Service Oriented Architecture -
Creating an Enterprise Service Mosaic

Businesses across the globe are making a conscious effort to align IT with their business needs. Towards this end, they have embarked on service oriented architecture for building services around business processes and IT infrastructure to realize a plug-n-play framework of “business capabilities”.

In this paper we detail out a framework which we first referred to as Service Mosaic in our paper “SOA – An Enterprise Perspective”. Service Mosaic forms the enterprise backbone for new business processes to optimize existing ones. The paper outlines the architecture principles and the approach for identifying services at different levels in an enterprise and realizing them using a standards-based approach for consistency and interoperability.
About the Authors

Harish Iyer
Harish Iyer is a senior solution architect with TCS who has been involved in architecting solutions on various platforms for customers in the Airlines and Travel domains. He holds a Bachelors degree in engineering from Mumbai University.

Robin Reuben
Robin Reuben is a senior solution architect with TCS and has been associated with many enterprise-wide SOA initiatives in the Airline and Travel domains.

Sachin Patil
Sachin Patil is a solution architect within the EAI practice focusing on IBM integration technologies and SOA. He holds a Masters degree in technology from Indian Institute of Technology, Mumbai. Sachin has been involved in architecting solutions on various platforms for customers in the financial services and telecom domains.

Sandeep Karkera
Sandeep heads the IBM center of excellence across the EAI, BI, and KM practices where SOA is one of the focus areas. He holds a Bachelors degree in engineering from Mumbai University and has been associated with many enterprise programs across various verticals including financial services and telecom.
# ACRONYMS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BAM</td>
<td>Business Activity Monitoring</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>BPM</td>
<td>Business Process Management</td>
</tr>
<tr>
<td>BU</td>
<td>Business Unit</td>
</tr>
<tr>
<td>CFO</td>
<td>Chief Financial Officer</td>
</tr>
<tr>
<td>CIO</td>
<td>Chief Information Officer</td>
</tr>
<tr>
<td>COO</td>
<td>Chief Operating Officer</td>
</tr>
<tr>
<td>CTO</td>
<td>Chief Technology Officer</td>
</tr>
<tr>
<td>EDMG</td>
<td>Enterprise Data Management Group</td>
</tr>
<tr>
<td>ESO</td>
<td>Enterprise Service Operations</td>
</tr>
<tr>
<td>ICC</td>
<td>Integration Competency Centre</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>PCB</td>
<td>Program Control Board</td>
</tr>
<tr>
<td>PEG</td>
<td>Process Engineering Group</td>
</tr>
<tr>
<td>SAG</td>
<td>Service Architecture Group</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SME</td>
<td>Subject Matter Expert</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SOBET</td>
<td>Service Oriented Business Engineering Team</td>
</tr>
<tr>
<td>SSC</td>
<td>SOA Steering Committee</td>
</tr>
<tr>
<td>WS</td>
<td>Web Services</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Definition Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>Extensible Stylesheet Language Transformations</td>
</tr>
</tbody>
</table>
# Table of Contents

1. Introduction ........................................ 3  
2. Service Mosaic Definition And Design Principles.......... 4  
3. Building A Service Mosaic ................................ 10  
4. Service Mosaic Governance .................................. 26  
5. Service Mosaic And The Value Measurement Framework .... 29  
6. Service Mosaic Usage And Optimization ....................... 33  
7. Service Mosaic Realization Strategy ......................... 35  
8. Conclusion .................................................. 37
Introduction

TCS believes that any solution proposed to its customers should achieve the strategic goals and objectives set by the business. We discussed in our earlier whitepaper “Service Oriented Architecture – An Enterprise Perspective”, that SOA is a key enabler for such solutions. Service-Orientation entails an enterprise-scale business transformation program with a vision to create a Self-Optimizing Enterprise that has superior ability to sense and respond to market opportunities with Agility (responsiveness), Efficiency (cost containment), and Resilience (adaptability).

SOA enables a plug-and-play framework of technology enabled business capabilities that form the building blocks for enterprise solutions. It enables the enterprises to leverage their key business assets within and beyond the enterprise boundaries with true interoperability independent of platforms, languages, and protocols.

The above referenced paper introduced the concept of a Service Mosaic that represents these diverse business capabilities and technology infrastructure that enables the business to compose custom applications (composite applications) and business processes based on their changing needs by orchestrating the business capabilities available in the Enterprise Service Mosaic.

This paper explores this idea further focusing on the following aspects of creating a Service Mosaic in an organization.

Section 2 Service Mosaic Definition and Design Principles – Explains the definition of service mosaic, its role in an organization, and guiding principles for designing the key component of the Service Mosaic viz. individual services.

Section 3 Building a Service Mosaic – Describes an approach to identifying services and realizing the same using different technologies.

Section 4 Service Mosaic Governance – Discusses how services are governed and managed in an enterprise and the team structure for administering the services.

Section 5 Service Mosaic and the Value Measurement Framework – Describes a value measurement framework to identify the business value of the Service Mosaic to an enterprise.

Section 6 Service Mosaic Usage and Optimization – Highlights some of the guiding principles on optimizing the service mosaic and effective utilization of the mosaic in the enterprise.

Section 7 Service Mosaic Realization Strategy – Discusses the different entry points in an engagement and how we realize the service mosaic in each of those entry points with case studies.
Service Mosaic Definition And Design Principles

The Random House Unabridged Dictionary defines the word service and mosaic as indicated below.

Service – an act of helpful activity; help; aid
Mosaic – composed of a combination of diverse elements

Thus a service is something that performs a task or function and a mosaic is a collection of a diverse set of elements. This resulted in the coinage of the phrase ‘Service Mosaic’ that can be defined as the entire portfolio of services and related infrastructure within an organization that is available internally within the organization to reuse or orchestrate into business processes or that can be exposed externally to other organizations seeking to interface with the former.

