabilities:

- Hardware Provisioning—Procuring and deploying telephone and laptop computer
- HR Registration—Registering with HR benefits and payroll
- Security Access—Creating badge and network credentials

Measuring the ROI of Your SOA

by Jeff Niblack

The following are two value-based metrics that you should consider the next time that you are asked to report on the IT spend for the SOA initiative of your company.

Consumer-to-Service Ratio

This is a useful metric to track primarily, because each reuse averts the costs of developing, operating, and maintaining a new single-purpose service. As the number of consumers increases for a service, the return on the costs—both initial and ongoing—that is attributed to that service increases.

However, use caution when you report this metric. As the number of consumers increases for a service, the reliance on that service increases, which exposes the enterprise to an increased consequence of failure. Because scalability is the result of design, implementation, and infrastructure choices and investments, as more consumers begin to employ and rely on a service, the workload of that service increases and could potentially move beyond the limits that the service was expecting—thus, increasing the likelihood of failure. Finally, with consumers converging into a single point, the ability of a service to mature over time becomes increasingly more challenging and costly, as any change to the service requires an increased investment in impact analysis, regression testing, and coordination across the enterprise.

Enterprise Adoption Rate

Tracking how services are leveraged across each organizational unit and rolling that information up to the enterprise level provides the enterprise adoption rate. This metric indicates how pervasive the use of services is across an enterprise. However, this metric is not necessarily easy to quantify. For example, given an enterprise that comprises five organizational units, if only two of the units are leveraging services, is the adoption rate of our enterprise at 40 percent? What if 80 percent of the total IT spend is within these two organizational units? Is the adoption rate of our enterprise now at 80 percent?

A more accurate and measurable approach for reporting the enterprise adoption rate begins with a catalog of IT resources that span the enterprise. After determining how many of the applications within the catalog are actually leveraging services, determine the total number of applications that are candidates for an SOA implementation or integration. The product of this review (applications using services/total candidate applications for leveraging services) is a realistic enterprise adoption rate. If the data is available, integration of this metric with actual IT spend provides the weighted enterprise adoption rate, which is an even more compelling value-based metric.

Jeff Niblack (j.niblack@ratchet.com) is a Team Manager and Architect at Ratchet. He has more than 20 years of experience in dealing with application architecture and design.

SOA Success Factor: Service Virtualization Supports SOA Governance

by Rama Vangipuram

Most every SOA implementation ends up failing to live up to what an enterprise expects of an implementation. The implementation devolves into an unmanageable spaghetti mess of services, with no foresight into governance, visibility, or control. Any benefit that is to be gained and potential return on investment (ROI) can be lost, as IT organizations revert to their old practices. What can an enterprise do to ensure a successful SOA implementation?

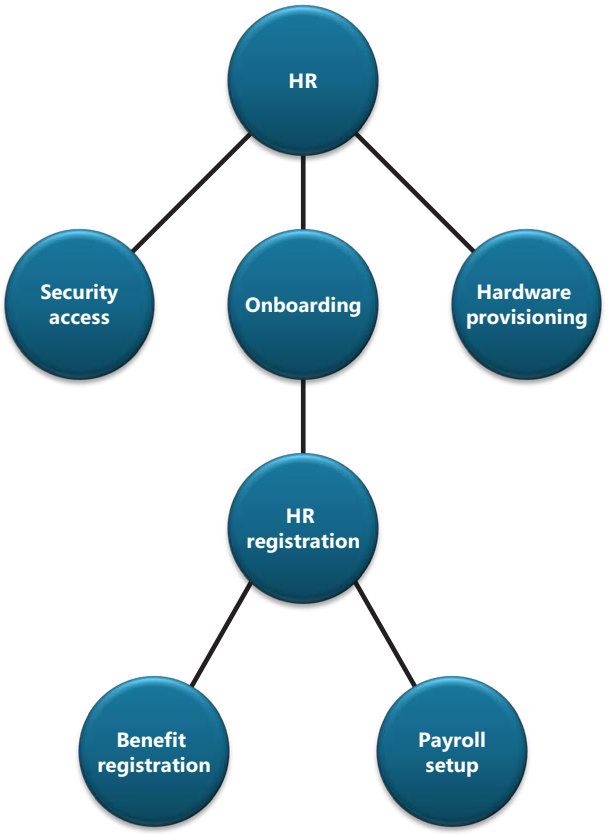
There are two keys to make it successful. The first is to have governance over the implementation, and the second is to extend your SOA implementation by leveraging service virtualization. The added benefit of service virtualization is that it helps govern the SOA implementation by providing a dynamic façade by which services are exposed—allowing for reusability of services, and providing policy enforcement through policy assertions. The ability to abstract policy enforcement, such as security or metadata publishing from the service implementation to a policy assertion, is a great advantage.

Abstraction allows developers to focus on the specific business functionality or data to expose as a service, without having to worry about how to enforce certain policies or standards in the service. One example is abstraction of security functionality out of the service to a policy assertion, which enables the development team to have a common security framework to leverage, as all that they would need to do is assign the security policy to their virtual endpoint. Virtualization also frees the developer from having to worry about how a service will be exposed, as logical services can be created that can be exposed through multiple protocols. So, you might ask, "How does this help govern my services?"

There are several key factors that help create a governance model that will stick. Have a board for oversight—developing a framework (standards are the foundation for SOA), creating policies, and enforcing them—and visibility into your services ecosystem. Service-virtualization tools, such as the Microsoft Managed Services Engine (MSE), help in all of those key areas. Virtualization of services helps SOA governance by providing the visibility of the services ecosystem through tools such as MSE—allowing for standardization, such as a security framework through policy assertions—and helps enforce those standards and policies.

Rama Vangipuram is a Senior Manager and SOA Architect at Pariveda Solutions.

Figure 4: Capabilities of Contoso Bank



The business might choose to model these capabilities at a lower level of granularity and include further child capabilities. In our example, the HR Registration capability could be composed on a Benefit Registration capability and a Payroll Setup capability. These capabilities are pure business abstractions, so that there are no assumptions of how these capabilities are implemented—or even assumptions of whether they are implemented at all.

Modeling Enterprise Layers and Capabilities

Artifacts in each enterprise layer can be modeled independently using existing tools in the disciplines of business-process modeling, capability models, and service modeling. Most enterprises would agree that there is benefit in developing their skills in these disciplines. Mature organizations that have established modeling methodologies face new challenges. Often, the business-process models, capability models, and service models are created by using different notations and tools. More importantly (at this point in the discussion), while the models are intended to represent the business and IT landscape, they drift apart quickly and lose value over time.

The Enterprise Service Model

Originally, the concept of an enterprise service model (ESM) existed to rationalize a portfolio of services. The ESM would include a conceptual model of services in the IT environment and services that were planned to be developed and deployed. Challenges immediately existed with the model becoming outdated, because IT employees manually modeled changes in the environment. Also, as enterprises saw SOA aligning closer with BPM, the model had to account for

processes and business capabilities—which it often did not, so that its value diminished. The concept of an ESM had to be expanded to keep the model in sync with reality and include capabilities and processes.

Today, our concept of an ESM aims to represent the instances of artifacts that are found in the enterprise layers and their relationships with capabilities that are found in a capability model (see Figure 5). Not only are the relationships for specific systems, resources, operations, and capabilities defined, but the ability to affect the real instances is an important goal.

Each capability can be enabled by one or more processes or service operations. Each step (activity) in a process can be mapped to a service operation. In cases in which an activity maps to multiple services, a façade should be created to manage the coordination of or aggregation to where the activity logically maps to a single service. In our Contoso Bank example, we are attempting to enable "onboarding." As described earlier, the Onboarding capability comprises child capabilities. However, not all capabilities need to be mapped. If a process or service can be identified that fulfills the entire Onboarding capability, it would be the only one that is mapped. Our customer does not have a single service to fulfill the Onboarding capability.

In the future, Contoso Bank might invest in creation of an onboarding process that would execute in its data center. At this point, the problem is that it is completely conceptual; it might exist in diagrams or described in documents, but there is no concrete IT-resource representation or execution of capabilities. Capabilities exist for traceability and for demonstrating a return on investment (ROI) to business stakeholders. They also aid in strategic architecture planning, by helping IT identify which processes might be leveraged to enable a capability and identify new processes that should be created.

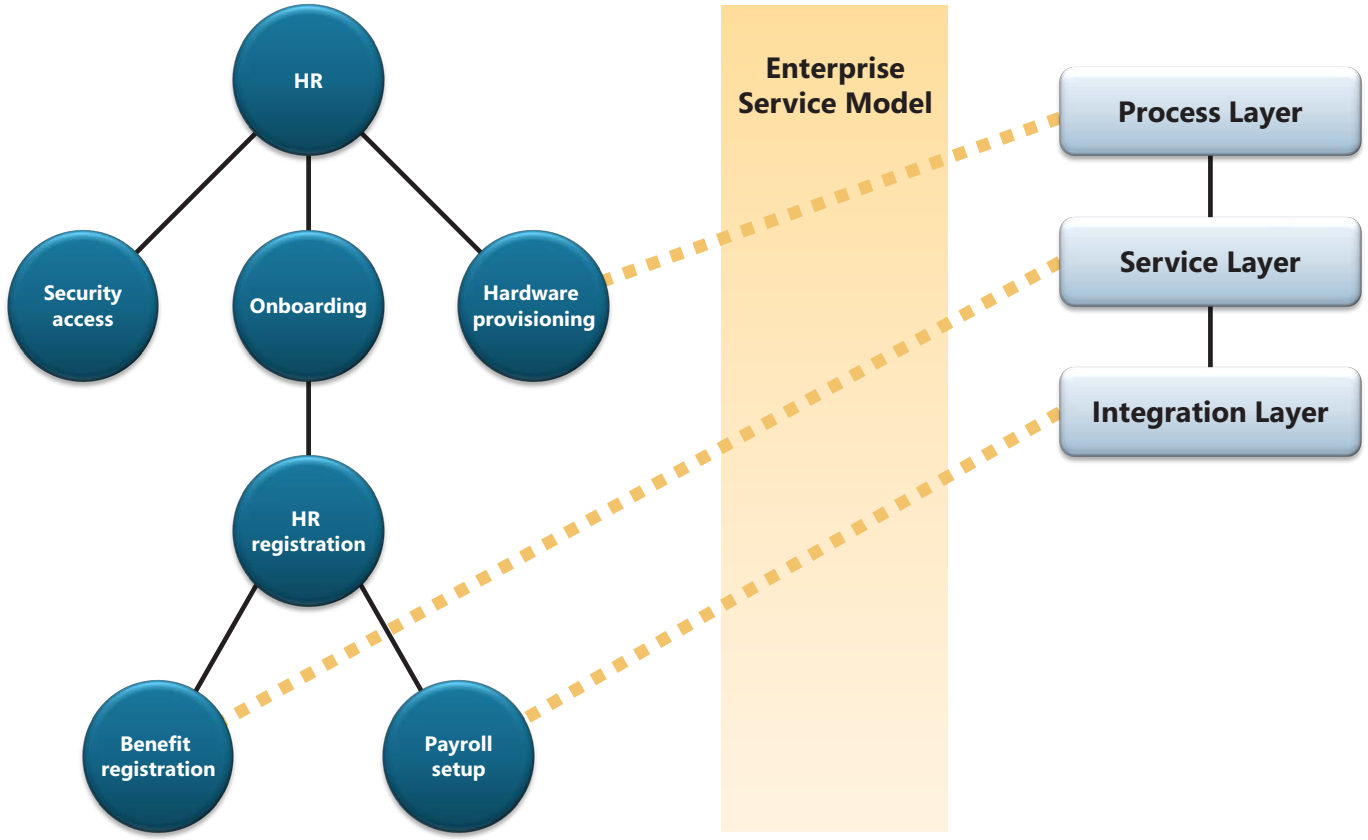
Microsoft Service-Oriented Infrastructure

The process of mapping capabilities to IT resources is powerful. The mapping might help identify gaps in a service portfolio and provide business traceability to service development. As this model become more concrete, moving from diagrams to metadata, it becomes more powerful. Enterprise assets can now develop stronger links—not just at design time, but at runtime.

This is one of the goals of the "Oslo" modeling technologies. "Oslo" is the code name for Microsoft's forthcoming modeling platform. "Oslo" extends beyond squares and rectangles that cannot be shared across tools to a set of technologies that are focused on enabling gains in productivity. Gains in productivity will come in the form of sharing metadata that is captured through diagrams and runtime environments that will execute metadata that is stored in the repository.

The Integration offering that we introduced earlier includes the service-oriented-infrastructure (SOI) offering that consists of the Microsoft Services Engine (MSE), which allows customers to start establishing their ESM today. The MSE stores information that is related to IT resources in its metadata repository. This metadata includes the location, types, and policies for Web services. It can also include the metadata that is necessary to invoke a CICS transaction through Microsoft Host Integration Server or the metadata that is required to drop a message in an MQ.

Figure 5: Mapping capabilities through the ESM to enterprise layers



As more metadata is imported into the MSE, a more complete picture of the ESM is created. Certainly, there is value in understanding what is in the environment. However, the value in MSE does not come from storing metadata, but from the ability to take this metadata and affect how the environment is constructed virtually. The MSE accomplishes this through its implementation of *service virtualization*. In this context, service virtualization is the abstraction of the address, binding, and contract between the service consumer and the service provider. Because we can hide where the service operation actually resides, change how we communicate with the service operation, and manipulate what the service operation looks like, we can create virtual services.

Adoption of the MSE by Contoso Bank allows them to take operations from disparate services and combine them to create a new endpoint. While Contoso Bank did not possess a single operation or process entry point to fulfill the Onboarding capability, they did have operations to start the hardware, HR, and payroll processes. By leveraging the MSE, Contoso Bank was able to project a virtual service that contained each operation—providing a simple façade that could be developed against, to fulfill the Onboarding capability.

Coming Enhancements

The MSE solution was developed over a four-year period by a team of architects in Microsoft Services who were working with customers and their scenarios. During this period, the MSE solution underwent several releases that added features; updated the ESM; and supported new operating systems, as well as new versions of Microsoft .NET Framework and Microsoft SQL Server. New features were developed in consultation with customers and as a result of frequent meetings with various Microsoft product groups. These meetings helped align the solution with product releases and validate the solution.

The result was the filling of a joint patent application that was related to service virtualization and acceptance of the MSE solution into the codename "Dublin" Microsoft Technology Adoption Program (TAP). "Dublin" is an extension of Windows Server that will provide enhanced hosting and management for Windows Communication Foundation (WCF) and Windows Workflow (WF) applications.

In the coming year, MCS plans to extend the ESM to include capabilities as first-class citizens. The ESM will be extended so that business capabilities can be associated with operations. When this is complete, we will be able to get end-to-end visibility—from the business capability all the way to the IT resources that deliver that capability. In our example, we find that the child capabilities are mapped to operations and that those operations are projected as virtual services. Someone who was examining the model would see the need to develop a workflow service to coordinate interaction between the three operations. After the development of that service, it too could be managed by the MSE and mapped to the Onboarding capability for dependency tracking.

As "Oslo" gets closer to release, the MSE will take it as a dependency and leverage the repository to store the metadata that represents the ESM. By utilizing "Oslo," we will map the virtualized service and operations to the Contoso Bank Onboarding business capabilities. A business analyst or domain expert can start with business capabilities and associate the services and resources that are utilized to perform the specified business capability. The addition of the "Oslo" modeling technology will be a natural extension of the ESM and MSE.

For additional information, please refer to César de la Torre Llorente's article titled "Model-Driven SOA with 'Oslo'" in this issue of *The Microsoft Architecture Journal*.

Conclusion

Services and applications that were developed over a period of months or even years at Contoso Bank and a variety of different technologies have been utilized. The developers at Contoso Bank followed common architectural patterns for distributed- and *n*-tier–application development. However, as their business evolves, they face the continued struggle of aligning applications with their business.

By focusing on a business capability such as "Onboarding," Contoso Bank is better able to align with the applications and business processes. Instead of having to write new applications or application adapters to map to the business capability, Contoso Bank found that by leveraging an MCS solution (the managed-services engine, or MSE), their developers could create new virtual services that align with the business capability. This is possible by modeling the existing services, endpoints, operations, and data entities with the use of the ESM. This process is quickly performed by importing existing services or applications into the ESM; then, new virtual services that align with the business capabilities are defined. Service virtualization enables the composition of multiple physical resources and operations into a virtual service that aligns with the business capability.

References

"An Introduction to Service Virtualization on the Microsoft .NET Platform." Microsoft Download Center. July 14, 2009. Download. Microsoft Corporation, Redmond, WA.

"Service-Oriented Infrastructure from MCS." Microsoft Services. Data sheet. October 23, 2008. Microsoft Corporation, Redmond, WA.

About the Authors

Chris Madrid (cmadrid@microsoft.com) is a Solution Architect in the Microsoft Consulting Services Global Practice. Focused on integration, he spends time with customers to grow their SOA maturity through service virtualization and service life-cycle management.

Blair Shaw (blairsh@microsoft.com) is a Senior Program Manager in the Microsoft Consulting Services Application Platform Optimization Service Line. He has spent the past four years working on SOA solutions with Microsoft customers and partners.

Follow up on this topic

- Microsoft Services. Architecture and Planning.

Service Registry: A Key Piece for Enhancing Reuse in SOA

by Juan Pablo García-González, Veronica Gacitua-Decar, and Dr. Claus Pahl

Summary

One of the promises of adopting a service-oriented approach in organizations is the potential cost savings that result from the reuse of existing services. A service registry is one of the fundamental pieces of service-oriented architecture (SOA) for achieving reuse. It refers to a place in which service providers can impart information about their offered services and potential clients can search for services. In this article, we provide advice for implementing an enterprise-wide service registry. We also discuss open issues in industry and academia that affect the management of service-repository information.

