Ontology and Taxonomy of Services in a Service-Oriented Architecture

by Shy Cohen

Summary: This paper describes an ontology and taxonomy for services in a service-oriented architecture (SOA). It discusses the nature of and interrelationships among different service types, describes a generic blueprint for SOA-based systems, and provides some guidance on the construction and management of services. The ontology and taxonomy described here provide a common language for architects, engineers, and business-decision makers, and facilitates better communication within and across different disciplines and organizations. (12 printed pages)

Contents

A Service Taxonomy

Service-oriented economies thrive by promoting composition. In SOAs, new solutions can be created by composing together new application-specific business logic and functionality with existing, recombinant business capabilities. Today, these business capabilities are mostly built in-house, or purchased for on-site deployment as a packaged solution. As we look into the near future, we see continued growth in the Software as a Service (SaaS) model as an option to provide organizations with the ability to "lease" componentized solutions and business capabilities, thus enriching the set of components that can be included in the organization's composite applications.

An *ontology* is a data model that represents a set of concepts within a domain and the relationships among those concepts. A *taxonomy* is a classification of things, as well as the principles underlying such a classification. A *hierarchical taxonomy* is a tree structure of classifications for a given set of objects. This paper defines the service categories in an SOA, the relationships among these categories, and the principles underlying the classification of different service types into the different categories. In addition to defining, classifying, and providing the classification principles, it touches on both the software architecture and the business-related aspects relevant to the construction and management of composite applications in SOAs.

Looking at composition from a software-architecture standpoint, a common ontology and taxonomy enables us to identify the common characteristics of services that fall into a particular category. These characteristics affect the architecture and design of SOA-based solutions from the individual service level up to the entire composite-application. Categorization supports composability by clarifying the roles of the different components, thus helping reason about component interrelationships. Categorization also assists with the discoverability of services (for example, searching for existing services by using a service repository), which can further promote reuse.

Looking at composition from the business standpoint, a proper common ontology and taxonomy can help in making business-related decisions, such as how to obtain a capability (build versus buy versus "lease"), how to manage services (central management versus per application), and so on.

Service Categories and Types

As we examine service types, we notice two main types of services: those that are infrastructural in nature and provide common facilities that would not be considered part of the application, and those that are part of the application and provide the application's building blocks.

Software applications utilize a variety of common facilities ranging from the low-level services offered by the operating system such as the memory management and I/O handling, to the high-level runtime-environment-specific facilities such as the C Runtime Library (RTL), the Java Platform, or the .NET Framework. Solutions built using an SOA make use of common facilities as well, such as a service-authoring framework (for example, Microsoft's Windows Communication Foundation) and a set of services that are part of the supporting distributed computing infrastructure. We will name this set of services **Bus Services** (these are sometimes referred to as infrastructure services).

Bus Services further divide into **Communication Services**, which provide message-transfer facilities such as message-routing and publish-subscribe, and **Utility Services**, which provide capabilities unrelated to message transfer such as service-discovery and identity federation.

The efficiency of software applications development is further increased through reuse of coarse-grained, high-level building blocks. The RAD programming environments that sprang up in the component-oriented era (such as Borland's Delphi or Microsoft's Visual Basic) provided the ability to quickly and easily compose the functionality and capabilities provided by existing building blocks with application-specific code to create new applications. Examples of such components range from the more generic GUI constructs and database access abstractions, to more specific facilities such as charting or event logging. Composite applications in an SOA also use building blocks of this nature in their composition model. We will name these building blocks **Application Services**.

Application Services further divide into **Entity Services**, which expose and allow the manipulation of business entities; **Capability Services** and **Activity Services**, which implement the functional building blocks of the application (sometimes referred to as components or modules); and **Process Services**, which compose and orchestrate entity, capability, and Activity Services to implement business processes.