Introduction

The reuse of services greatly depends on the ability to describe and publish the offered functionality of the services to potential consumers (clients). A service registry allows you to organize information about services and provide facilities to publish and discover services.[1]

Universal Description Discovery and Integration (UDDI) and the Web Services Description Language (WSDL)—together with SOAP—are standards for describing services and their providers, as well as how services can be consumed:

- **WSDL**[2] provides a model and XML format for describing what a Web service offers. A service description in WSDL separates abstract-service functionality from details such as how and where the service is offered. While the abstract-service description includes *types* and an abstract *interface*, concrete details include *bindings*, a *service* element that includes all available implementations of the abstract *interface* at *endpoints*.
- **UDDI**[3, 4] provides an infrastructure that supports the description, publication, and discovery of service providers; the services that they offer; and the technical details for accessing those services. A core aspect of UDDI is how it organizes information about services and the providers of services. Information entities (UDDI data) are organized in a data model and stored in a UDDI service registry. *Inquiring* (search and lookup entries) and *publication* (publish, delete, and update registry–related information) are core APIs.

Figure 1 illustrates some relationships between a WSDL service description and information that is stored in a UDDI service registry.

Originally, UDDI was conceived to cover both publicly exposed services and services that were available within an organization. Currently, most existing implementations are internal to organizations. Service publication, discovery, and (finally) *reuse of services* is more complicated in an inter-organizational scenario; for example, additional legal and commercial agreements are often needed among parties.

Dedicated (public) UDDI service registries were criticized for their limitations (among other reasons) during service inquiry/discovery. Recently, however, Web search engines—which could be crawling publicly available WSDL documents—have raised promising expectations for discovering publicly available services.[5]

Designing an Enterprise Service Repository

This section proposes some design guidelines to develop an enhanced enterprise service repository. The focus is on improving the reuse of services over time in different IT projects. The aim is to increase service visibility to domain experts (often, this refers to a business-analyst role) and enhance service descriptions with practical information for architects. Business analysts, who have a less technical background but strong knowledge of the business domain, are frequently the early designers of new initiatives for incorporating or

Figure 1: Relationships between WSDL and UDDI

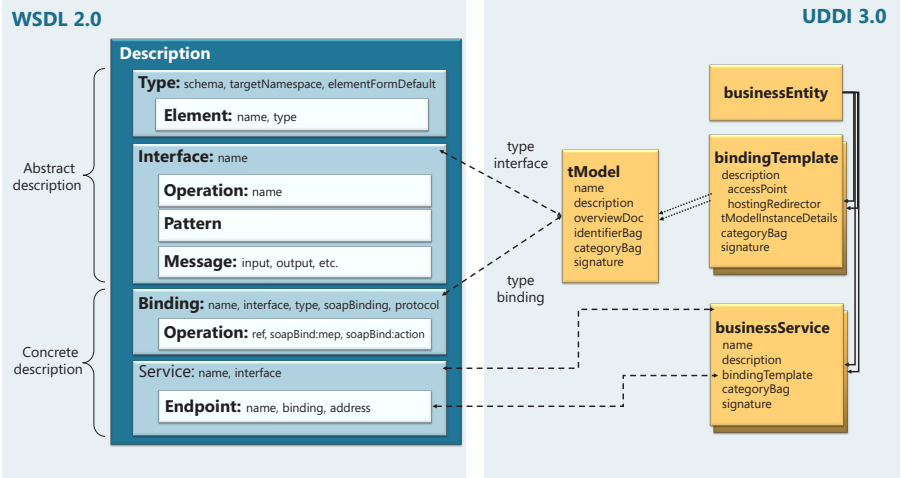


Figure 2: Service-based architecture and its relation to virtualization-pattern roles

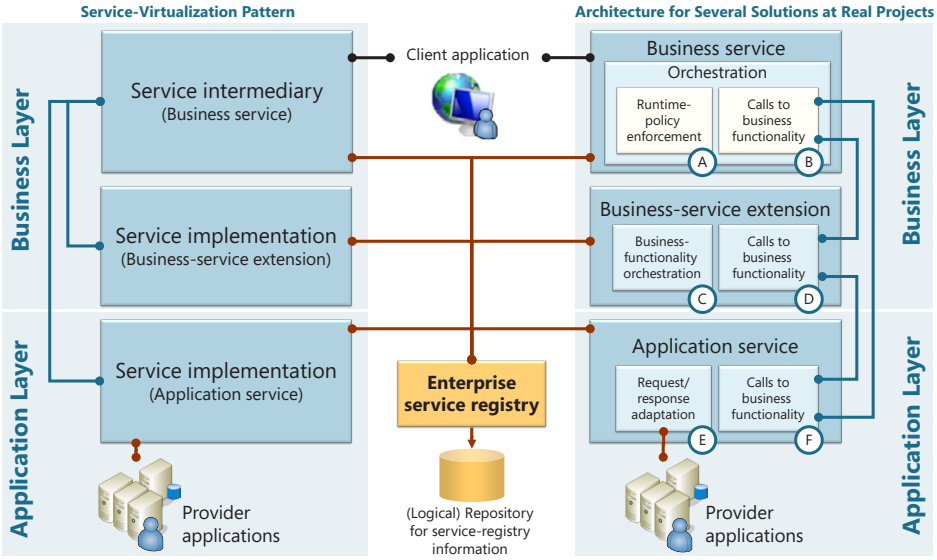
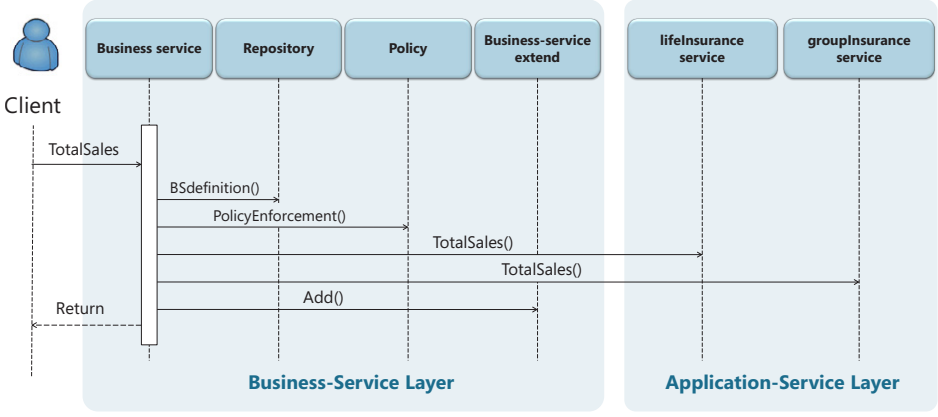


Figure 3: Main interaction among elements of the service-based solution from Example 1



modifying the software support at companies; they play a key role with regard to the reuse of services.

Enterprise Services

Enterprise service–based solutions involve different types of service. Following the separation of concerns that is addressed by the *service-virtualization* pattern,[6] services can act as an intermediate layer between the client and provider applications of the services. The virtualization pattern focuses on the abstraction of technical details—such as service-endpoint location, policy enforcement, service versioning, and dynamic service-management information from service consumers—which access an *intermediate* service level. Technical concerns are managed at an *implementation* level, at which the actual business logic is implemented.

Based on SOA initiatives in several companies, we can identify three types of service:

- A business service (BS), which client applications use for accessing the functionality that is implemented in provider applications.
- An application service (AS), which can be consumed by a BS to access the functionality of the provider applications.
- A business-service extension (BSE), which can be consumed by a BS to operate on different AS responses and consolidate a single answer that is sent to a BS. In turn, the BS delivers the consolidated response to the client application. The aggregator pattern[7] is core to the design of a BSE.

Figure 2 illustrates the main static relations among elements of an enterprise service–based solution, as well as their relationship to elements from the virtualization pattern. A service registry organizes the description of the three different types of service and their relationships. Client and provider applications interchange messages that are mediated by BS, BSE, and AS. The service registry manages (at the configuration level) the information that relates the different types of service. The information is persisted in a service repository and used at runtime by a BS to answer client requests.

Example 1

Let us consider a simplified BS that is used for calculating the total sales that are related to the *life-insurance* and *group-insurance* products of an insurance company. The total sales that are associated with *life-insurance* products are obtained from a *life-insurance* legacy application. Analogously, the total sales that are associated with *group-insurance* products are obtained from the *group-insurance* legacy application. Each legacy application exposes an AS (*lifeInsurance* and *groupInsurance*, respectively) that provides the total sales for each type of insurance product. A BS receives requests from clients who are asking for the total sales; afterwards, it calls the service registry that has the information that is required to enforce specific policies on messages and dependencies to an AS and a BSE. A BSE operates on the answers of an AS and provides a single answer to the BS that contains the total sales of the company. The BS, in turn, delivers this response to the client application.

Figure 3 illustrates the described interaction.

Enterprise Service Registry

The enterprise service registry (ESR) is a core element that organizes service information and supports the interaction among enterprise applications that provide and consume services.

Basic functionalities of an UDDI-based ESR can be enhanced by using:

- Service-dependencies management.
- Runtime-policy enforcement.
- Service versioning.
- Service-history data (logs) management.

A service repository persists the information and documentation that are logically managed by the service registry. Figure 4 illustrates the main information that is organized in an ESR, persisted in a service repository, and provided to end users through a Web-based user interface.

Services Descriptions

Services descriptions are core to the service registry. They determine how services can be discovered and subsequently reused:

- A BS is described at a high level—often, via textual descriptions in natural language and examples that facilitate understanding by business analysts.
- A BSE and an AS contain more technical details. A BS is implemented by at least one AS and also might involve a BSE. To associate a BS to one or more AS(s) and/or BSE(s), a dependency mapping is created and managed by the service registry.

Table 1 describes in more detail the information that is managed by the service registry:

- The main attributes that describe a BS are shown at the beginning of Table 1. A BSE and an AS share attributes (see middle of Table 1). The end of the table describes binding information that relates a BS to a BSE and an AS.
- Information that describes services and is independent of any registry implementation is shown in the **Service information** column, while information that is managed by the service registry is shown in the **Registry information** column. If information in the **Service information** and **Registry information** columns is the same, an X appears in the **Replicated information** column.
- The remaining columns indicate information that is relevant to different roles.[8] Business analysts and solution architects manage information about business and technical concerns, respectively. Both roles work at a project or business-unit scale. Enterprise and infrastructure architects manage service information from a global (enterprise-wide) perspective. While enterprise architects might be interested in managing (for instance) service versions, infrastructure architects care about providing the required infrastructure support to keep services running with the adequate quality of service (QoS), as defined in service-level agreements (SLAs).

Using the Enterprise Service Registry to Improve Reuse of Services

Based on our experience in a range of projects, providing simple descriptions about a BS, facilitating its access, and managing services dependencies have been key to improve reuse. For this purpose, an ESR was a core element.

- Business analysts who trigger new requirements for software support can improve their communication with solution and enterprise architects by referring to a BS that is described in the ESR. Based on the descriptions of the BS and its dependencies to an AS and a BSE, domain experts become aware of available functionality at back-end applications. From our experience, this has facilitated a shift from requirements that are specified in a vague manner to initial solution blueprints that comprise orchestrated services (created by business analysts). Long meetings between architects and business analysts can be reduced to short meetings or even telephone calls that refer only to information at the service registry.

Figure 4: Enterprise service registry (ESR)

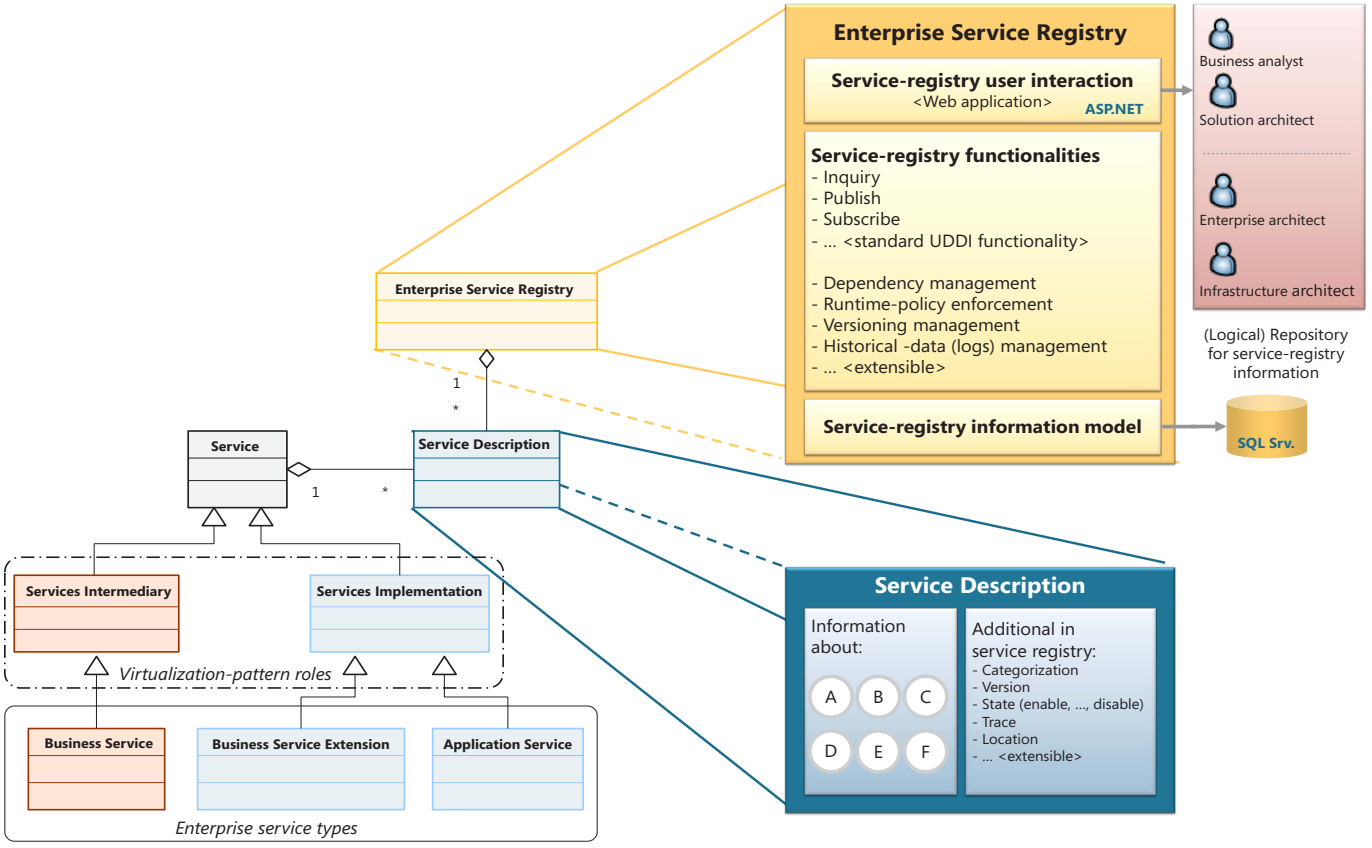


Table 1: Main service information at the ESR

	Information			Role			
Business service (BS)	**Service information**	**Registry information**	**Replicated information**	**Business analyst**	**Infrastructure architect**	**Solution architect**	**Enterprise architect**
Service ID: Key used to access the service from client applications.		X				X	
Service name	X	X	X	X			
Service description: Textual description explaining what the service does and some other business related information such as what business objective the service addresses and if some business rules apply when it is utilized.		X		X			X
Input messages: Textual reference for input messages required to execute the service		X		X		X	X
Output messages: Textual reference for output messages generated after the execution of the service. Error messages might also be specified.		X		X		X	X
Category: Category(ies) to which the service belong(s). For example, one categorization schema used to classify business services was defined based on organizational business units.		X		X			X
Application service (AS) and business-service extension (BSE)	**Service information**	**Registry information**	**Replicated information**	**Business analyst**	**Infrastructure architect**	**Solution architect**	**Enterprise architect**
Service name	X	X	X	X		X	X
Service description: Textual description—often, together with a link to a WSDL file.	X	X	X	X		X	
Binding information: Technical information to access the service and subsequently execute it.	X	X			X	X	
Operation: Reference to functionality being implemented by the service.	X	X				X	X
Policies: Listing of runtime policies applicable to the service and link to associated documentation. Policies might include timeout, bandwidth, and debugging information (among others).		X			X	X	X
Category: Category(ies) to which the service belong(s). For example, one categorization schemas used to classify application services was defined by the technology implementing it. Different categories of SLA might also be associated.		X			X	X	X
State: Mainly, describes if a service is active or inactive. Intermediate states might also exist.		X			X	X	X
Message in: parameters	X	X				X	
Message out: response	X	X				X	
Binding: BS to AS and BSE	**Service information**	**Registry information**	**Replicated information**	**Business analyst**	**Infrastructure architect**	**Solution architect**	**Enterprise architect**
Business-service name	X	X	X	X		X	X
Application-service and business service–extension names	X	X	X	X		X	X
Message in: parameters MAP (dependencies)		X		X		X	X
Message out: response MAP (dependencies)		X		X		X	X

- Software architects can refine orchestrations that are depicted in the initial blueprints that are made by business analysts. Subsequently, they can agree with enterprise and infrastructure architects on service versions and infrastructure support. Again, information at the ESR was central during the agreement.
- Information about service dependencies helped infrastructure architects to analyze the impact of binding new consumers to application services. This is critical for maintaining SLAs.
- Runtime policy–enforcement configurations at the service registry allowed specialized treatment for different client-application requests that were associated with a single BS—for example, applying particular validations with regard to formatting, security, and parameterization.
- In the case of new requirements triggering modifications to existing services:
  - Often, extensions or modifications involved changes only at the BSE level.
  - If an AS or BSE was modified and new versions were deployed, the version of the associated BS remained unchanged. (Service versioning is discussed in more detail in the "Impact of Service Versioning on Service Registries" section.)
  - Only incompatible changes that modify the business functionality could trigger new BS versions (in general, a BS is designed with forward compatibility in mind).

Among the lessons that were learned from different projects, we can emphasize the following:

- Decoupling of a BS from an actual implementation (by using AS(es) and a BSE) is an effective way of keeping domain experts separated from technical information, which facilitates service discovery at the business level.

- BS discovery support is key to enable the reuse of services beyond a single solution or project—allowing their use across projects and at an enterprise-wide scale.
- In practice, when only an AS is presented to domain experts, it remains almost untouched; that is, it is rarely reused in further developments.
- Even when new requirements involved modifications to existing service solutions, the reuse of a BS has still been strong. This was facilitated by addressing the required modifications at the BSE level.
- During legacy-application migration, client applications kept consuming the same BS. Changes mostly occurred at the AS level. At the ESR, AS descriptions and service dependencies were updated. New projects could reuse a BS independently when a migration had occurred.

Open Issues in Industry and Academia

This section discusses a number of observations in industry and academia with regard to enhanced service descriptions, organization of service information in a service registry, and the role of such a registry to enhance the reuse of services.

Strategies for Organizing and Finding Services in Registries

If service information in an enterprise service registry is difficult to distinguish because of inadequate organization or ineffective search mechanisms, the value of that registry is reduced.

Services *categorization* can help to distinguish services and classify them according to one or more categories. UDDI registries support this through the tModel. The categorization schemas of UDDI refer to taxonomic classifications. *Taxonomies* organize concepts in a hierarchical structure; multiple taxonomies can apply to a single UDDI entity. Standard classification schemas are suggested, such as the United Nations Standard Products and Services Code (UNSPSC[9]); however, other standards or internally created taxonomies can also be used. The UDDI Inquiry API supports different forms of query, such as browse pattern, drill-down pattern, and invocation pattern. *Queries* can refer directly to services, as well as to *service categories*.

Similarly to a Web search engine, the browse pattern allows one to find registry elements by matching keywords. Although this mechanism automates part of a service search, the results are limited to the coding system's value set and direct value matching. Services whose description includes similar or related concepts, but different syntax, cannot be retrieved by using this approach. Also, during use of different categorization schemas, the management of overlapping categories can become expensive.[10] Taxonomy maintenance is an added load that must be considered during the implementation of a service registry. Classification schemas that are not updated can affect the quality of the discovery results.[11]

The semantic research community has proposed alternatives to enrich service descriptions semantically and enhance classification schemas in services registries. Basic taxonomies can be enriched or replaced by ontologies. Ontologies structure concepts within a domain and define their *meaning*. Axioms constrain possible interpretations of concepts and reasoning mechanisms that support inferences from existing data.

According to Küster *et al.*,[12] although semantic enrichment of services descriptions can improve service discovery, several issues still must be addressed, such as reducing the computational cost of reasoning, maintaining the ontologies, and refining search results to improve effectiveness.

Impact of Service Versioning on Service Registries

When a service has been implemented, changes can occur—from the implementation itself to parts of the service description in a service repository. Changes might aim to improve reuse:

- Implemented services that follow a *bottom-up* approach[13] often fulfill particular project requirements within a domain. When any of these services becomes a candidate for reuse in a different context, it usually requires modifications or extensions.
- Analogously, services that follow a *top-down* approach often must be changed (specialized) to fit in particular contexts.

Different versioning strategies address different requirements. A single solution is not likely to be satisfactory for all situations. WSDL and UDDI do not define guidelines for versioning services. Some authors have proposed strategies for service versioning; most of them relate to *backward* and *forward* compatibility:[14]

- A *backward-compatible* version refers to the ability to support consumers of older versions of a service.
- A *forward-compatible* version refers to the ability to adapt to unknown future requests that are intended for newer versions of the service. This type of compatibility involves not only a service-*versioning* strategy, but also a service-*design* strategy that is related to *changeability*.

Often, new service versions are replications of a previous version that have additional or modified elements. New versions are named differently (by using some naming convention), and their description is stored in the registry as a new entry. Juric *et al.*[15] propose extensions to WSDL and UDDI for service versioning. The approach addresses run-time and development-time versioning. Efficiency at the code level is addressed by allowing multiple versions of a service to refer to the same codebase. Additionally, notifications about new and deprecated versions are communicated to consumers. Traceability support is provided to track changes. This academic research promotes the reuse of services and keeps the complexity of a service registry manageable.

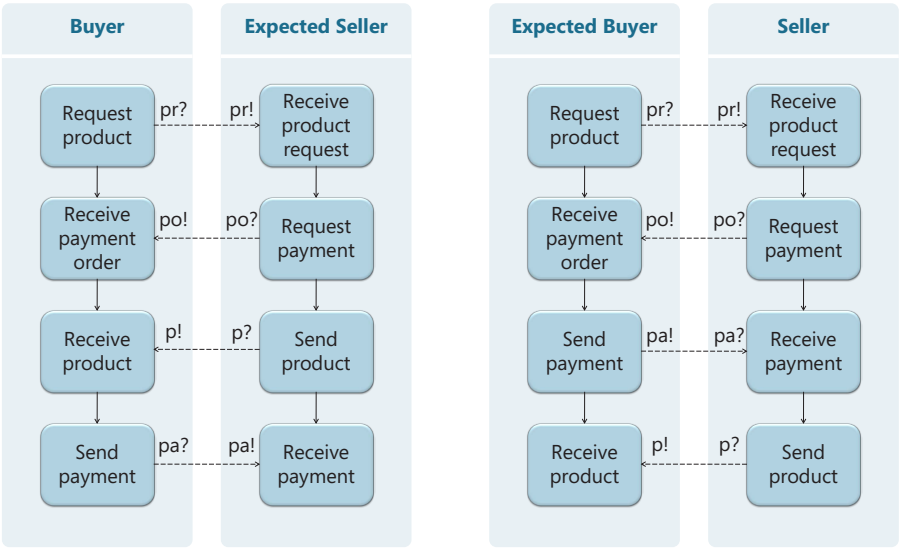
Service-Usage Information for Enhancing Service Description and Discovery

The history of service usage can be an interesting source of information—not only to re-create the actual behavior,[16] but also for service discovery. Stored service usage–history (logs) can help to categorize services according to the user or how services have behaved over time. Let us consider a service description that indicates a specific performance level in its contract; however, the actual measured performance in a given timeframe (extracted from logs) is lower. This information could be used during service discovery; a service that had lower-than-expected performance levels would be discarded from the search.

Statistically extracted information about how services behave against historic interactions can help to build less biased rankings and make service discovery more precise; however, an infrastructure for the constant monitoring of services and storing of the history information must be provided. Based on the service history, probabilities can be assigned to quantify uncertainty. Clark *et al.*[17] consider uncertainty with regard to the configuration of a service-based system, the rate parameters of system components, and the duration of events. An uncertainty model is used to predict system performance under increased demand. This type of analysis is fundamental when one is dimensioning the service support infrastructure. Historical data about individual services helps to predict the performance of an entire system.

Offer and demand in an inter-organizational scenario are subject to how much parties trust one another. "Trust in others" is one of several criteria for assigning reputation—witness reputation[18]—

Figure 5: Example of incompatibility at protocol level between two parties



to publicly available services. If company X knows that a service is being used or was positively rated for company Y, whom X trusts, the reputation of that service would increase from the point of view of X. One associated problem is the eventual bias for positive ratings, unfair ratings, and the variations of quality between ratings.[19]

Sufficiency of WSDL Descriptions to Find Services for Composition Efficiently

Services are reused not only by client applications, but also by other services in a service composition. A service composition can provide a more coarse-grained functionality and be closer to a business need. One problem when finding a service (useful in a service composition) is the need to verify if the services that are involved are able to "talk" to one another—that is, if the associated message-interchange protocol among them is *compatible*. A basic requirement for compatibility is deadlock-freeness. Moreover, the message syntax and semantics should be compatible.

Figure 5 illustrates a typical example of incompatibility at the protocol level between two parties. In the figure, a *Buyer* party offers a *buyProduct* service, and a *Seller* party offers a *sellProduct* service. To automate a hypothetical sale process, the message-interchange protocol between *buyProduct* and *sellProduct* should be compatible. However, Figure 5 illustrates that the Seller expects a payment before sending the product, and the Buyer expects the product before sending the payment.

When more and more services are offered and advertised in repositories, there are more chances of satisfying a service demand by composing existing services. However, mediation at the protocol level might be required. Matchmaking conflicts at the message and/or conversation level(s) can be solved—to a certain degree—by a mediator component.[20, 21] However, verifying and solving compatibility among services at the behavioral level is expensive; it involves the (expensive) exploration of possible states of the services during interaction. To increase reuse here, we need efficient mechanisms for finding compatible services.

For instance, instead of directly publishing the behavior of a service in a repository, a provider can publish a "summarized description" of the expected behavior of all compatible services to service (compatibility refers to deadlock-freeness). The "summarized description" is called an "operating guideline[22]"and allows the hiding of implementation details, while exposing enough information to find compatible partners. Checking if a service can be composed with others is reduced to checking if a graph-based representation of the potential partner is a subgraph of the "operating guideline," which is less expensive than exploring all possible states of the services.

Conclusion

To improve the reuse of services at the enterprise level, architects must define a strategy for publishing and providing facilities to access services information. For this purpose, an enterprise service registry is a key piece. Information about services can be organized at the registry, and basic functionality can be enhanced—including, for instance, functionality for service versioning, management of service dependencies, and enforcement of runtime policy. In this article, we have provided some design guidelines for enhancing an enterprise service registry to improve the reuse of enterprise services. We have also discussed some open issues in industry and academia with regard to the design and implementation of service registries and associated aspects that are required to describe and organize services information.

Resources

1 Pijanowski, Keith. "Visibility and Control in a Service-Oriented Architecture." MSDN, May 2007.

2 W3C.org. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.

3 UDDI.org. UDDI Version 3.0.2. *UDDI Spec Technical Committee Draft*, October 19, 2004.

4 Microsoft Corporation. "UDDI Specification Index Page." MSDN, July 2009.

5 Al-Masri, Eyhab, and Qusay H. Mahmoud. "Investigating Web Services on the World Wide Web." *WWW 2008: Web Engineering—Web Service Deployment Track*, April 21–25, 2008, 795–804.

6 Madrid, Chris. "SOA Realization Through Service Virtualization." *SOA Magazine*, September 3, 2007. (Available also here.)

7 Hohpe, Gregor, and Bobby Woolf. *Enterprise Integration Patterns*. Boston: Addison-Wesley, 2004.

8 Cardinal, Mario. "The Hidden Roles of Software Architects: The Three Types of Architect." MSDN, March 2008.

9 UNSPSC.org. "UNSPSC Codeset." (Managed by the Electronic Commerce Code Management Association (ECCMA).) August 2007.

10 Klein, Michel. "Combining and Relating Ontologies: An Analysis of Problems and Solutions." Proceedings of Workshop on Ontologies and Information Sharing at 17th International Joint Conference on Artificial Intelligence (IJCAI-01), 2001, 53–62.

11 Hepp, Martin, Joerg Leukel, and Volker Schmitz. "A Quantitative Analysis of eClass, UNSPSC, eOTD, and RNTD: Content, Coverage, and Maintenance." Proceedings of IEEE International Conference on e-Business Engineering (ICEBE'05), 2005, 572–581.

12 Küster, Ulrich, Holger Lausen, and Birgitta König-Ries. "Evaluation of Semantic Service Discovery: A Survey and Directions for Future Research." *Emerging Web Services Technology*. Volume II, 2008, 41–58.

[13] Erl, Thomas. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall Service-Oriented Computing Series from Thomas Erl, 2004.
[14] Parrish, Allen, Brandon Dixon, and David Cordes. "A Conceptual Foundation for Component-Based Software Deployment." *Journal of Systems and Software*. Vol. 57, Issue 3, July 2001, 193–200.
[15] Juric, Matjaz B., Ana Sasa, Bostjan Brumen, and Ivan Rozman. "WSDL and UDDI Extensions for Version Support in Web Services." *Journal of Systems and Software*. Vol. 82, Issue 8 (doi:10.1016/j.jss.2009.03.001), August 2009, 1326–1343.
[16] van der Aalst, W. M. P., and A. J. M. M Weijters. "Process Mining: A Research Agenda." *Computers in Industry*. Vol. 53, Issue 3 (doi: 10.1016/j.compind.2003.10.001), April 2004, 231–244.
[17] Clark, Allan, Stephen Gilmore, and Mirco Tribastone. "Quantitative Analysis of Web Services Using SRMC." *SFM*. Vol. 5569, 2009, 296–339.
[18] Huynh, Trung Dong, Nicholas R. Jennings, and Nigel R. Shadbolt. "An Integrated Trust and Reputation Model for Open Multi-Agent Systems." *Journal of Autonomous Agents and Multi-Agent Systems*. Vol. 13, Issue 2, March 2006, 119–154.
[19] Jøsang, Audun, Roslan Ismail, and Colin Boyd. "A Survey of Trust and Reputation Systems for Online Service Provision." *Decision Support Systems*. Vol. 43, Issue 2, March 2007, 618–644.
[20] Dustdar, Schahram, and Wolfgang Schreiner. "A Survey on Web Services Composition." *International Journal of Web and Grid Services (IJWGS)*. Vol. 1, Issue 1, 2005, 1–30.
[21] Li, Xitong, Yushun Fan, Jian Wang, Li Wang, and Feng Jiang. "A Pattern-Based Approach to Development of Service Mediators for Protocol Mediation." Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), February 2008, 137–146.
[22] Kaschner, Kathrin, Peter Massuthe, and Karsten Wolf. "Symbolic Representation of Operating Guidelines for Services." *Petri Net Newsletter*. Vol. 72, April 2007, 21–28.

About the Authors

Juan Pablo García-González is a senior software developer and chief architect at DATCO Chile S.A. He has been involved in numerous software projects involving service-based solutions. In addition to their regular projects, he and his team are working on the creation of innovative service-centric mobile solutions for the banking industry.

Veronica Gacitua-Decar is a postgraduate researcher at the School of Computing, Dublin City University, and Lero - the Irish Software Engineering Research Centre. Her research is focused on the development of tools and methods for designing process-centric service architectures. Previously, she worked as an IT architect in a large mining company.

Dr. Claus Pahl is a senior lecturer at the School of Computing, Dublin City University, where he is leader of the Software and Systems Engineering group. He is also involved in Lero - the Irish Software Engineering Research Centre, as well as the Centre for Next Generation Location (CNGL).

A Simple, Successful EA/SOA Model

by Tom Hansen and Michael Andersen

When it comes to developing an EA, there are several acknowledged frameworks out there from which you can choose to aid you in your work. Some of the best known are the Zachman Framework and The Open Group Architecture Framework (TOGAF). These frameworks are excellent; when they are used correctly, they give good results. However, they are also very complex and require a lot of experience.

Many enterprises are not ready to start with such complex frameworks, no matter how good the framework.

Getting Backing from the Business

To get backing from the organization, an EA/SOA project must be kept simple—or, at least, be explained in a simple way. A principle that we follow at Toyota is that you must be able to explain any project in just one page. If you are not able to do that, the project will not be funded.

It is hard to make Zachman/TOGAF look simple. But as Albert Einstein once put it, "If you can't explain it simply, you don't understand it well enough."

A good way to keep explanations of complex SOA projects simple is to limit the primary focus to:

- An easy-to-understand model for describing business processes.
- A simple model for describing the service architecture.
- An expandable technical infrastructure that will support the SOA.

Extending the EA and SOA Infrastructure Iteratively

A key point for success with keeping SOA projects simple is to build the architecture and the infrastructure in an iterative way. The architecture should evolve together with the projects, instead of being the result of several months (or years) of work during which all other projects are put on hold.

The business people must understand what is going on, all of the way. If the business does not understand the EA/SOA model, it is because you are moving too fast or making things too complex. So, slow down!

A Successful Approach

We have used this approach with several customers who have had little or no prior experience in developing enterprise architectures. One of these customers is Toyota Denmark, where the approach has provided the following results (among others):

- Involvement of the businesspeople already has ensured a high level of reuse of messages from the second project, after the implementation of the initial enterprise-architecture model.
- Having seen the immediate reuse of messages, the businesspeople display further engagement and enthusiasm.
- A new project model is developed with consistent mapping, which gives shorter and more predictable implementation of projects and better control of IT projects (as well as the vendors).

Please stop by our Web site for all of the details.

Tom Hansen works with technical and organizational aspects of SOA and EA. He represents Globeteam in the SOA Partner Expert Team, which is hosted by Microsoft Denmark.

Michael Andersen is the manager of the Architecture and Development team at Globeteam. He has delivered presentations at conferences that have been hosted by Microsoft and Computerworld.

How the Cloud Stretches the SOA Scope

by Lakshmanan G and Manish Pande

Summary

In this article, we will describe how the enterprise service-oriented-architecture (SOA) scope is stretched with the advent of cloud computing. With the help of the case study of a fictitious global retailer, we will demonstrate the process for identifying cloud scenarios. Also, we will come across an emerging breed of distributed applications—both on-premise and in the Cloud—and discuss the integration considerations for building them.