Figure 1 shows a sample composition of services in the service categories in order to implement a business process. In this sample scenario, a Process Service is shown to be orchestrating three Activity Services and two Capability Services. It is fairly typical for Process Services to tie together multiple other services in order to implement a complex business process. In the diagram, one of the Activity Services (Activity 2) is using a Capability Service (Capability 3) as indicated by the line connecting them. One possible reason for this is that Activity 2 may be implementing a multi-step activity over that Capability Service, or exposing a different service interface than that of the underlying Capability Service. Capability Services 2 and 3 are implemented over Entity Services 2 and 3 (respectively). It is interesting to note that Entity Services 2 and 3 are accessing the same underlying data store. This may be in order to expose different entities from underlying store, or expose the same entity in different ways in order to adhere to different requirements that Capability Services 2 and 3 have regarding the data model. In this sample scenario we can also see that Capability 1 resides outside of the organizational boundary (as indicated by the Internet cloud), and acts as an external resource that is woven into the business process implemented by the Process Service.

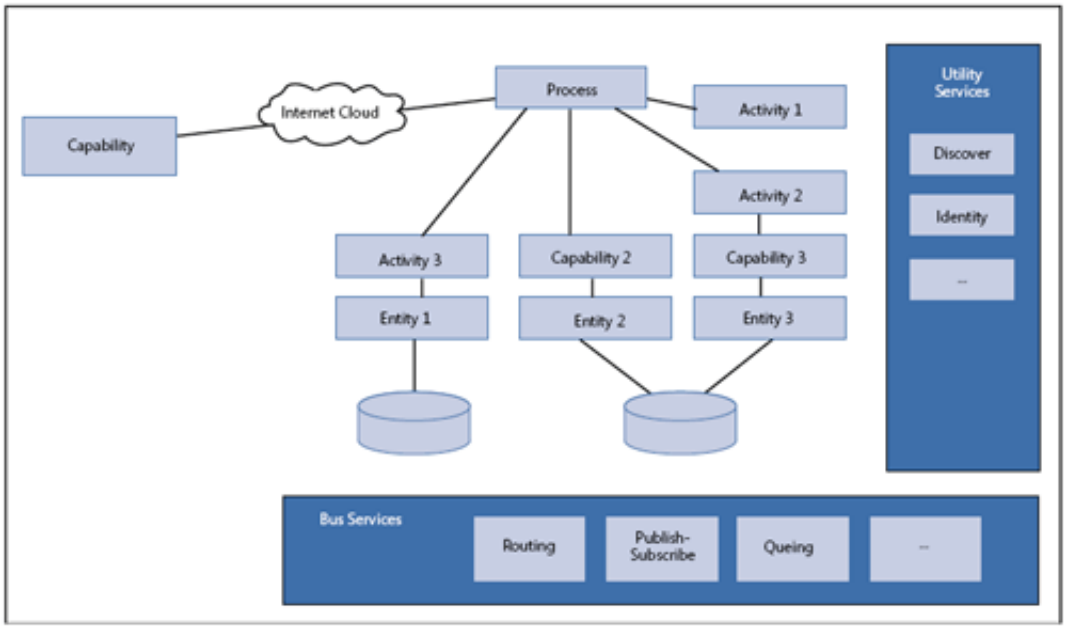

Figure 1. Sample composition of services in the service categories (Click on the picture for a larger image)

Bus Services

Bus Services are common facilities that do not add any explicit business value, but rather are part of the required infrastructure for the implementation of any business process in an SOA. Bus Services are typically purchased or centrally built components that serve multiple applications and, as a consequence, are typically centrally managed.

Communication Services

Communication Services transport messages into, out of, and within the system without being concerned with the content of the messages. For example, a bridge may move messages back and forth across a network barrier (that is, bridging two otherwise disconnected networks) or across a protocol barrier (such as moving queued messages between IBM's WebSphere MQ queuing system and Microsoft's MSMQ queuing system). Examples of Communication Services include relays/bridges/routers/gateways, publish-subscribe systems, and queues.