Introduction

Over the past few years, enterprises have adopted strategies for both build and build vs. buy for their enterprise-application development. In many business domains, such as retail and manufacturing, packaged application has found favor because of the availability of best-practice business functionality for faster time to market. This has led to a mesh of packaged applications, legacy systems, and bespoke and homegrown line-of-business (LOB) applications that are connected by point-to-point integration and mature enterprise-application integration (EAI) and SOA-based enterprise service bus (ESB) implementations, in some cases.

Enterprises are now extending the existing enterprise SOA investments into the Cloud to address new business pressures and opportunities.

In this article, we use the example of a fictitious global retailer (Globtron) to demonstrate how an enterprise should explore the new business opportunities and map application requirements to the Cloud.

Also, with the help of this case study, we describe the integration requirements and solution options for on-premise and cloud integration in the context of Microsoft Windows Azure Platform .NET Services.

SOA in an Enterprise

As an architectural principle for delivering agility and flexibility by decoupling interface and implementation (business logic), SOA has been in existence for decades. Object-oriented and component-based development models were established by using the same principle. Recent technological developments and Web-service standards have made SOA easier by replacing method calls on object references to message passing.

In large enterprises, services have been delivered in isolation by several projects that are based on LOB requirements in the organization—not as a big-bang SOA-definition project. Enterprise-architecture (EA) efforts in the organizations helped identify the reusable services and leverage existing investments. Because SOA facilitated interoperability, many legacy applications were exposed as services and the business functionality was reused.

SOA became the EAI driver across the company, and EAs have evolved into on-premise distributed systems that are enabled by SOA.

The Cloud Opportunity

Cloud computing presents both a model and new opportunities for technology buyers in an enterprise. It offers the opportunity to focus on the core capabilities of an enterprise by outsourcing certain aspects of IT and reducing IT costs. It also accelerates the provisioning and deployment with the support hassles that are transferred to the cloud-service provider. The Cloud is altering the way in which organizations build their infrastructure and applications.

The essential characteristics of cloud computing include both the delivery model and the commercial model. In terms of the delivery model, it should be available as a service over the Internet and accessible either from a Web browser or as a Web service. Commercially, users pay for service usage; both the overall maintenance effort and user costs are low.

Technologically, the Cloud is a culmination of standards and technologies that have come together over the past several years. They include server virtualization, Web Security, and Web Services.

The implication for the enterprise is that dynamically scalable infrastructure is available transparently, without any IT involvement in building and managing the infrastructure.

The popular cloud-computing models include Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), as described in the following:

- **Infrastructure as a Service**—Delivery of server infrastructure as a service. An enterprise does not need to purchase servers, networking equipment, or data-center real estate. It is all managed by the cloud provider who also dynamically scales up or down, based on the application-workload requirements. Examples of IaaS are Amazon Elastic Compute Cloud (EC2) and AppNexus.
- **Platform as a Service**—Infrastructure and complete development environment and computing platform for application development and deployment in the Cloud. Examples of PaaS are the Windows Azure Platform and the Google App Engine.
- **Software as a Service**—Web-based deployment model, so that the entire application software is available through the Web browser as a service. Again, pricing is based on usage. Examples of SaaS are SalesForce.com, Windows Live Hotmail, and Microsoft Dynamics CRM Online.

Application Characteristics for the Cloud in an Enterprise

The cloud adoption in enterprises initially is led by tactical opportunities that offer implementation speed and cost advantages to these enterprises, although there are a few misconceptions about when to use the Cloud and the associated technological challenges in moving toward it. Security, performance, availability, and reliability are the top concerns that IT managers cite for adopting the Cloud. Cloud providers are addressing these concerns through appropriate architectural measures and service-level agreements (SLAs) to gain user confidence in the Cloud. (See Figure 1.)

The following are the characteristics of applications that are potential candidates for the Cloud:

1. **Non-core business services**—For an enterprise, there are certain business functions that are core to the value that the enterprise delivers and drive more business and customers to the organization. For example, the core functions for a retailer are merchandizing and customer service. However, functions such as Web conferencing, enterprise-content management, and portal are non-core or non-differentiators; they are only enablers to support the business. Non-core applications are good candidates for the Cloud and allow organizations to focus their business and IT staff on building the core.
2. **High computing workloads over short time span**—Applications that do not have uniform workload requirements. There will be spikes in usage for a small period; otherwise, however, resource utilization will be low most of time. Instead of acquiring infrastructure to support the peak workload, an enterprise can acquire space in the Cloud and pay for usage.
3. **New business services and pilots**—Applicable in scenarios in which enterprises need the flexibility to launch new business applications, without having to invest in full IT infrastructure upfront. The Cloud offers a good platform for experimentation, as businesses can terminate the contract, if needed—thus, helping them manage their risks better.
4. **Web 2.0 collaborative applications**—Applications (such as video sharing, discussion forums, and blogs) that generate data that is exposed to the public.
5. **Centralized applications**—Applications that have cross-enterprise and cross-departmental reach. Instead of duplicating the effort in creating multiple applications or copies of the same software, these can be offered as cloud applications. Besides allowing enterprises to offload the infrastructure management to the cloud provider, they also offer economics of scale through their reuse across different departments.

The Case of a Global Retail Enterprise

Model Business Drivers

How do cloud opportunities for enterprise IT derive from the business pressures and enterprise requirements? Let us take the example of Globtron, a fictitious retailer of electronics goods. Globtron is a U.S.-based retailer that has a strong brand presence. Recently, it has increased its global footprint with rollouts in a few European countries through acquisitions. Over several decades of running the business, Globtron management knows that its core business value that drives customers to their stores is in merchandising, customer service, and branding.

The key business challenges for Globtron are to expand rapidly and extend its footprint to emerging markets in Eastern Europe and Asia, establish its brand worldwide via superior customer service, and drive growth further through new business services and channels.

- **Rapid global expansion**—Although Globtron is growing both organically and via acquisitions, its objective is to adopt its existing business best practices and IT systems, while taking care of local requirements and constraints. Globtron wants to leverage its existing investments in IT and reduce licensing, hosting, and maintenance costs to benefit from what Globtron believes will offer benefits of economics of scale. The challenge is to deploy existing staff in the United States and local IT to focus on the core business and build innovative applications for merchandising and customer service, instead of managing servers and infrastructure.
- **Branding and superior customer service**—Customer feedback and collaboration are key to the success of Globtron, who wants to offer collaborative applications and business tools to customers and store staff, so as to connect better with customers and improve customer service.
- **Driving growth via new business services and channels**—Besides opening new stores across several countries, Globtron also wants to increase its existing service and channels mix to improve customer experience. Again, Globtron wants to try out the new services and channels quickly and review initial progress through pilot launches, without having to commit to extensive investments in IT infrastructure.

Application Scenarios for Each Business Driver

Next, we list the application requirements to address the key business issues that are linked to the key business drivers. We are limiting it to the sales and marketing subdomain of the retail enterprise. Please note that this is only an illustration for discussing the concept and approach, and is not a comprehensive listing of all possible scenarios and solutions. (See Figure 2.)

Figure 1: Application characteristics for the Cloud in an enterprise

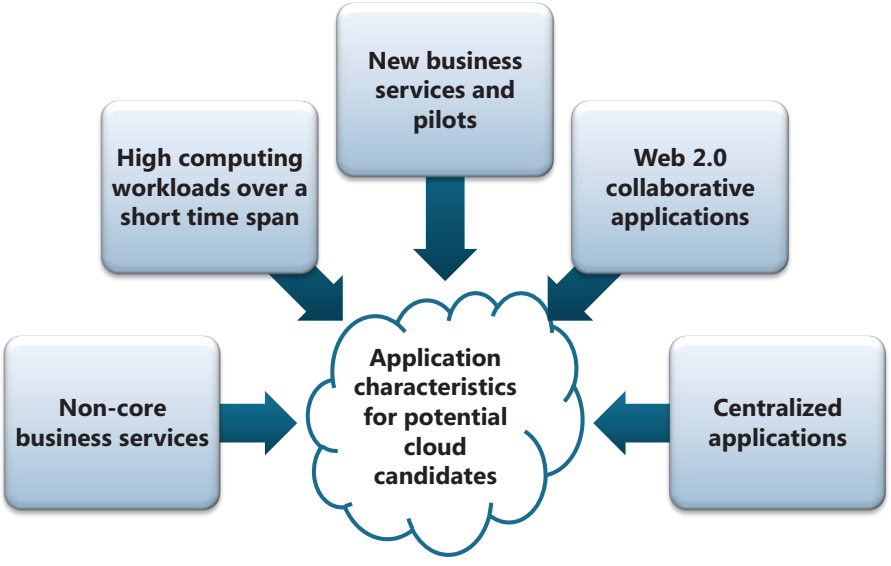
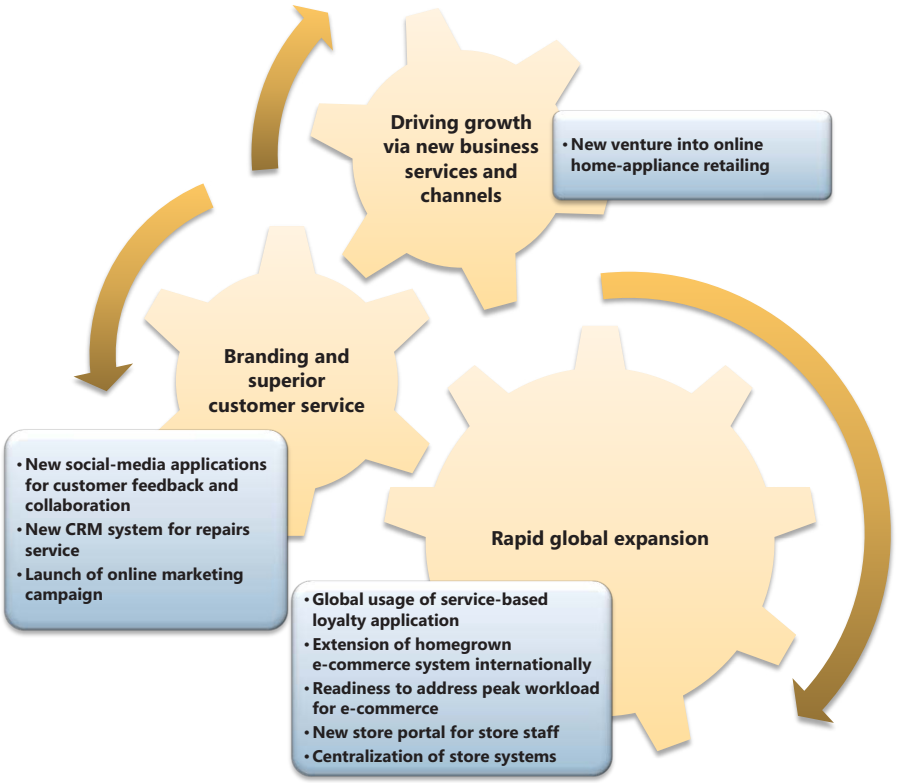


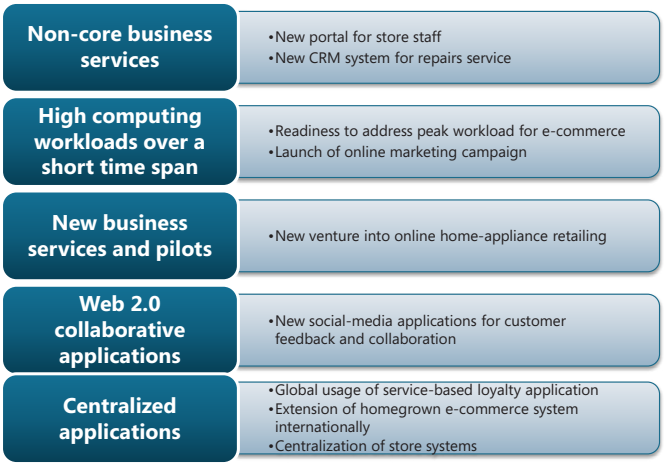
Figure 2: Case of a global retail enterprise—application scenarios for each business driver



Rapid Global Expansion

- **Global usage of service-based loyalty application**—Globtron has built a loyalty application in the United States that exposes Web Services for consumption from multiple channels, including store and e-commerce systems. The services allow store and e-commerce systems to validate the loyalty card number of a customer, compute loyalty points, and set up the customer in the loyalty system. The Globtron business team believes that it should leverage this system globally for all other countries.
- **Extension of homegrown e-commerce system internationally**—Globtron has established a successful online e-commerce and is able to drive huge traffic and sales in the United States. Globtron wants to utilize the existing e-commerce application for rollouts across the world, but without the hassles and costs to set up multiple instances for similar functionality across different countries.
- **Readiness to address peak workload for e-commerce**—Globtron has asked its IT department to assess the readiness of its infrastructure to address peak workload demand for the upcoming holiday season.
- **New store portal for store staff**—Globtron also wants a store portal for the store staff, so that the retailer can improve communication and collaboration with the store staff and share information, such as operation manuals and best practices.
- **Centralization of store systems**—One of the pain points for retail IT is the overhead and cost that are involved in distributing software upgrades and patches across stores. Globtron wants to re-architect and deploy a central store system that is accessible from the store devices via the Internet and the smart-client model.

Figure 3: Case of a global retail enterprise—mapping application scenarios to cloud-application characteristics



Branding and Superior Customer Service

- **New social-media applications for customer feedback and collaboration**—Globtron wants to engage better with its customers, obtain feedback on its product and services, and drive sales and customer loyalty through collaborative applications such as reviews and ratings, videos sharing, and discussion forums.
- **New CRM system for repairs service**—Globtron wants to improve the customer service for its electronics-repairs business that is available in several stores through automation.
- **Launch of online marketing campaign**—Although Globtron does not have a significance presence in online selling outside the United States, it wants to launch an online marketing campaign to provide offers for its customers, build loyalty, and drive more sales.

Driving Growth via New Business Services and Channels

- **New venture into online home-appliance retailing**—As part of its strategy to venture into new services, Globtron decided to retail large home appliances in Germany. However, because its store format is small and medium, Globtron does not have much real estate for this business. The business team decided to experiment with online selling for this segment. However, the success of this business will require some initial pilots and feedback before it can go full steam into it.

Mapping Application Scenarios to Cloud-Application Characteristics

We have mapped each of the previous application scenarios to the cloud-application characteristics that we discussed earlier. Figure 3 illustrates the mapping.

Mapping Application Scenarios to Cloud Types

With an understanding of the application characteristics and mapping them to cloud characteristics, we can now map each application to one of the cloud types. The following are some of the cloud-solution options for Globtron (see Figure 4):

- **New portal for store staff and CRM system for repairs service**—Leverages SaaS offerings such as Microsoft Office SharePoint Online and Dynamics CRM Online.
- **Readiness to address peak workload for e-commerce**—Cloud burst to an IaaS service such as Amazon EC2
- **Extension of homegrown e-commerce system internationally**—Can again deploy it on the Cloud by using an IaaS offering
- **Global usage of service-based loyalty application**—Exposes on-premise loyalty Web Services to all other countries by using integration in cloud offerings such as Windows Azure Platform .NET Services
- **Centralization of store systems**—Re-architects and deploys the existing application on the Windows Azure Platform

The remaining three could be developed by using a PaaS offering, such as the Windows Azure Platform and the Google App Engine:

- Launch of online marketing campaign
- New venture into online home-appliance retailing
- New social-media applications for customer feedback and collaboration

Extending On-Premise Applications to the Cloud

For large enterprises, the different applications and associated data cannot exist in silos, whether on-premise or in the Cloud. They must integrate for exchange of data, so as to complete the different steps of larger business process or leverage the reusable business services that are exposed by each other in delivering superior business applications.

The cloud model offers additional choice with management simplicity to enterprises, so as to augment their core on-premise applications. The EA challenge is to integrate with the Cloud. The good news is that with adoption of existing SOA and Web Services, this will be easier than ever. The Cloud should be viewed as an extension of the existing EA.

In the following section, we discuss the integration scenarios for the Cloud and the considerations for extending existing SOA investments into the Cloud, along with some examples from the Globtron case study.

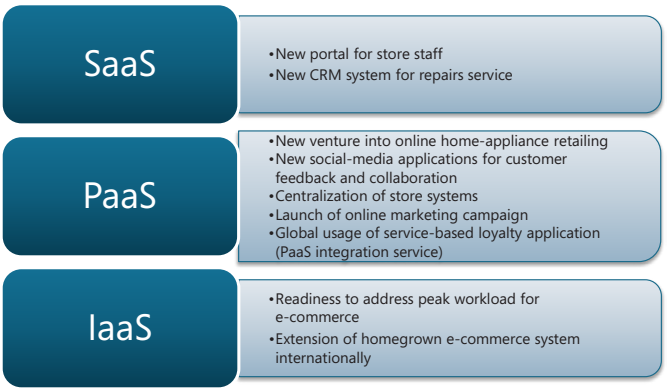
Possible Integration Scenarios

The possible integration scenarios between on-premise and Cloud-based applications are as follows:

- Browser access only: Usually, SaaS applications are full-service standalone applications that are accessible via a Web browser.
- Cloud functionality that is exposed as a service for consumption by on-premise or other cloud applications.
- Existing on-premise business functionality or services that are exposed to the Cloud.
- Data integration between on-premise and cloud data stores.

Such emerging hybrid models of on-premise and cloud applications result in a new class of distributed applications. SOA-based integration with SOAP or REST Web Services seems to be the easy answer to this problem; however, there are various other issues that should be handled for it to work across organizational boundaries and firewalls.

Figure 4: Case of a global retail enterprise—mapping application scenarios to cloud types



Considerations for Cloud Integration

- **Existing SOA investments**—There is commonality in the business objectives of SOA and cloud computing with regard to delivering agility, flexibility, and reuse. The existing core business services that have been developed within the enterprises will coexist and integrate with the services in the Cloud. This is again enabled by SOA. The integration solution should address the challenge in letting outside organizations find endpoints of the on-premise services.
- **Enterprise-architecture practices**—SOA is part of the EA mainstream. The scope of existing organization practices for EA, such as governance, are preserved and extended to the Cloud (the Cloud is also part of the enterprise technology portfolio).
- **Access across firewalls**—To expose existing on-premise services to the Cloud, enterprise IT has to deal with issues such as handling network-address translation and enabling access through a firewall by opening new ports.
- **Security-access control**—In the distributed application, we need an access-control solution that works across enterprises that holding their own identity information. The solution should address key quality-of-service (QoS) requirements, such as system availability and reliability of integration points.

Extension of On-Premise Applications to the Cloud by Using Windows Azure Platform .NET Services

The Windows Azure Platform is an Internet-scale, cloud-computing services platform that is hosted in Microsoft data centers. The Windows Azure Platform uses SOA heavily and provides SOA-based access to data.

.NET Services is one of the components of the Windows Azure Platform; it provides cloud-based infrastructure services and facilitates creation and deployment of composite applications. .NET Services comprises an access-control service and a service bus.

- **Access control**—Cloud implementation of claims-based identity federation for performing federated access-control authorization as a Web service across organizations and protocols. This service utilizes standard protocols, such as WS-Trust and WS-Federation.
- **Service bus**—Allows applications that expose Web-service endpoints to be located and accessed by clients. This service enables communication between two applications—both of which might run in the Cloud; one of which might run in the Cloud and the other on-premise; or both of which might run on-premise,

Table 1: Cloud integration for the "New venture into online home-appliance retailing" scenario

Integration requirement(s)	Applicable integration type	Integration pattern	Windows Azure solution
Expose existing on-premise loyalty, order management, and fulfillment services	Functional integration	Service interface	.NET Services
Periodic import of catalogs from on-premise merchandising systems	Data integration	Message-oriented middleware	Queue in Windows Azure computing resources
Customer data to be replicated between the new online database and on-premise customer repositories	Data integration	Data replication	Replication capability of cloud data platforms, such as SQL Azure
Inventory updates from the existing fulfillment and warehousing systems of Globtron	Data integration	Message-oriented middleware	Queue in Windows Azure computing resources

Table 2: Cloud integration for the "Global usage of service-based loyalty application" scenario

Integration requirement	Applicable integration type	Integration pattern	Windows Azure solution
Existing on-premise loyalty service	Functional integration	Service interface	.NET Services

Table 3: Cloud integration for the "New social-media applications for customer feedback and collaboration" scenario

Integration requirement	Applicable integration type	Integration pattern	Windows Azure solution
Aggregate user-generated content such as blogs, reviews and ratings, and videos with the product data in the marketing site to generate interesting product reviews and information	Data integration	Mash-up	Windows Azure computing and storage resources

but across different enterprise data centers. It also addresses the firewall-access issues of enterprise systems and enables connection, without requiring data centers to open new ports for access—thus, providing location transparency to integrating applications.

Refer to the Resources section for links to the Windows Azure resources that are available on the Internet.

Integration Scenarios for Globtron

Let us consider the integration requirements of some of the applications for Globtron, so as to illustrate the on-premise and cloud-integration requirements and possible solution patterns. We also look at integration-solution options in the context of the Windows Azure Platform.

Example 1: *New venture into online home-appliance retailing*

Table 1 shows the integration requirements for this application that Globtron is building by using Windows Azure computing and storage services.

Example 2: *Global usage of service-based loyalty application*

To expose the on-premise loyalty service, Globtron can utilize integration services—such as .NET Services—on the Cloud. This allows existing enterprises to expose their Web-service endpoints for usage from the Cloud and partner enterprises (see Table 2).

Example 3: *New social-media applications for customer feedback and collaboration*

The social-media application that is built on Windows Azure can expose content by using RSS feeds or Web Services (RESTful or SOAP). Globtron can create mash-ups by combining these with the existing data services or data source of the on-premise marketing application (see Table 3).

Conclusion

In this article, we discussed how organizations have evolved EAs into on-premise distributed systems that are enabled by SOA and can identify cloud opportunities, so as to address their new business requirements. We discussed the application characteristics for the Cloud. We also discussed how an enterprise can map its business drivers and application scenarios to cloud characteristics and identify cloud solutions that coexist and integrate with the existing on-premise applications. Finally, we discussed integration scenarios, requirements, and solution considerations.

Resources

Armbrust, Michael, et al. "Above the Clouds: A Berkeley View of Cloud Computing." *Technical Report No. UCB/EECS-2009-28*, Electrical Engineering and Computer Services, University of California at Berkeley, February 10, 2008.

Bertocci, Vittorio. "Claims and Identity: On-Premise and Cloud Solutions." *The Architecture Journal*, MSDN, July 2008.

Carr, Nicholas G. *The Big Switch: Rewiring the World, from Edison to Google.* New York: W. W. Norton & Co., 2008.

Chappell, David. "Introducing the Azure Services Platform: An Early Look at Windows Azure, .NET Services, SQL Services, and Live Services." MSDN, October 2008.

Hoover, J. Nicholas, Paul McDougall, Art Wittmann, Rob Preston, and John Foley. "Microsoft's Stake in the Cloud." Research and report, *InformationWeek*, October 31, 2008.

Microsoft Corporation. "Architecture Journal Profile: Ray Ozzie." *The Architecture Journal*, MSDN, October 2007.

Microsoft Corporation. "Infosys Creates Cloud-Based Solution for Auto Dealers Using SQL Data Services." Microsoft Case Studies, October 2008.

Pace, Eugenio, and Gianpaolo Carraro. "Head in the Cloud, Feet on the Ground." *The Architecture Journal*, MSDN, October 2008.

Trowbridge, David, et al. "Integration Patterns." patterns & practices Developer Center, MSDN, June 2004.

About the Authors

Lakshmanan G (lakshmag@infosys.com) is an educator and mentor in IT architecture with the Education and Research group of Infosys Technologies Ltd. With more than 20 years of experience in the IT industry, he nurtures a vibrant community of architects at Infosys. Prior to this, he was Head of Engineering, SaaS Practice, and Head of Infosys EMEA Architecture Practice.

Manish Pande (manishp@infosys.com) is a lead architect with the Product Incubation and Engineering (PIE) unit of Infosys Technologies Ltd. Currently, he is responsible for architecting Infosys SaaS–based product offerings. With 11 years of consulting experience, he has played a variety of roles, including Solution Architect, Engineering Manager, and Technical and Performance Architect.

Follow up on this topic

- Applications in the cloud with Windows Azure Platform

Calculating the Economic Viability of Cloud Computing for Your Organization

by Shy Cohen

There are many aspects to consider when you plan your cloud-computing strategy, such as the architectural fit of your application for the cloud model, the existing knowledge and training needs for your development and IT staff, and the protection of intellectual property. In this column, I would like to touch on just one aspect: the economic angle.

When you plan cloud adoption, start by looking closely at the cost of running your application on the various cloud platforms. Compare this cost to the cost of aggressive virtualization or of running your own "Internal Cloud." Examine the physical computer and/or virtual machine and process-level models for running applications off-premises.

Measuring is key! Cloud computing is a nascent technology, so you should not blindly trust the numbers that other people come up with. The best way to assess the actual cost of running your applications in these different environments is to test them yourself. Remember that doing so might require some re-architecting of your application (to optimize it for the platform on which you are measuring cost) and that measuring takes time and money.

When you have measured the actual cost of running your applications, expand your calculation to consider more than "where is it cheaper to run my app?" Think about your ROI and ROA. Consider the facilities, employees, and management costs that are associated with running your own hardware. Evaluate the possible gains (monetary and other) of being able to capitalize quickly on emerging opportunities through avoiding the delay that can be introduced by IT procurement. Think holistically.

It is likely that different departments within your organization will employ different cloud strategies. For some, adopting a cloud platform might be better, while the right choice for others might be to invest in building up a departmental computer lab, as well as hiring and managing the personnel to run and manage it. The decision will be based on the focus of the department and its available IT resources.

Another factor is the duration for which the resources are needed. Short-term usage and sporadic usage might require a strategy that is different from the one that is used for long-term investments. Processing and/or storage capacity are no longer assets that you must buy and manage, especially if you do not need them most of the time. Consider the ongoing cost of running and maintaining computer resources, given their actual utilization, and determine their real cost and associated ROA.

Lastly, remember that cost must be measured on an ongoing basis. The cost structure of on-demand resource utilization is different from the cost structure of on-premises computing. Protect yourself by continually monitoring and managing your resource utilization, as well as the charges that this consumption creates. Whichever approach you choose, prepare a contingency plan in case the cost of that approach becomes too high.

Shy Cohen (shy@shycohen.com) is an independent Software Architect, Consultant, Speaker, Coach, and Entrepreneur. Before he started his own practice, Shy worked at Microsoft for many years as Senior Program Manager on projects such as "Oslo," WCF, MSN, Windows, and ISA.

Event-Driven Architecture: SOA Through the Looking Glass

by Udi Dahan

Summary

This article proposes that event-driven architecture (EDA) and service-oriented architecture (SOA) are really two sides of the same coin.

Introduction

While event-driven architecture (EDA) is a broadly known topic, both giving up ACID integrity guarantees and introducing eventual consistency make many architects uncomfortable. Yet it is exactly these properties that can direct architectural efforts toward identifying coarsely grained business-service boundaries—services that will result in true IT-business alignment.

ACID is an acronym for Atomicity, Consistency, Isolation, Durability: a set of properties that traditionally describe database transactions.

Business events create natural temporal boundaries across which there is no business expectation of immediate consistency or confirmation. When they are mapped to technical solutions, the loosely coupled business domains on either side of business events simply result in autonomous, loosely coupled services whose contracts explicitly reflect the inherent publish/subscribe nature of the business.

A *temporal boundary* is a boundary on the axis of time, where one side of the boundary exists in a time-frame disjoint from the other side.

This article will describe how all of these concepts fit together, as well as how they solve thorny issues such as high availability and fault tolerance.

Commands and Events

To understand the difference in nature between "classic" service-oriented architecture (SOA) and event-driven architecture (EDA), we must examine their building blocks: the command in SOA, and the event in EDA.

In the commonly used request/response communication pattern of service consumer to service provider in SOA, the request contains the action that the consumer wants to have performed (the command), and the response contains either the outcome of the action or some reaction to the expressed request, such as "action performed" and "not authorized."

Commands are often named in imperative, present-tense form—for example, "update customer" and "cancel order."

In EDA, the connection between event emitters and event consumers is reversed from the previously described SOA pattern. Consumers do not initiate communication in EDA; instead, they receive events that are produced by emitters. The communication is also inherently unidirectional; emitters do not depend on any response from consumers to continue performing their work.

Events are often named in passive, past-tense form—for example, "customer updated" and "order cancelled"—and can represent state changes *in the domain of the emitter.*

Events can be thought of as mirror images of the commands in a system. However, there might be cases in which the trigger for an event is not an explicit command, but something like a timeout.

Business Processes with Commands and Events

The difference between commands and events becomes even more pronounced as we look at each one as the building block in various business processes.

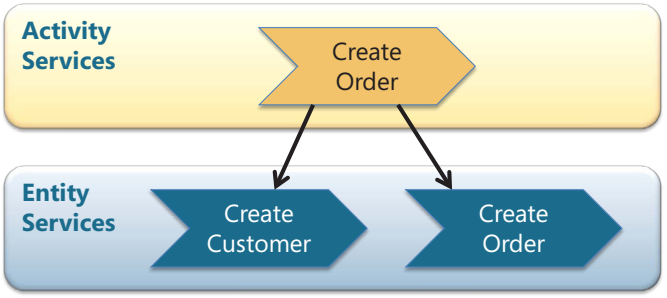
When we consider commands such as "create customer" and "create order," we can easily understand how these commands can be combined to create more involved scenarios, such as: "When creating an order, if a customer is not provided, create a new customer." This can be visualized as services that operate at different layers, as shown in Figure 1.

One can also understand the justification for having activity services perform all of their work transactionally, thus requiring one service to flow its transactional context into other lower-level services. This is especially important for commands that deal with the updating of data.

When working with commands, in each step of the business process, a higher-level service orchestrates the work of lower-level services.

When we try to translate this kind of orchestration behavior into events, we must consider the fact that events behave as mirror images of commands and represent our rules by using the past tense.

Figure 1: Commands and service orchestration



Instead of: "When creating an order, if a customer is not provided, create a new customer."

We have: "When an order *has been created*, if a customer was not provided, create a new customer."

It is clear that these rules are not equivalent. The first rule implies that an order should not be created unless a customer—whether provided or new—is associated with it. The second rule implies that an order can be created even if a customer has not been provided—stipulating the creation as a separate and additional activity.

To make use of EDA, it is becoming clear that we must think about our rules and processes in an event-driven way, as well as how that affects the way in which we structure and store our data.

Event-Driven Business Analysis and Database Design

When we analyze the "When an order has been created, if a customer was not provided, create a new customer" rule, we can see that a clear temporal boundary splits it up into two parts. In a system that has this rule, what we will see is that at a given point in time, an order might exist that does not have a corresponding customer. The rule also states the action that should be taken in such a scenario: the creation of a new customer. There might also be a nonfunctional requirement that states the maximum time that should be allowed for the action to be completed.

From a technical/database perspective, it might appear that we have allowed our data to get into an inconsistent state; however, that is only if we had modeled our database so that the Orders table had a non-nullable column that contained CustomerId—a foreign key to the Customers table. While such an entity-relationship design would be considered perfectly acceptable, we should consider how appropriate it really is, given the requirements of *business consistency*.

The rule itself indicates the business perspective of consistency; an order that has no connection to a customer is valid, for a certain period of time. Eventually, the business would like a customer to be affiliated with that order; however, the time frame around that can be strict (to a level of seconds) or quite lax (to a level of hours or days). It is also understandable that the business might want to change these time frames in cases in which it might provide a strategic advantage. An entity-relationship design that would reflect these realities would likely have a separate mapping table that connected Orders to Customers—leaving the Orders entity free of any constraint that relates to the Customers entity.

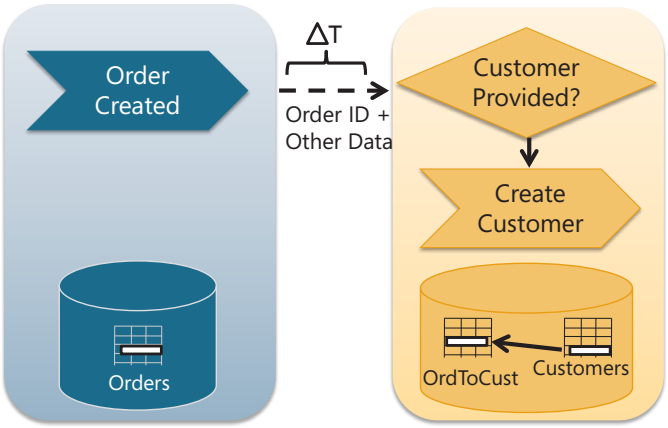
That is the important thing to understand about eventual consistency: It starts by identifying the business elements that do not have to be 100-percent, up-to-the-millisecond consistent, and then reflecting those relaxed constraints in the technical design.

In this case, we could even go so far as to have each of these transactions occur in its own database, as shown in Figure 2.

Benefits of Event-Driven Architecture

Given that EDA requires a rethinking of the core rules and processes of our business, the benefits of the approach must be quite substantial to make the effort worthwhile—and, indeed, they are. By looking at Figure 2, we can see very loose coupling between the two sides of the temporal boundary. Other than the structure of the event that passes from left to right, nothing is shared. Not only that, but after the event is published, the publisher no longer even needs to be online for the subscriber to process the event, so long as we use a durable transport (such as a queue).

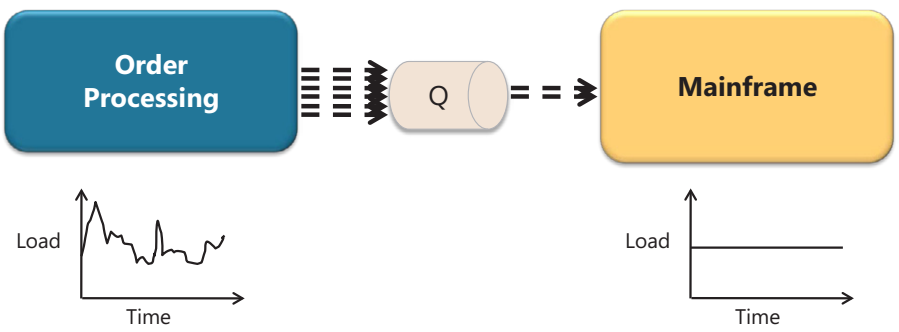
Figure 2: Event-driven data flows



These benefits become even more pronounced when we consider integration with other systems. Consider the case in which we want to integrate with a CRM, whether it is onsite or hosted in the cloud. In the EDA approach, if the CRM is unavailable (for whatever reason), the order will still be accepted. Contrasting this with the classic command-oriented service-composition approach, we would see there that the unavailability of the CRM would cause the entire transaction to time out and roll back. The same is true during integration of mainframes and other constrained resources: Even when they are online, they can process only N concurrent transactions (see Figure 3). Because the event publisher does not need to wait for confirmation from any subscriber, any transactions beyond those that are currently being processed by the mainframe wait patiently in the queue, without any adverse impact on the performance of order processing.