Communication Services do not hold any application state, but in many cases they are configured to work in concert with the applications that use them. A particular application may need to instruct or configure a Communication Service on how to move the messages flowing inside that application such that intercomponent communication is made possible in a loosely coupled architecture. For example, a content-based router may require the application to provide routing instructions such that the router will know where to forward messages. Another example may be a publish-subscribe service that will deliver messages to registered subscribers based on a filter that can be applied to the message's content. This filter will be set by the application. In both cases, the Communication Service does not process the content of the message but rather (optionally) uses parts of it as instructed in advance by the application for determining where it should go.

In addition to application-specific requirements, restrictions imposed by security, regulatory, or other sources of constraints may dictate that in order to use the facilities offered by a particular Communication Service, users will need to possess certain permissions. These permissions can be set at the application scope (allowing an application to use the service regardless of which user is using the application), at the user scope (allowing a specific user to use the service regardless of which application that the user is using), or at both scopes (allowing the specific user to access the service while running a specific application). For example, a publish-subscribe service may be configured to restrict access to specific topics by only allowing specific users to subscribe to

them.

Other application-level facilities that may be offered by Communication Services pertain to monitoring, diagnostics, and business activity monitoring (BAM). Communication Services may provide statistical information about the application such as an analysis of message traffic patterns (how many messages are flowing through a bridge per minute), error rate reports (how many SOAP faults are being sent through a router per day), or business-level performance indicators (how many purchase orders are coming in through a partner's gateway). Although they may be specific to a particular application, these capabilities are not different than the configuration settings used to control message flow. This information is typically provided by a generic feature of the Communication Service, which often needs to be configured by the application. The statistical information being provided typically needs to be consumed by a specific part of the application that knows what to do with it (raise a security alert at the data center, or update a BAM-related chart on the CFO's computer screen, for example). Table 1 summarizes the characteristics of Communication Services.

Communication Services	
Main purpose	Message transportation
Interface	Not processing application messages; may have Management and Monitoring interfaces
State management	No management of application state
Transactions	Not applicable (they are not processing application messages)
Error handling	No application-related error handling
Security	User/App/Both
Management/Governance	Centralized
How built	Centrally built or purchased

Table 1. Summary of the Communication Services category

Utility Services

Utility Services provide generic, application-agnostic services that deal with aspects other than transporting application messages. Like Communication Services, the functionality they offer is part of the base infrastructure of an SOA and is unrelated to any application-specific logic or business process. For example, a discovery service may be used by components in a loosely coupled composite application to discover other components of the application based on some specified criteria; for example, a service being deployed into a preproduction environment may look for another service that implements a certain interface that the first service needs and that is also deployed in the preproduction environment. Examples of Utility Services include security and idEntity Services (for example, an Identity Federation Service or a Security Token Service), discovery services (such as a UDDI server), and message-transformation services.

As in the case of Communication Services, Utility Services may also be instructed or configured by a particular

application on how to perform an operation on their behalf. For example, a message-transformation service may transform messages from one message schema to another message schema based on a transformation mapping that is provided by the application using the message-transformation service.

Although Utility Services do not hold any application state, the state of a Utility Service may be affected by system state changes. For example, a new user being added to the application may require an update to the credential settings in the Security Token Service. Unlike in the case of Communication Services, Application Services directly interact with the Utility Services that process and (if needed) respond to the messages that the Application Services send them.

Users of Utility Services may require that a permission be configured for them in order to use the service, be it at the application, user, or the application-user scope. For example, a discovery service may only serve domain-authenticated users (users who have valid credentials issues by a Windows domain controller).

Like Communication Services, Utility Services may provide application-level facilities for monitoring, diagnostics, BAM, and so on. These may include statistical information about usage patterns (how many users from another organization authenticated using a federated identity), business-impacting error rates (how many message format transformations of purchase orders failed due to badly formatted incoming messages), and so forth. As with Communication Services, these facilities are typically generic features of the Utility Service and need to be configured and consumed by the particular solution in which they are utilized. Table 2 summarizes the characteristics of Utility Services.