If all systems had to wait for confirmation from one another—as is common in the command-oriented approach—to bring one system to a level of 5 nines of availability, all of the systems that it calls would need to have the same level of availability (as would the systems that they call, recursively). While the investment in infrastructure might have business justification for one system (for example, order processing), it can be ruinous to have to multiply that level of investment across the board for nonstrategic systems (for example, shipping and billing).

Figure 3: Load-leveling effect of queues between publishers and subscribers



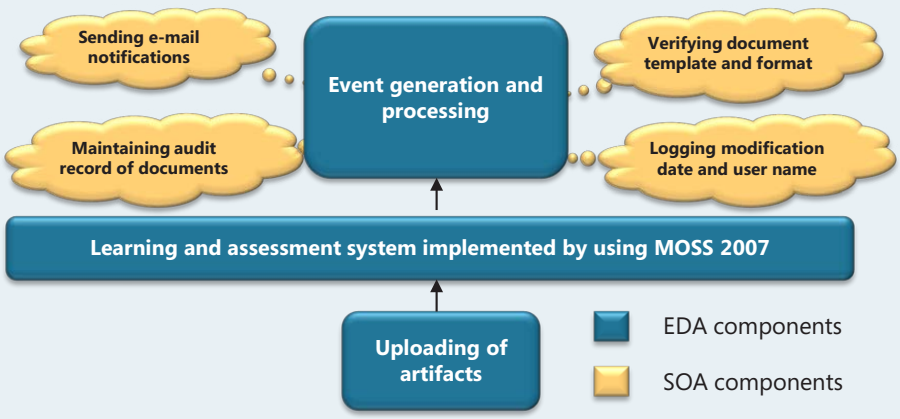
Service-Oriented Architecture and Its Implication in Business

by Manoj Manuja, Rajender Kalra, and Ruchi Malhotra

Service-oriented architecture (SOA) is one of the more buzzed architectures that are being adopted in business these days. It is meant to create a business model that is agile, flexible, and cost-effective. SOA is complemented by event-driven architecture (EDA). An EDA can be designed with systems that produce and transmit events among the loosely coupled service-oriented systems. These events fire certain triggers that, in turn, activate the services.

This can be depicted by a case study of a learning and assessment system that exists in a business enterprise (see Figure 1). The whole enterprise is driven by an SOA that is designed to handle and integrate copious subsystems. The learning infrastructure that is in place is meant to develop the competency of employees on various technologies that are prevalent in the software industry. There is a separate team of technology experts in place who run the complete infrastructure. The company relies on its own proprietary study material and other artifacts for training and assessment of its employees.

Figure 1: SOA supported by EDA implemented by using MOSS 2007



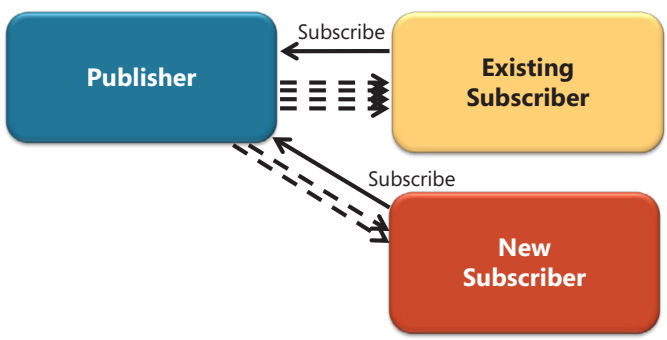
The whole infrastructure for the management and access of these artifacts by the technology-focus team and the employees of the company has been automated by using Microsoft Office SharePoint Server (MOSS) 2007, which is the latest portal-development product from Microsoft. The artifacts are updated on a monthly basis. This process is decentralized and is performed by the respective technology teams.

In this process, some events are produced that, in turn, fire triggers. The triggers activate services that exist in the complete organizational infrastructure that is based on SOA. The services carry out tasks such as sending e-mail notifications to the concerned persons, maintaining an audit record of the documents, verifying the document template and format, and logging the modification date and the name of the user who has performed the update. In this way, the SOA of the entire organization is supported by the event-based system that is implemented by using MOSS 2007, as shown in Figure 1.

The role of EDA in implementing SOA is very significant. The functioning of the various services that form part of a SOA can depend on the occurrence of certain events. In the case study that is discussed in this article, the use of MOSS 2007 for management of the company's learning and assessment system largely contributed to the management of documents, as well as the running of the SOA infrastructure within the enterprise. The use of MOSS 2007 decentralizes the whole process, and the events that are produced in it also drive the SOA services. Along with EDA, SOA implementation in an enterprise leads to better productivity and improved customer satisfaction, as it provides increased control over business.

Manoj Manuja (Manoj_Manuja@infosys.com), **Rajender Kalra** (Rajender_Kalra01@infosys.com), and **Ruchi Malhotra** (Ruchi_Malhotra@infosys.com) belong to the Education and Research Department, Infosys Technologies Limited, in Chandigarh, India.

Figure 4: Adding new subscriber to existing publisher



In companies that are undergoing mergers or acquisitions, the ability to add a new subscriber quickly to any number of events from multiple publishers without having to change any code in those publishers is a big win (see Figure 4). This helps maintain stability of the core environment, while iteratively rolling out bridges between the systems of the two companies. When we look practically at bringing the new subscriber online, we can take the recording of all published events from the audit log and play them to the new subscriber, or perform the regular ETL style of data migration from one subscriber to another.

IT-Business Alignment, SOA, and EDA

One of the more profound benefits that SOA was supposed to bring was an improved alignment between IT and business. While the industry does not appear to have settled on how this exactly is supposed to occur, there is broad agreement that IT is currently not aligned with business. Often, this is described under the title of application "silos."

To understand the core problem, let us try to visualize this lack of alignment, as shown in Figure 5.

What we see in this lack of alignment is that IT boundaries are different from business boundaries, so that it is understandable that the focus of SOA on explicit boundaries (from the four tenets of service orientation) would lead many to believe that it is the solution. Yet the problem that we see here is while there *are* explicit technical boundaries between App 1 and App 2, the *mapping* to business boundaries is wrong.

Figure 5: Lack of IT-business alignment

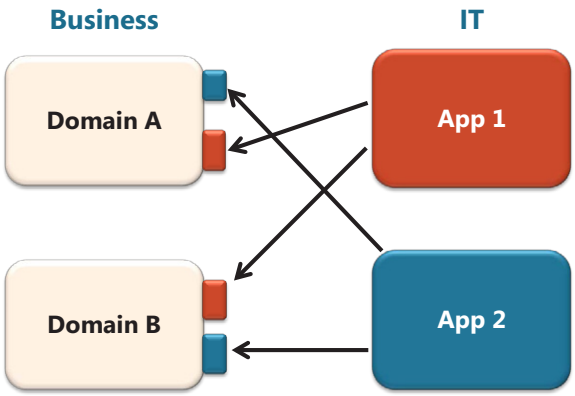
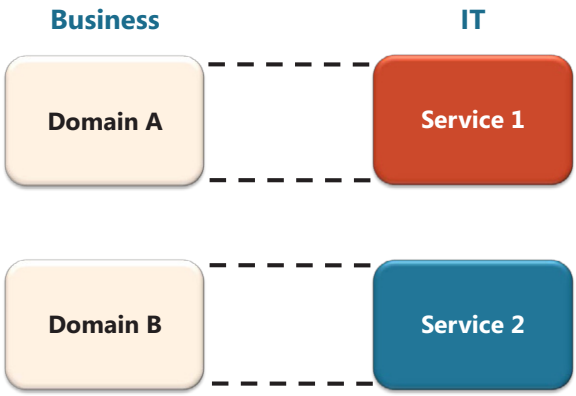


Figure 6: Services aligned with business boundaries



If SOA is to have any chance of improving IT-business alignment, the connection between the two needs to look more like the one that is shown in Figure 6.

One could describe such a connection as a service "owning" or being responsible for a single business domain, so that anything outside the service could not perform any actions that relate to that domain. Also, any and all data that relates to that domain also would be accessible only within the service. The EDA model that we saw earlier enabled exactly that kind of strict separation and ownership—all the while, providing mechanisms for interaction and collaboration.

We should consider this strong connection when we look at rules such as: "When an order has been created, if a customer was not provided, create a new customer." The *creation* of the order as an object or a row in a database has no significance in the business domain. From a business perspective, it could be the *acceptance* or the *authorization* of an order that matters.

> *What SOA brings to EDA in terms of IT-business alignment is the necessity of events to represent meaningful business occurrences.*

For example, instead of thinking of an entity that is being deleted as an event, you should look for the business scenario around it—for example, a product that is being discontinued, a discount that is being revoked, or a shipment that is being canceled. Consider introducing a meaningful business status to your entities, instead of the technically common "deleted" column. While the business domain of sales will probably not be very interested in discontinued products and might treat them as deleted, the support domain might need to continue troubleshooting the problems that clients have with those products—for a while, at least. Modern-day collaborative business-analysis methodologies such as value networks can help identify these domains and the event flows between them.

What an EDA/SOA Service Looks Like

In the context of combined EDA and SOA, the word "service" is equivalent to a logical "thing" that can have a database schema, Web Services, and even user-interface (UI) code inside it. This is a very different perspective from the classic approach that considers services as just another layer of the architecture. In this context, services cut across multiple layers, as shown in Figure 7.

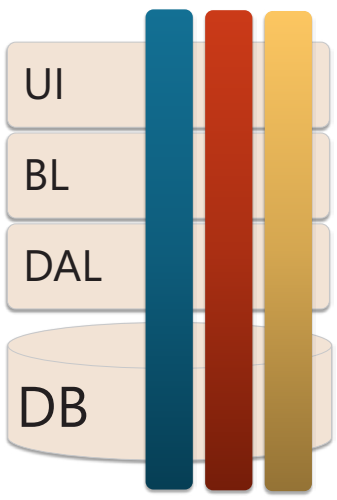
In this model, the processes that are running on various computers serve as generic, composite hosts of service code and have no real logical "meat" to them.

When we look at the code in each of the layers in light of the business domain that it addresses, we tend to see fairly tight coupling between a screen, its logic, and the data that it shows. The places in which we see loose coupling is between screens, logic, and data from different business domains; there is hardly any coupling (if at all) between the screen that shows employee details and the one that is used to cancel an order. The fact that both are screens and are categorized in the UI "layer" appears not to have *much* technical significance (if any business significance). Much the same can be said for the code that hooks those screens to the data, as well as the data structures themselves.

Any consistency concerns that might have arisen by this separation have already been addressed by the business acceptance of eventual consistency. If there are business demands that two pieces of data that have been allocated to different services always be consistent, this indicates that service boundaries are not aligned with business boundaries and must be changed.

This is extremely valuable. Architects can explain to the business the ramifications of their architectural decisions in ways that the business can understand—"There might be a couple of seconds during which these two bits of data are not in sync. Is that a problem?"—and the answer to those kinds of question is used to iterate the architecture, so as to bring it into better alignment with the business.

Figure 7: Services logically connecting code from different layers



As soon as service boundaries reflect business boundaries, there is great flexibility within each service; each can change its own database schema without having to worry about breaking other services, or even choose to change vendors and technology to

such things as object or XML databases. Interoperability between services is a question of how event structures are represented, as well as how publish/subscribe is handled. This can be done by using basic enterprise service bus (ESB) functionality, such things as the Atom Publishing Protocol, or a mix.

Integration of legacy applications in this environment occurs *within* the context of a service, instead of identifying them as services in their own right. Use of Web Services to ease the cost of integration continues to make sense; however, from the perspective of a business domain, it really is nothing more than an implementation detail.

Conclusion

EDA is not a technical panacea to Web Services–centric architectures. In fact, attempting to employ EDA principles on purely technical domains that implement command-centric business analysis will almost certainly fail. The introduction of eventual consistency without the ratification of business stakeholders is poorly advised.

However, if in the process of architecture we work collaboratively with the business, map out the natural boundaries that are inherent in the organization and the way in which it works, and align the boundaries of our services to them, we will find that the benefits of EDA bring substantial gains to the business in terms of greater flexibility and shorter times to market, while its apparent disadvantages become addressed in terms of additional entity statuses and finer-grained events.

By itself, EDA ignores the IT-business alignment of SOA—so critical to getting boundaries and events right. Classic SOA has largely ignored the rock-solid foundation of publish/subscribe events—dead Web Services eventing and notification standards notwithstanding. It is only in the fusing of these two approaches that they overcome the weaknesses of each other and create a whole that is greater than the sum of its parts.

Interestingly enough, even though we have almost literally turned the classic command-driven services on their heads, the service-oriented tenets of autonomy and explicit boundaries have only become more pronounced, and the goal of IT-business alignment is now within our grasp.

Beyond just being a sound theoretical foundation, this architecture has weathered the trials of production in domains such as finance, travel and hospitality, aerospace, and many others—each with its own challenging constraints and nonfunctional demands. Organizations have maximized the effectiveness of their development teams by structuring them in accordance with these same service boundaries, instead of the more common technical specialization that corresponds to layered architectures. These loosely coupled service teams were able to wring the most out of their agile methodologies, as competition for specialized shared resources was eliminated.

Oracle once named this approach SOA 2.0. Maybe it really is the next evolutionary step.

Toward Web-Scale SOA

by Carlos Pedrinaci, Elena Simperl, Reto Krummenacher, and Barry Norton

Opening SOA technologies to the Web has important implications from an engineering perspective, as well as with respect to the usage of technology that one should expect and accommodate. Typically, these implications have been overlooked; as a consequence, SOA remains mostly an enterprise-specific solution, and its adoption for supporting the creation of distributed systems on the Web has largely fallen behind initial expectations.

The Web is built upon the essential principle of *openness*, which has defined its character and led to its growth to become the world's richest source of information in its 20 years of existence. Anyone can provide information, and anyone else can use this information for whatever purpose is deemed appropriate. However, this lack of centralized control and the scale that characterizes the Web clash with current Web Services and SOA-based solutions, in which services are typically known in advance and stored in some centralized repository, and flexibility in consumption is not contemplated.

A second Web principle of particular relevance is the basis of communication through *persistent publication*. This style of communication allows efficient asynchronous communication between one provider and many (possibly unknown) consumers. This simple mechanism supports an unprecedented level of creation, flow, and recombination of information, which contributes significantly to the enrichment of the Web. In contrast, service-oriented systems are most often limited to communication on the basis of one-to-one synchronous messaging. This fails to take into account that systems might suddenly disappear and new consumers who have different requirements might appear, and it maintains unnecessarily a strict separation between procedures and data.

The E.U. project that is known as Service Oriented Architectures for All (SOA4All) is working toward an architecture and language stack for a service-delivery platform that fosters the Web-scale adoption of service technologies. This architecture extends SOA with essential principles upon which the Web builds, such as openness and the support for communication that is based on persistent publication. Additionally, we adopt semantic technologies as a means to lift services and their descriptions to a level of abstraction that deals with computer-understandable conceptualizations, so as to increase the level of automation that can be achieved while carrying out common tasks during the life cycle of services, such as their discovery, composition, and invocation.

Further extensions come from the adoption of Web 2.0 principles—notably, the tight integration of people as service prosumers and RESTful services as a technology that complements traditional Web services. Finally, automated context-adaptation capabilities are embedded within the architecture to support the use of services in unforeseen contexts—thus, increasing the versatility of services while retaining their manageability.

For more information, visit the Service Oriented Architectures for All (SOA4All) Web site.

Dr. Carlos Pedrinaci is a research fellow at the Knowledge Media Institute (KMi) at the Open University, United Kingdom.

Dr. Elena Simperl works as senior researcher and vice-director of the Semantic Technology Institute at the University of Innsbruck.

Reto Krummenacher is researcher and project assistant for the Semantic Technology Institute at the University of Innsbruck.

Barry Norton is a senior researcher for STI Innsbruck at the University of Innsbruck.

References

ACID
Four Tenets of Service Orientation
Value Networks
The Atom Publishing Protocol

About the Author

Udi Dahan is an internationally renowned expert on software architecture and design. Recognized four years in a row with the coveted Most Valuable Professional (MVP) award by Microsoft Corporation for solutions architecture and connected systems, he is also on the advisory board of Microsoft's next-generation technology platforms: WCF/WF/OSLO, the Software Factories Initiative, and the Composite Application Library & Guidance. He provides clients all over the world with training, mentoring, and high end–architecture consulting services—specializing in service-oriented, scalable, and secure enterprise architecture and design.

Dahan is one of 33 experts in Europe who are recognized by the International .NET Association (INETA); an author and trainer for the International Association of Software Architects on Reliability, Availability, and Scalability; and an SOA, Web Services, and XML guru who is recommended by *Dr. Dobb's*—the world's largest software magazine.

Follow up on this topic

- Enterprise Service Bus Toolkit 2.0

How Managed Is Your SOA?

by Aarti Kaur

The dynamic, collaborative nature of today's business processes makes it necessary to have a scalable and flexible integration approach. This is when the paradigm shift occurred and the IT industry came up with SOA as one of the most viable solutions for EAI. The big question is whether implementing a pure SOA will address the real-time business issues in the long term.

Business Problem

The basic services that are provided by telecom providers are commoditized, which forces telecom operators to come up with innovative value-added services that will distinguish them from their competitors and help them retain their existing subscriber base. Introduction of a new service means the ability to test the service quickly in the market and, if it is good enough, the ability to scale as quickly as possible. On the other hand, if the service is not very popular, telecom operators should be able to decommission the service and replace it with another, without a lot of integration effort. Introduction of a value-added service is marred by various issues—such as time-to-market delay, fragmented OSS/BSS platforms, and high operational and development costs—which results in a lack of flexibility to introduce new business models and products rapidly.