Utility Services	
Main purpose	Generic (non-application-specific) infrastructural functionality
Interface	Service interface for exposing the service's functionality; may also have Management and Monitoring interfaces
State management	No management of application state; may have their own state
Transactions	Typically, not supported
Error handling	No/limited application-related error handling
Security	User/App/Both
Management/Governance	Centralized
How built	Typically, purchased

Table 2. Summary of the Utility Services category

Application Services

Application Services are services that take part in the implementation of a business process. They provide an explicit business value, and exist on a spectrum that starts with generic services that are used in any composite-application in the organization on one end, ends with specialized services that are part of a single composite-

application on the other end, and has services that may be used by two or more applications in between.

Entity Services

Entity Services unlock and surface the business entities in the system. They can be thought of as the data-centric components ("nouns") of the business process: employee, customer, sales order, and so on. Examples of Entity Services include a customers service that manages the customers' information, or an orders service that manages the orders that customers placed.

Entity Services abstract data stores (such as SQL Server or Active Directory) and expose the information stored in one or more data stores in the system through a service interface. Therefore, it is fair to say that Entity Services manage the persistent state of the system. In some cases, the information they manage transcends a specific system and is used in several or even all the systems in the organization.

It is very common for Entity Services to support a create, read, update and delete (CRUD) interface at the entity level, and add additional domain-specific operations needed to address the problem-domain and support the application's features and use cases. An example of a domain-specific operation is a customers service that exposes a method called **FindCustomerByLocation** that can locate a customer's ID given the customer's address.

The information that Entity Services manage typically exists for a time span that is longer than that of any single business process. The information that Entity Services expose is *typically* structured, as opposed to the relational or hierarchical data stores that are being fronted by the service. For example, a service may aggregate the information stored in several database tables or even several separate databases and project that information as a single entity.

In some cases, typically for convenience reasons, Entity Service implementers choose to expose the underlying data as data sets rather than strongly-schematized XML data. Even though data sets are not entities in the strict sense, those services are still considered Entity Services for classification purposes.

Users of Entity Services may require that a permission be configured for them in order to use the service, be it at the application, user, or the application-user scope. These permissions may apply restrictions on data access and/or changes at the "row" (entity) or "column" (entity-element) level. An example of "column" level restriction would be that an HR application might have access to both the social security and home address elements of the employee entity while a check-printing service may only have access to the home address element. An example of "row" level restriction would be an expense report application that lets managers see and approve expense reports for employees that report to them, but not for employees who do not report to them.

Error compensation in Entity Services is mostly limited to seeking alternative data sources, if at all. For example, if an Entity Service fails to access a local database it may try to reach out to a remote copy of the database to obtain the information needed. To support system-state consistency, Entity Service may support tightly coupled distributed atomic transactions. Services that support distributed atomic transactions participate in transactions that are flowed to them by callers, and subject any state changes in the underlying data store to the outcome of these distributed atomic transactions. To allow for a lower degree of state-change coupling, Entity Services may provide support for the more loosely coupled Reservation Pattern, either in addition to or instead of supporting distributed atomic transactions.

Entity Services are often built in-house as a wrapper over an existing database. These services are typically implemented by writing code to map database records to entities and exposing them on a service interface, or by using a software factory to generate the mapping code and service interface. The Web Services Software Factory from Microsoft's Patterns & Practices group is an example of such a software factory. In some cases, the database (Microsoft's SQL Server 2005, for example) or data-centric application (SAP, for instance) will natively provide facilities that enable access to the data through a service interface, eliminating the need to generate and maintain

a separate Entity Service.

Entity Services are often used in more than one composite application and thus they are typically centrally managed. Table 3 summarizes the characteristics of Entity Services.