SOA Solution

Business-Process Centralization

One of the key issues with the current system is business-process redundancy; that is, based on the type of service that is requested, the appropriate provisioning and billing subsystems are invoked. This process can be automated as an orchestration that consists of all the steps (validation, provisioning, and billing) in a sequential arrangement. The orchestration invokes the respective generic service at each step.

Service Identification and Design

In the current context, one of business pain areas is integration with legacy systems, which leads to high integration and maintenance costs. Introduction of a level of service abstraction that is composed of a set of generic services (which will implement the major service-processing steps, such as validation, provisioning, and billing) can address the aforementioned issue. These, in turn, will call core services that are based on the requested service type, line type, and other parameters. Core services will interact with the BSS/OSS systems, whenever it is required. Some examples of core services are the SMS Validation Service, SMS Provisioning Service, SMS Billing service, and MMS Provisioning Service.

Event-Driven SOA

Events can be of two types: internal system events and business events. A *business event* is any meaningful activity that alters the flow of a business process or triggers a new process. The subscription to a service by an end user is an example of a business event, which will invoke the orchestration. Usually, services follow a simple request-reply communication, which might not be appropriate in the current scenario. For instance, the generic provisioning service will interact with the appropriate provisioning system; the time that is taken to provision a service can vary, depending on the type of service that is requested. In this case, the provisioning service should be able to signal the orchestration when the provisioning is complete. This is an example of an internal event and can be addressed by using asynchronous Web services, such as "WaitHandlers" or the WCF Callback mechanism.

Managing SOA with ESB

ESB is the infrastructure for managing an SOA. The Microsoft ESB Toolkit 2.0 itinerary Web service can be used for service discovery and invocation. The resolver mechanism can provide a generic service resolution. An itinerary service can be an entry point to this system; it will read the message and use the ESB resolver mechanism internally to invoke the business-process orchestration. Addition of a new service in the system means registering the service in a service registry and using the appropriate discovery mechanism. Message transformations between services can be performed by using Microsoft BizTalk maps—thus, each service will complete its processing, and the ESB will take care of the message transformations and routing.

Conclusion

SOA is a very powerful architectural concept; but, all by itself, it might not address the dynamics of a real-time business. Managing SOA with ESB offers many potential benefits; also, events and services complement each other and cannot be looked at in isolation from one other. In combination with event processing and ESB, SOA can provide an optimal solution that not only addresses the business complexities, but also adds value to it.

Aarti Kaur is a solution architect with Mahindra Satyam, India. The complete article is available on her blog.

Is SOA Being Pushed Beyond Its Limits?

by Grace A. Lewis

Summary

This article presents some of the characteristics of future service-oriented systems. Also, it focuses on a set of architecture and design drivers for these future service-oriented systems that can help meet new expectations without sacrificing the loosely coupled, stateless, standards-based characteristics that have driven SOA adoption in many contexts. The article concludes with thoughts on the key role of the architect in the service-oriented systems-development process

Introduction

It is clear that service-oriented architecture (SOA) is having a substantial impact on the way in which software systems are developed. According to a 2007 Gartner Group report, 50 percent of new mission-critical operational applications and business processes were designed in 2007 around SOA, and that number will be more than 80 percent by 2010. Despite recent news that SOA adoption rates are falling and that "SOA is dead," Forrester Group recently reported that SOA adoption is increasing across all of its vertical-industry groups. The reality is that SOA is currently the best option available for systems integration and leverage of legacy systems.

SOA is a way of designing, developing, deploying, and managing systems, and it is characterized by coarse-grained services that represent reusable business functionality. Service consumers compose applications or systems by using the functionality that services provide through standard interfaces.

At a high level:

- Services provide reusable business functionality.
- Service consumers are built by using the functionality from available services. Service-interface definitions are first-class artifacts.
- There is a clear separation between the service interface and service implementation that come from the legacy systems, external systems, or code that was built specifically for this purpose.
- An SOA infrastructure enables the discovery, composition, and invocation of services.
- Protocols are predominantly, but not exclusively, message-based document exchanges.

From a more technical point of view, SOA is an architectural style or design paradigm; it is neither a system architecture nor a complete system. As an architectural style, it is characterized by a set of components and connectors, situations in which the style is applicable, and benefits that are associated with implementing the style.

If it is implemented correctly, SOA adoption can provide business agility, reuse of business functionality, and leverage of legacy systems for an organization. Many organizations recognize these potential benefits and are adopting SOA—some more successfully than others. SOA has indeed "crossed the chasm,"[1] according to a recent Software AG user survey in which 90 percent of the respondents claim to have made some commitment to SOA adoption.[2]

However, as with any technology, as SOA is adopted within organizations and becomes a mainstream paradigm for systems development, the requirements and expectations that are placed on service orientation increase. What was initially an approach for asynchronous document-based message exchanges now has performance, availability, reliability, security and other expectations of traditional distributed systems. As a result, the loosely coupled, stateless, standards-based nature of the relationship between service consumers and service providers in service-oriented systems is changing, so as to meet these new requirements. In addition, global enterprises and the emerging market of third-party services that are being made available through the Cloud are also placing expectations on service-oriented system architecture and design.

The first part of this article presents some of the characteristics of future service-oriented systems. The second part focuses on a set of architecture and design drivers for these future service-oriented systems that can help meet new expectations without sacrificing the loosely coupled, stateless, standards-based characteristics that have driven SOA adoption in many contexts. Finally, the article concludes with thoughts on the key role of the architect in the service-oriented systems-development process.

Future Service-Oriented Systems

Between 2005 and 2007, multiple surveys were conducted by organizations such as Forrester, Gartner, and IDC that showed that the top drivers for SOA adoption were mainly internally focused: application integration, data integration, and internal process improvement. This fact is changing. A recent survey published by Forrester shows that the number of organizations that are currently using SOA for external integration is approximately one third of the surveyed organizations.[3] While the percentage of externally focused SOA applications is still a minority, this percentage has been growing, and the trend will continue as organizations look at SOA adoption for supply-chain integration, access to real-time data, and cost reduction through the use of third-party services via the Cloud or Software as a Service (SaaS). As organizations expand their systems to cross

organizational boundaries, the requirements on their systems also expand—from consumer, provider, and infrastructure perspectives. What follows are some requirements that will be typical of these future (or even current) service-oriented systems.

Security

The security threats for service-oriented systems are not new or different; it is the level of exposure that is greater. Service-oriented systems have an unknown and dynamic attack surface. *Attack surface* refers to the set of ways in which an adversary can exploit vulnerabilities and potentially cause damage. An attack surface can be measured in terms of three kinds of resources that are used in attacks on the system: methods (for example, an API), channels (for example, sockets), and data (for example, input parameters). The greater the number of resources that are accessible for attack, the greater the attack surface and, therefore, the more insecure the software environment.[4] From a more global perspective of security, issues such as identity management, dynamic secure-service composition, and trust in third-party services become important requirements in this type of system.

Runtime Monitoring and Adaptation

Runtime monitoring of systems is a common practice for determining the health of a system. SOA infrastructures can be configured to gather certain measures during system execution, and tools can be integrated into the system to produce reports and alerts if measures cross certain thresholds. *Runtime adaptation* refers to the capability of the system to adjust itself at runtime when these thresholds are crossed, so as to continue to meet quality requirements. For example, a system might start an additional instance of a service under particular load conditions or restrict access to a service if there is a suspicion that the security of the system has been compromised. These actions are possible when there is full control over a system; however, when services belong to third parties, there is much less control, and it becomes difficult to weave and consolidate the different logs that are emitted from different sources to paint an overall picture of the system.

Dynamic Binding

The word *dynamic* is often used to describe the binding between service consumers and services. There are various degrees of dynamism. At the lower end of the spectrum is late binding of a proxy service to a specific service instance that depends on user context or load-balancing policies. At the higher end of the spectrum is fully dynamic binding in which service consumers are capable of querying service registries at runtime, selecting the "best" service from the list of returned services, and invoking the selected service—all at runtime, and without human intervention. Late binding is a common, out-of-the-box feature of many commercial and open-source SOA infrastructures, such as an enterprise service bus (ESB). Fully dynamic binding, on the other hand, requires semantically described services that use an ontology that is shared between service consumers and service providers. Semantic Web Services represent an active area of research, as well as an unsolved problem that is not yet ready for large-scale deployment.

Multiple Consumers and Consumer Devices

As service-oriented systems start crossing organizational boundaries, the variety of service consumers will increase. Services will have to deal with heterogeneous service-consumer development and computing platforms. The proliferation and increasing power of

SOA at Credit Suisse: Success Through Continuous Governance

by Dirk Becker

SOA at Credit Suisse is motivated by cost reduction and service-quality improvement. The success of SOA has proven that both objectives can be achieved by implementing a managed service-oriented infrastructure; together, they guarantee the continuous evolution and support of SOA at Credit Suisse.

The SOA services at Credit Suisse are geared for reuse; therefore, they have to be easy to find, build, understand, and reliable to use. The following are the three major, key success factors:

- Service design
- Developer support
- Distributed ownership

Service design—Credit Suisse has a strictly governed process for the design of services that are considered to be reusable. This process includes rules on the general semantics of services, the usage of reusable data structures, and the documentation of service contracts. Every reusable service design must go through a central quality-assurance process.

Developer support—An application developer at Credit Suisse does not have to be concerned with any aspect of the infrastructure technology. The service-design process is linked with the tool chain to generate all infrastructure components. The same process guarantees the correct descriptions for the deployment and the configuration of the naming service. As a result, the developer receives a real benefit, while the service governance ensures that only services that have gone through the design process will be available on the naming service.

Distributed ownership—Efficient governance cannot be run by a central organization only. It needs the contribution of the whole organization; otherwise, it will inevitably become a bottleneck. The Credit Suisse service-design process is owned jointly by the central architecture team and the IT business organizations; these, in turn, run the application-development projects. The amount of effort that is invested in the application project determines the processing time of the service request, which assures motivated participation.

Starting in 1998, this setup has allowed for the continuous management support of the SOA program, even when the results of the first years have not shown a substantial benefit. It took about three years before the number of services reached a critical mass (about 800 services), and sufficient trust on the side of the service consumer has developed to let the number of users rise and the number of service calls surge. Today, 10 years after the start of the program, the promise of SOA for reusable and reliable services has become a well-accepted reality with over 3,000 managed services, and it will continue to do so in the future.

Dirk Becker (dirk.becker@credit-suisse.com) is a member of the global-integration architecture team at Credit Suisse.

handheld devices, along with the need for access to real-time data, are driving business applications to run on resource-constrained devices such as handheld devices, PDAs, and cell phones. In the case of third-party service providers, the fact that service consumers might be unknown adds an additional requirement of anticipating potential consumer profiles and usage patterns.

Coexistence with Other Architectural Paradigms and Technologies

Because Web Services are the main standards-based technology that is available today for implementation of service-oriented systems, a common misconception is that Web Services and SOA are the same. In fact, Web Services are only one potential approach to SOA implementation. In a traditional Web Services environment, service consumers interact with services via XML-messages that are encoded by using SOAP over HTTP in a request/response manner. While this is appropriate in many contexts, especially in enterprise contexts, it might not be appropriate in other contexts of high-performance or real-time requirements. For example, certain business processes or real-time workflows might be too dynamic and complex to be modeled by traditional sequential processing methods. In this case, event-driven SOA provides a potential solution by combining the traditional SOA request/response paradigm with the event-driven architecture (EDA) event publish/subscribe paradigm.

Another example is high-performance and real-time systems that are usually tied to requirements for higher information bandwidth, as well as much lower latencies or delays on the information. As demands become more real-time, the need for performance, predictability, and load balancing tips the scale towards point-to-point (P2P), tightly coupled architectures, as opposed to more loosely coupled architectures. Common SOA implementations that are based on HTTP, such as Web Services, might not be acceptable, because HTTP is not reliable, has limited bandwidth, introduces very high latencies, and cannot buffer, queue, and deliver messages to systems that are either temporarily unavailable or will join at a later time. For this reason, real-time support in SOA environments focuses on EDAs and publish/subscribe systems as a way to support real-time requirements, yet maintain the loosely coupled nature of service-oriented systems.[5, 6]

As service-oriented systems depart from what is currently standardized—mainly, Web Services (whether WS* or REST)—there will be trade-offs. For example, maintainability of the system becomes more difficult when there are multiple architecture paradigms and when tool availability decreases.

Governance

The requirement for governance will not come as an explicit requirement; however, as systems start to cross organizational boundaries, the need for governance becomes even more important. SOA governance is the set of policies, rules, and enforcement mechanisms for developing, using, and evolving service-oriented systems, as well as for analysis of their business value. It includes policies and procedures, roles and responsibilities, design-time governance, and run-time governance.[7, 8, 9] Design-time governance includes elements such as rules for strategic identification of services, development, and deployment of services; reuse; and migration of legacy systems. It also enforces consistency in the use of standards, SOA infrastructure, and processes. Run-time governance develops and enforces rules to ensure that services are executed only in ways that are legal, and that important run-time data is logged. From a life-cycle point of view, design-time governance applies to early activities such as planning, architecture, design, and development. Run-time governance applies to the deployment and management of service-oriented systems. In a multiorganizational environment, governance has to be extended to include policies and procedures for the identification and binding to external services and the establishment and monitoring of service-level agreements (SLAs) between service providers and consumers.

Architecture and Design Drivers for Future Service-Oriented Systems

The requirements for future service-oriented systems present a challenge to system architects. Ideally, the goal is to meet new expectations without sacrificing the loosely coupled, stateless, standards-based characteristics that have driven SOA adoption in many contexts. What follows are some architecture and design drivers that will have to be embedded into these systems.

Context Awareness

In a context-aware SOA environment, services can be selected and adapted every time in accordance with the user and invocation-context requirements and profiles—for example, provision of a service that:

- Has different performance, reliability, or security characteristics, according to who invokes the service and from where it is invoked.
- Returns information that is based on the language, time zone, and invocation environment of the user.
- Returns different views of data, depending on the characteristics of the device from where it is invoked.

To enable loose coupling between service consumers and services, the system architecture will have to abstract the complexity and multiplicity of implementation options. Architects will have to make trade-offs, such as whether services will expose a single standardized interface and a robust infrastructure will handle all of the necessary transformations and routing, or whether multiple service interfaces are exposed, which places fewer requirements on the infrastructure (probably, at the expense of maintainability). From a technology perspective, there is currently no standard for representing user context, which means that design decisions must be made to determine when and how user-context information is obtained.[8]

Instrumentation for Runtime Monitoring and Adaptation

If a service-oriented system includes runtime monitoring and adaptation, all system elements must be instrumented so as to gather the right measures and receive the proper "orders" on what to do when thresholds are crossed. From an architecture and design perspective, this translates into architectural constructs for measurement and instrumentation in the SOA infrastructure, services, and even service consumers. Ideally, these constructs should be highly configurable so as to accommodate SLA changes and changes in service providers. For example, recent research shows that to design self-adaptive systems, the feedback loops that control self-adaptation must become first-class entities at the expense of added complexity.[9]

Service Usability

In a growing market of third-party service brokers and providers, the aspects that can make a service more or less attractive include functionality, attached SLAs, and usability. Characteristics that make a service more usable or less usable can include interface design, options in messaging protocols, add-ons (such as test cases and test instances), and any other metadata that can tell consumers more about the service. Therefore, the task of service-interface design extends beyond simply defining the messages that are exchanged

Considerations when Building an Enhanced SOA Framework

by Alex Cameron

Service-oriented architecture (SOA) is a style that is maturing rapidly and taking us in new directions. When it is implemented, it can just as easily take the form of a situational mash-up as it can of an enterprise architecture. So, as the implementations flourish and become highly network-centric and heterogeneous, there will be a need for enhancement technologies. These technologies will provide not only the telemetry and control mechanisms, but the means to mine the strategic information that the business needs.

So, What Is an Enhanced SOA?

Put simply, *enhanced SOA* is the incorporation of business- and infrastructure-enhancing components to form a comprehensive SOA-enabled business and technology platform. These "enhancing" functions have a singular purpose, which is to underpin and derive value from the architecture and make it appear greater than the sum of its parts. The Identity Life-Cycle Management System (IDLMS), Enterprise Decision Management (EDM), Service Component Architecture (SCA), Event Stream Processing (ESP)/Complex Event Processing (CEP), Enterprise Service Bus (ESB) and, not least, Business Process Management (BPM) are emerging and the most prominent. The ability to absorb and ultimately be enhanced by these technologies represents the true genius of SOA.

Steps Toward Building an Enhanced SOA

Building an enhanced SOA requires that a true evolutionary approach be adopted with a clear set of guiding principles in place from the outset. These principles not only will help make sense of the technologies, but they also will guide the take-up. Proceeding without them will lead to a technology-for-technology's-sake–based approach; such approaches invariably fail, in one way or another. Let us consider the following as guidelines:

- Agree that a BPM layer will be implemented with a long-living Business Process Execution Language (BPEL) process that represents the business-process models. -> BPM
- Recognize that there is a natural demarcation between business process and business decision points. This will allow for the externalization of decision-making. -> EDM
- Agree that all architectural interaction will occur via state-defining asynchronous events. -> ESP/CEP
- Normalize system messages and develop standard patterns, so that your architecture will scale and remain relevant. -> ESB
- Virtualize and componentize services to increase business-level awareness, better infrastructure utilization, reuse of services, and reduction of complexity. -> SCA
- Approach security as a federation of services in which endpoints are policy-driven and become responsive to changing needs. -> IDLMS

Benefits of Developing an Enhanced SOA

From a Business Perspective

- As the business owner, you now have the tools that will give you far greater insight into and control of the systems that run your business.