Entity Services	
Main purpose	Expose and manage business entities
Interface	Entity-level CRUD and domain specific operations
State management	Managing application state is the primary purpose of the service
Transactions	Atomic and/or Reservation Pattern
Error handling	Limited error compensation (affects SLE and SLA)
Security	User/App/Both
Management/Governance	Centralized
How built	In-house/Purchased/Leased

Table 3. Summary of the Entity Services category

Capability Services

Capability Services implement the business-level capabilities of the organization, and represent the action-centric building blocks (or "atomic verbs") that make up the organization's business processes. A few examples of Capability Services include third-party interfacing services such as a credit card processing service that can be used for communication with an external payment gateway in any composite application where payments are made by credit card, or a value-add building block like a rating service that can process and calculate user ratings for anything that can be rated (usefulness of a help page, a book, a vendor, and so forth) in any application that utilizes ratings. Capability Services can be further divided by the type of service that they provide (for example, third-party interfacing or value-add building block), but this further distinction is out of scope for this discussion.

Capability Services expose a service interface specific to the capability they represent. In some cases, an existing (legacy) or newly acquired business capability may not comply with the organization's way of exposing capabilities as services, or even may not expose a service interface at all. In these cases, the capability is typically wrapped with a thin service layer that exposes the capability's API using a service interface that adheres to the organization's way of exposing capabilities. For example, some credit-card processing-service companies present an HTML-based API that requires the user to complete a Web-based form. A capability like that would be wrapped by an in-house-created-and-managed-façade service that will provide easy programmatic access to the capability. The façade service is opaque and masks the actual nature of the capability that's behind it, to the point where the underlying capability can be replaced without changing the service interface used to access it. Therefore, the façade service is considered to be the Capability Service, and the underlying capability becomes merely an implementation detail of the façade service.

Capability Services typically do not directly manage application state; to make state changes in the application, they utilize Entity Services. If a Capability Service does manage state, that state is typically transient and lasts for a duration of time that is shorter than the time needed to complete the business process that this Capability Service partakes in. For example, a Capability Service that provides package shipping price quotes might record the fact that requests for quotes were sent to the shipping providers until the responses come back, thereafter erasing that record. In addition, a Capability Service that is implemented as a workflow will manage the durable, transient execution state for all the currently running instances of that workflow. While most of the capabilities are "stateless," there are, of course, capabilities such as event logging that naturally manage and encapsulate state.

Users of Capability Services may require that a permission be configured for them in order to use the service, be it at the application, user, or the application-user scope. Access to a Capability Service is typically granted at the application level. Per-user permissions are typically managed by the Process Services that make use of the Capability Services to simplify access management and prevent midprocess access failures.

Error compensation in Capability Services is limited to the scope of meeting the capability's Service-Level Expectation (SLE) and Service-Level Agreements (SLA). For example, a Shipping Service that usually compares the rates and delivery times of four vendors (FedEx, UPS, DHL, and a local in-town courier service, for example) may compensate for a vendor's unavailability by ignoring the failure and continuing with the comparison of the rates that it was able to secure as long as it received at least 2 quotes. This example illustrates that compensation may result in lowered performance. This degradation can be expressed in terms of latency, quality of the service, and many other aspects, and therefore it needs to be described in the SLE and SLA for the service.

Capability Services may support distributed atomic transactions and/or the Reservation Pattern. Most Capability Services do not manage resources whose state needs to be managed using atomic transactions, but a Capability Service may flow an atomic transaction in which it is included to the Entity Services that it uses. Capability Services are also used to implement a Reservation Pattern over Entity Services that do not support that pattern, and to a much lesser extent over other Capability Services that do not support that pattern.

Capability Services can be developed and managed in-house, purchased from a third party and managed in-house, or "leased" from an external vendor and consumed as SaaS that is externally developed, maintained, and managed.

When developed in-house, Capability Services may be implemented using imperative code or a declarative workflow. If implemented as a workflow, a Capability Service may be modeled as a short-running (atomic, nonepisodic) business activity. Long-running business activities, in which things may fail or require compensation, often fall into the Process Service category.

A Capability Service is almost always used by multiple composite applications, and is thus typically centrally managed. Table 4 summarizes the characteristics of Capability Services.

Capability Services	
Main purpose	Implement a generic value-add business capability
Interface	Service interface for exposing main functionality
State management	Typically, holds no application-specific state

Transactions	No state implies there is no need for transactions; may implement atomic and/or Reservation Pattern over an Entity service
Error handling	Limited error compensation (affects SLE and SLA)
Security	Application-level
Management/Governance	Centralized
How built	In-house/Purchased/Leased

Table 4. Summary of the Capability Services category

Activity Services

Activity Services implement the business-level capabilities or some other action-centric business-logic elements ("building blocks") that are unique to a particular application. The main difference between Activity Services and Capability Services is the scope in which they are used. While Capability Services are an organizational resource, Activity Services are used in a much smaller scope, such as a single composite application or a single solution (comprising of several applications). Over the course of time and with enough reuse across the organization, an Activity Service may evolve into a Capability Service.

Activity Services are typically created to facilitate the decomposition of a complicated process or to enable reuse of a particular unit-of-functionality in several places in a particular Process Service or even across different Process Services in the application. The forces driving the creation of Activity Services can stem from a variety of sources, such as organizational forces, security requirements, regulatory requirements, and so forth. An example of an Activity Service that is created in a decomposition scenario is a vacation eligibility confirmation service which, due to security requirements, separates a particular part of a vacation authorization application's behavior in order for that part to run behind the HR department's firewall and access the HR department's protected databases to validate vacation eligibility. An example of an Activity Service used for sharing functionality would be a blacklist service that provides information on a customer's blacklist status, so that this information can be used by several Process Services within a solution.

Like Capability Services, Activity Services expose a service interface specific to the capability they implement. It is possible for an Activity Service to wrap an existing unit of functionality, especially in transition cases where an existing system with existing implemented functionality is being updated to or included in an SOA-based solution.

Like Capability Services, Activity Services typically do not manage application state directly; if they do manage state, that state is transient and exists for a period of time that is shorter than the life span of the business process that the service partakes in. However, because of their slightly larger granularity—and because, in some cases, Activity Services are used to wrap an existing system—it is more likely that an Activity Service will manage and encapsulate application state.

Users of Activity Services may require a permission to be configured for them in order to use the service, be it at the application, user, or the application-user scope. As in the case of Capability Services, access to an Activity Service is typically granted at the application level and managed for each user by the Process Services that are using the Activity Service.

Activity Services have the same characteristics for error compensation and transaction support as Capability Services. Activity Services are typically developed and managed in-house, and may be implemented using

imperative code or a declarative workflow. As in the case of a Capability Service, if implemented as a workflow an Activity Service may be modeled as a short-running business-activity.

Activity Services are typically used by a single application or solution and, therefore, are typically managed individually (for example, at a departmental level). If an Activity Service evolves into a Capability Service, the management of the service typically transitions to a central management facility. Table 5 summarizes the characteristics of Activity Services.

Activity Services	
Main purpose	Implement a specific application-level business capability
Interface	Service interface for exposing main functionality
State management	Typically, holds no application-specific state (unless when wrapping and exposing an existing system)
Transactions	May support atomic transactions and/or implement the Reservation Pattern over an existing system
Error handling	Limited error compensation (affects SLE and SLA)
Security	Application-level
Management/Governance	Per application
How built	In-house

Table 5. Summary of the Activity Services category

Process Services

Process Services tie together the data-centric and action-centric building blocks to implement the business processes of the organization. They compose the functionality offered by Activity Services, Capability Services, and Entity Services and tie them together with business logic that lives inside the Process Service to create the blueprint that defines the operation of the business. An example of a Process Service is a purchase order processing service that receives a purchase order, verifies it, checks the customer blacklist service to make sure that the customer is okay to work with, checks the customer's credit with the credit verification service, adds the order to the order-list managed by the orders (entity) service, reserves the goods with the inventory (entity) service, secures the payment via the payment processing service, confirms the reservation made with the inventory (entity) service, schedules the shipment with the shipping service, notifies the customer of the successful completion of the order and the ETA of the goods via the e-mail gateway service, and finally marks the order as completed in the order list.