From a Technical Perspective

1. As the information architect, you will have the ability to conceptualize, simulate, and monitor compliance against the desired business outcomes.
2. As the developer, you now have a clearer separation of concerns and should see a more constrained system that offers higher levels of productivity and maintainability.

Alex Cameron (Alexander.Cameron@hp.com) is a Fellow and Enterprise Architect within the HP TSG Group. Visit the Web site.

between providers and consumers. For example, architectural constructs would have to be put in place to support advanced service registries, multiple messaging options, test instances, SLA monitoring, and any other characteristic that contributes to the perception of service usability.

Federation

As service-oriented systems grow in size, the centralization of certain aspects might become a bottleneck. Federation can be a solution to this problem. In this context, federation refers to predefined agreements on aspects of the system that allow the autonomy of individual components.

Some aspects of service-oriented systems that might require federation in large-scale settings are:

- **Identity management.** This is the aspect that is most commonly associated with federation in SOA environments. Federated identity management means that there is a cooperative contract that has been set up among multiple identity providers and uses a decentralized approach, so that an identity in one of the identity providers is recognized by other identity providers in the federation.[12] From a consumer perspective, this means not having to log in to every single system that is involved in the execution of a particular business process or workflow. Some of the challenges of federated identity management include trust, translation among multiple standards, and synchronization.
- **SOA infrastructures.** In large-scale service-oriented systems that span multiple organizations, it is unlikely that all organizations will have the same SOA infrastructure. In this case, federation would allow participating organizations to maintain their SOA infrastructures, while shared aspects such as policy management and governance mechanisms are agreed upon, propagated throughout the system, and implemented locally.
- **Service registries.** Federated service registries allow registries to appear as a single, virtual registry and individual organizations to retain local control over their own registries.

Regardless of the aspect of the system that is federated, there will need to be architectural constructs for establishing agreements, virtualization, and synchronization upon changes.

Automated Governance

The key to governance implementation is adding control to a system without creating a lot of extra work to its developers and users. The approach is governance automation. The burden of ensuring compliance and enforcement gets pushed to the SOA infrastructure. There are tools and SOA infrastructures in which some governance

automation is built in; in the end, however, the goal is the ability of an organization to ensure that development and deployment adhere to its own policies and standards, which might not be what is codified in existing tools.

Some aspects of governance that can be automated are:

- Workflows for service identification.
- Service-deployment procedures.
- Compliance with regulations such as the Health Insurance Portability and Accountability Act (HIPAA) and Sarbanes-Oxley.
- Compliance with internal security policies.
- Runtime measurements and logging.
- SLA management.

Architectural constructs will need to be developed for aspects of SOA governance that are critical for SOA implementation and are not covered (or are implemented differently) by the existing SOA infrastructure and SOA governance tools. In multiorganizational settings, the challenge is how to deal with conflicting policies and procedures among organizations.

Specialized SOA Design Patterns

In software engineering, a *design pattern* is a general reusable solution to a commonly occurring problem in software design. Gamma *et al*. produced a set of design patterns for object-oriented systems that triggered the usage of the term in software design.[10] Since then, design patterns have been produced for different types of systems, including service-oriented systems.[11]

Given the expectations that are being placed on service-oriented systems, architects will have to build and research patterns to address the expectations of future service-oriented systems. This includes patterns for:

- Service orientation in multiorganizational environments.
- Embedding system qualities into SOA infrastructures.
- Service-interface design.
- Integration with other technologies.

Conclusion

SOA is potentially being stretched beyond its limits. What was initially an approach for asynchronous document-based message exchanges now has performance, availability, reliability, security, and other expectations of traditional distributed systems. To solve this problem, multiple specifications and standards have been proposed and created, middleware products are becoming more robust, and the community has started to embrace terms such as event-driven SOA and real-time SOA. Therefore, the loosely coupled, stateless, standards-based nature of the relationship between service consumers and service providers in service-oriented systems is changing, so as to meet these new requirements. In addition, global enterprises and the emerging market of third-party services that are being made available through the Cloud are placing new expectations on service-oriented system architecture and design.

SOA is not a "one-size-fits-all" solution. As an architectural style, SOA is an appropriate solution in some situations; however, there are situations in which it is not appropriate or it has to be used in conjunction with other technologies to meet service qualities. The architect of future service-oriented systems is going to play a crucial role in determining what expectations can or cannot be met by SOA adoption, and where trade-offs can be made for the benefit of the organization and the accomplishment of system qualities.

- **Early, contextual technology evaluation**—As the use of SOA for external integration and the expectations of SOA adoption increase, many promises will be made on the benefits of SOA in these scenarios that will probably not be validated until implementation. The role of the architect is to perform early, contextual technology evaluation and continuous technology scouting that can lead to more informed decisions on what parts of the system will benefit from SOA technologies.[13]
- **Architecture trade-off analysis**—It is well known that trade-offs must be made in systems, because the accomplishment of a certain quality is often at the expense of another quality. Common examples of trade-offs are performance versus modifiability, availability versus safety, security versus performance, and interoperability versus cost.[14] The use of service orientation in systems that have high system-quality requirements will require architectural trade-offs at the expense of loose coupling and flexibility. If the added overhead for a service-oriented system to meet quality requirements comes at the expense of the characteristics for which SOA is known, the decision to use service-oriented concepts should be reevaluated. An architecture analysis and evaluation method that is guided by business drivers and performed via scenarios in which the usage of SOA technologies is key can also help an architect make better, early, and informed decisions.

Finally, as service-oriented systems start to cross organizational boundaries, architects will have to reevaluate the use of SOA as an architectural style in these systems or to architect their systems in such a way that qualities are met without having to sacrifice the characteristics that have made SOA a worthwhile technology to adopt.

References

1 This term was coined by Geoffrey A. Moore in his book *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers* (Rev. ed. New York: Collins Business Essentials, 2006) and refers to the chasm that exists between visionaries (early adopters) and pragmatists (early majority) from a technology-adoption perspective.

2 Softwareag.com. "Best Practices for SOA Governance User Survey." Software AG, Summer 2008. [Accessed July 13, 2009.]

3 Forrester.com. "Enterprise and SMB Software Survey, North America and Europe, Q4 2008." Forrester, February 2009.

4 Manadhata, Pratyusa K., Kamie M. C. Tan, Roy A. Maxion, and Jeannette M. Wing. "An Approach to Measuring a System's Attack Surface." CMU Technical Report CMU-CS-07-146, August 2007.

5 Pardo-Castellote, Gerardo. "SOA Feature Story: Real-Time SOA Starts with the Messaging Bus!" *SOA World Magazine*, November 2007. [Accessed July 13, 2009.]

6 Joshi, Rajive. "Data-Oriented Architecture: A Loosely Coupled Real-Time SOA." Real-Time Innovations, Inc., August 2007.

7 Simanta, Soumya, Ed Morris, Grace A. Lewis, Sriram Balasubramaniam, and Dennis B. Smith. "A Scenario-Based Technique for Developing SOA Technical Governance." Software Engineering Institute, June 2009. [Accessed July 13, 2009.]

8 Kontogiannnis, Kostas, Grace A. Lewis, and Dennis B. Smith. "A Proposed Taxonomy for SOA Research." In Proceedings of the International Workshop on the Foundations of Service-Oriented Architecture (FSOA 2007). Software Engineering Institute, June 2008. [Accessed July 13, 2009.]

9 Brun, Yuriy, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè,

and Mary Shaw. "Engineering Self-Adaptive Systems Through Feedback Loops." Lecture Notes in Computer Science Hot Topics, Volume 5525, 2009.

10 Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1994.

11 Erl, Thomas. *SOA Design Patterns*. Upper Saddle River, NJ: Prentice Hall, 2009.

12 Balasubramaniam, Sriram, Soumya Simanta, Ed Morris, Grace A. Lewis, and Dennis B. Smith. "Identity Management and its Impact on Federation in a System of Systems Context." Proceedings of the 2009 3rd Annual IEEE Systems Conference, 2009.

13 Lewis, Grace A., and Lutz Wrage. "A Process for Context-Based Technology Evaluation." Software Engineering Institute, June 2005. [Accessed July 13, 2009.]

14 Clements, Paul, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA; London: Addison-Wesley, 2001.

About the Author

Grace Lewis is a Senior Member of the Technical Staff at the Software Engineering Institute (SEI) in Pittsburgh, PA. Currently, she is the lead for the System of Systems Engineering team within the Systems of Systems Practice (SoSP) initiative in the Research, Technology, and Systems Solutions (RTSS) program. Her current interests and projects are in service-oriented architecture (SOA), technologies for systems interoperability, characterization of software-development life-cycle activities in systems of systems environments, and establishing an SOA research agenda. Grace's latest publications include multiple reports and articles on these subjects, as well as a book in the SEI Series in Software Engineering. She is also a member of the technical faculty for the Master in Software Engineering program at Carnegie Mellon University (CMU). Grace holds a B.Sc. in Systems Engineering; an Executive MBA from Icesi University in Cali, Colombia; and a Masters in Software Engineering from CMU.

Extending Service Boundaries to Infrastructure Resources

by Nayan B. Ruparelia and Salim Sheikh

With data centers considering infrastructure resources as service-on-demand resources and amalgamating them with application services, we redefine an SOA service as comprising the server-side application and the infrastructure that hosts it.

The business benefits of this type of architecture are twofold—a reduced infrastructure footprint and a lower cost of service delivery—and are enunciated as follows:

- Commoditization of services enables true SOA-service rationalization that takes place at the service level, instead of separately at the software and hardware boundaries. This reduces the data-center (or infrastructure) footprint for the business and, thus, provides a more environmentally friendly SOA service at lowered costs.
- Encapsulation of an SOA service across the hardware-software stack provides a complete measure of return on investment (ROI)—from application to infrastructure. This gives a more transparent, accurate, and rationalized view of the cost of a service and its time-to-market (TTM) cost. These factors drive down the cost of an SOA service, because it allows the business to measure reusability of the SOA service across all tiers of IT and, thereby, create a framework for more efficient use of services.

Problem Statement

Generally, an SOA service comprises applications that serve client (or consumer) applications. The management of infrastructure resources that the service consumes—specifically, in relation to capacity and performance—is left to the engineer to plan and model manually. This means that service demands might not always be met within the required time constraints, if the capacity management is reactive instead of proactive; thus, there is a danger that the infrastructure will act as a bottleneck.

Conversely, infrastructure resources can remain idle should the service demand be below that which is provisioned by capacity planning. This represents a waste of money on unused infrastructure. Furthermore, capacity planning and modeling, when performed properly, can itself be a very costly and time-consuming exercise and would need to be performed regularly in order to track changing SOA-service workloads over time. Therefore, a more efficient and cost-effective solution is needed for designing and implementing SOA services.

Figure 1: Infrastructure scope

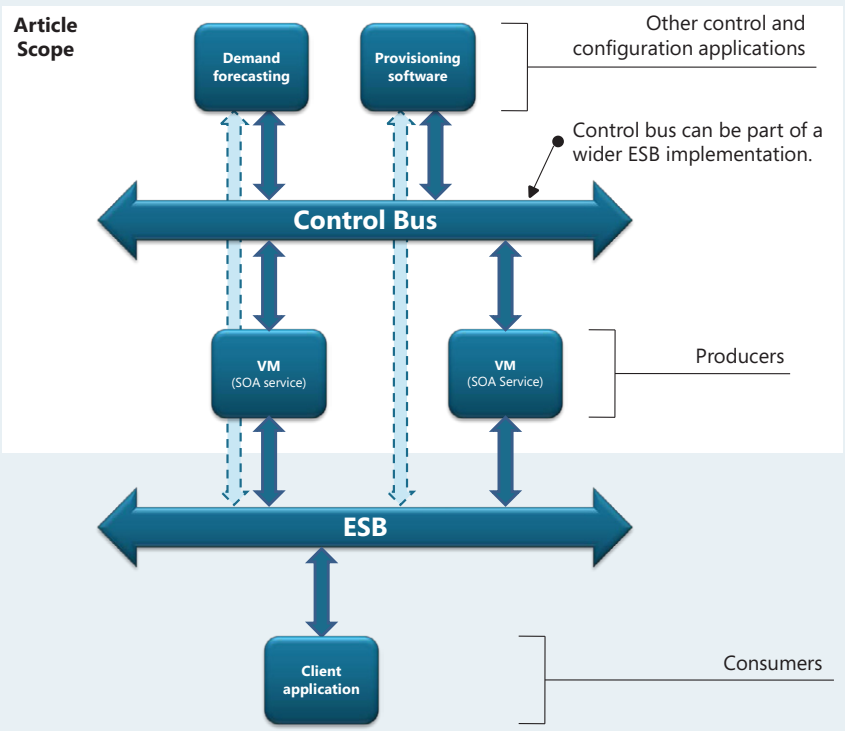
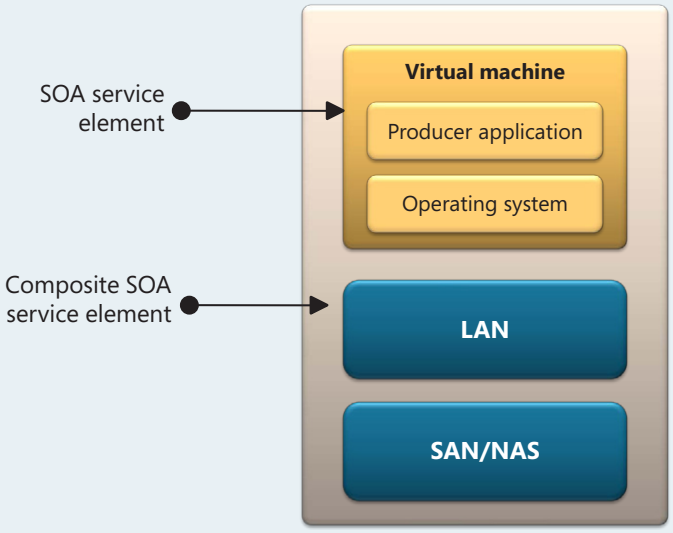


Figure 2: Redefining an SOA service



In this context, we refer to infrastructure resources as the hardware platform and its related services that are required for hosting the SOA applications, not the ESB, as Figure 1 shows.

Proposed Solution

We propose an SOA service that makes up the entire hardware-software stack. We term such an SOA service a "composite service element" that would be provided in a virtualized environment, as Figure 2 shows, in order to better provision the resources of the service and measure its resource usage.

The host platform (comprising hardware computing resources such as CPU, RAM, and local storage) as well as the operating system would be provisioned as a virtual machine (VM). The VM, as much as the software that it hosts, can be considered a service; we denote this VM as an SOA service element. Furthermore, the entire hardware-software stack can be virtualized to include the storage (such as SAN and NAS devices), the LAN, and the interfaces to both of these. This means that all of the layers in the architecture that comprise the logic, data-access, and data-storage layers can be virtualized and provisioned as part of a single SOA service, as Table 1 shows.

Table 1: Components of a proposed composite service element

Layer	Component	Virtualized?	Part of composite service element?
Presentation	Client-side applications	Possibly	No
Logic	Server-side applications	On a VM	Yes
Data access	Database/ middleware	On a VM	Yes
Data storage	SAN, NAS, local storage	Possibly	Yes

To address performance, scalability, and availability requirements, the VM can be configured in a cluster to provide both integrity and workload sharing. The number of VMs that are used in the cluster would be defined by orchestration software that would instantiate VMs as needed. Indeed, the resources of the physical computer that hosts the VM would also be provisioned to suit demand.

In addition, virtualization of the storage and the LAN means that these components can be provisioned individually to meet the workload demand. Demand forecasting can be performed by instruments that measure the resource requirements and predict their usage over certain timeframes by using various load-prediction models.[1] This demand forecast can then be used to drive workflow-automation software (such as the HP Blade Matrix System) that would provision the virtualized resources to meet the predicted demand.

Service-Element Pattern

A *pattern* is a solution to a common problem that is documented in a consistent format and used as part of a larger collection.[2] We propose the *service element* as a pattern that is defined by a VM; thus, the SOA-service design pattern comprises the producer application and its virtual host.

Conclusion

In this article, we redefine an SOA service as being composed of the entire hardware-software stack; we refer to this type of service as a *composite service element*. Furthermore, virtualization is used so that we might be able to provision our SOA service in an efficient and complete manner and with the smallest cost overhead possible. Also, our approach predicts the workload demand and integrates it with workflow-orchestration technologies, so as to automate the provisioning of resources that are used by the composite service element. Apart from a reduction in cost, a benefit of this approach is that it reduces resource consumption in an environmentally friendly manner.

Resources

For additional information, please refer to the blog of the lead author.

References

1. Andreolini, Mauro, and Sara Casolari. "Load-Prediction Models in Web-Based Systems." *ACM International Conference Proceeding Series*. Vol. 180, October 2006.
2. Erl, Thomas. *SOA Design Patterns*. Upper Saddle River, NJ: Prentice Hall, 2009. Page 86.

Nayan B. Ruparelia (nayan@acm.org) is an IT Architect at EDS, part of Hewlett Packard's TSG group.

Salim Sheikh (salim.sheikh@gmail.com) is an Independent Consultant and Enterprise Architect, certified in a number of frameworks including TOGAF, Zachman, COBIT and ITIL. He is also a Certified Process Professional and LEAN/Six Sigma practitioner.

Platform
Architecture Team

Microsoft

21

subscribe at
www.architecturejournal.net