Process Services may be composed into the workflows of other Process Services, but they will not be recategorized as capability or Activity Services, because of their scope and long-running nature.

Because Process Services implement the business processes of the organization, they are often fronted with a

user interface that initiates, controls, and monitors the process. The service interface that these services expose is typically geared towards consumption by an end-user application, and provides the right level of granularity required to satisfy the use cases that the user facing front-end implements. Monitoring the business process will at times require a separate monitoring interface that exposes BAM information. For example, the order processing service may report the number of pending, in-process, and completed orders, and some statistical information about them (median time spent processing and order, average order size, and so on).

Process Services typically manage the application state related to a particular process for the duration of that process. For example, the purchase order processing service will manage the state of the order until it completes. In addition, a Process Service will maintain and track the current step in the business process. For example, a Process Service implemented as a workflow will hold the execution state for all the currently running workflow instances.

Users of Process Services may require that a permission be configured for them in order to use the service. Access to a Process Service is typically granted at the user level.

Process Services very rarely support participation in a distributed atomic transaction, because they provide support for long-running business activities (long-running transactions) where error compensation happens at the business-logic level and compensation may involve human workflows. Process Services may utilize distributed atomic transactions when calling into the services they use. They may also implement the reservation pattern.

Process Services are typically developed and managed in-house, because they capture the value-add essence of the organization, the "secret sauce" that defines the way in which the organization does its business. Process Services are designed to enable process agility (that is, to be easily updatable), and the processes that they implement are typically episodic in nature (the execution consists of short bursts of activity spaced by long waits for external activities to complete). Therefore, Process Services are best implemented as declarative workflows using a workflow server (such as Microsoft's BizTalk Server) or a workflow framework (such as Microsoft's Windows Workflow Foundation).

Process Services are typically used by a single application and, therefore, are managed individually (for example, at a departmental level). In some cases, a reusable business process may become a commodity that can be offered or consumed as SaaS. Table 6 summarizes the characteristics of Process Services.

Process Services	
Main purpose	Implement a business process by orchestrating other services
Interface	Service interface targeted at user-facing applications
State management	Manages process state
Transactions	No support for atomic transactions; may use atomic transactions with the services it uses; may implement a Reservation Pattern
Error handling	Implemented as part of the business logic
Security	User

Management/Governance	Per application
How built	In-house

Table 6. Summary of the Process Services category

Conclusion

For the architect, having a good grasp of the different categories assists in classifying existing or new services, as well as helps define the appropriate functionality to include in a particular service in order to promote composition and reuse. The architectural blueprint defined here can be used for the design of new systems, as well as the refactoring of existing systems.

For the business-decision maker, understanding the business value of a component and its commoditization level makes it easier to make build-versus-buy-versus-lease decisions, and may expose business opportunities for making a service available for others.

About the author

Shy Cohen is a program manager in the Distributed Systems Group at Microsoft. Shy joined Microsoft in 1996 and has worked on different technologies at different groups in the company, always staying true to his decade-long passion for distributed computing.

For the past five years, Shy has focused his time on designing several of the technical and architectural aspects of Windows Communication Foundation (WCF). Today, he directs his attention to several aspects relating to the creation of distributed systems and provides guidance on distributed systems, SOA, Web Services and WCF, and workflow technologies.

This article was published in the Architecture Journal, a print and online publication produced by Microsoft. For more articles from this publication, please visit the Architecture Journal Web site.