At TCS, we feel that building the Service Mosaic is the most important step in using SOA to bring value to an enterprise. The Service Mosaic in an enterprise serves both IT as well as businesses. IT can use the infrastructure and application services in the mosaic for plumbing and assembling into other services, while the business can use the business and process services to choreograph other business processes. The process of creation of service mosaic itself is the culmination of collaboration between business and IT.

This process of building a Service Mosaic (or Service Portfolio) consists of the steps indicated below:

- Service Identification – This entails identifying the business capabilities that are the key priorities of the enterprise and taking these further to identify services that will support the business capabilities.
Service Definition – Once the services are identified, the meta-data that will define and expose the service needs to be defined in conjunction with the enterprise standards and conventions. This includes the contracts and policies that will govern the run time lifecycle of the service.

Service Realization – This entails development and implementation of the service using tools and techniques available.

But first, we will start with defining a service and classifying different types of services that will form the service mosaic.

Defining a Service
We have already seen the dictionary meaning of ‘service’ in the earlier section. From an IT perspective, a service is an encapsulated piece of functionality which can be invoked and reused by various clients. In SOA context, this invocation is consistent across technologies and protocols.

The concept of a service is the next step in the evolution of the software programming paradigm from modular to object oriented to component based. Each step in this evolution brought about increasing levels of flexibility, reusability, and maintainability in software. The key considerations in the earlier approaches i.e. modular, object-oriented, and component based were the reuse of code fragments, methods and thus producing maintainable code. With the service oriented approach, the movement is more about defining reusable business components and their implementation using platform independent techniques such that they can be assembled or orchestrated into composite business processes.

Types of Service
Now that we have defined a service, let us look at the different types of service. Services can be classified on the basis of the function they perform and the granularity and reuse that is achieved from the service. Based on this, we define the following types of services.

Infrastructure Services
These are the lowest level services in an enterprise and can be of different sub-types namely:

- Services performing an application infrastructure task such as logging, authentication, auditing, data transformation etc.
- Services executing common business rules or logic services
- Services facilitating interaction between different systems such as adapters
- Services manipulating data from a common reusable store including metadata

These services can be typically reused across different applications in an enterprise and the level of reuse here is the highest. An attribute of infrastructure services is that they are widely accepted and taken for granted and are not expected to change frequently.

Application Services
These are one level higher than the infrastructure services and may internally reuse various infrastructure services. These are typically business functions exposed out of individual applications e.g. customer services from a CRM application, billing services from a billing system, and so on. These services can be reused by business services and different applications within the enterprise.

Business Services
These services perform a significant business task and may internally reuse multiple application services and infrastructure services to implement the same. An example of this, is a customer registration business service which may internally invoke a CRM registration service, a billing system service, and another transactional system to accomplish the complete registration. Business services can also involve core processing logic e.g. computation of insurance premium for a given set of policy attributes.
While doing a service identification, the high level process services and the involved business services (activities or steps in the process) are conceptualized based on business requirements whereas the implementation will translate these to existing applications (application services or infrastructure services). In simplified application scenarios, a business service functionality may be implemented completely by one application service and there would be a one-to-one mapping between the business and application service. In such cases, the business service may not be physically implemented and the functionality may be realized directly by the application service.

**Process Services**
These perform high level business tasks and are typically implemented as a choreography of various services (business, application and infrastructure). These services are at the highest level in an enterprise and the reuse of services here may be minimal. But these are the services that bring agility and adaptability to an enterprise by supporting business changes with run time/design time alterations in the service choreography. The realization of these services may also involve manual steps intermingled with automated workflows. An example of this service is the flight ticket reservation process explained in more detail later in this paper.

Having described the four types of services above, we would like to highlight some interesting thoughts on these.

Over time, as an industry or enterprise matures, its business services may become infrastructure services. For example, the payment gateway is a very widely accepted, standards based, and mature functionality. Hence, today, the payment gateway service has become a ‘taken for granted’ infrastructure service. Similarly within an enterprise, over time, some services become very mature and change very infrequently. Such services get identified as infrastructure services.

The definition of an application service is based on the existence of pre-packaged or custom developed applications. As an organization matures along its SOA path or in a completely ground up SOA implementation, there would not be any application services. The Service Mosaic would consist of a collection of process services which would in turn use business services and these business services would in turn use other business services or infrastructure services. Hence instead of existing applications exposing application services, we would have a logical application using process and business services from the mosaic.

Another term that is much used in the industry today is ‘composite’. We define composite as a complex business or infrastructure service which is an assembly of other business or infrastructure services. Thus a composite is an additional qualifier to a business or infrastructure service based on the nature of its implementation. This is different from a process service which is an end-to-end functionality achieved by orchestrating together a set of services in a workflow. Another difference is that a process service can be long-running with human intervention while a composite service will always be an atomic functionality without any human intervention or lag time.

**Design Principles of a Service Mosaic**
The key principles on which the Service Mosaic Design is based is discussed in this section. Also discussed are the situations where the principles may not necessarily apply. In subsequent sections, we will see how the various design principles are realized in the build process of a Service Mosaic.

**Services should be Reusable**
This is the basic guiding principle for the design of a service. As described in the beginning of this paper, service is an evolution in thought from modular to object oriented to service based. Services should encapsulate a piece of functionality which can be reused across multiple business functions. This ensures less redundancy in the application, ease of maintenance, and reduced development efforts for new business functions.
To ensure that services are reusable, they should be designed in a generic and configurable manner so that they can cater to a wide variety of scenarios. For example, use of business rule engines in the implementation of a service or the abstracting hard-coded parameters into configuration files. The payment gateway example used earlier is a good example of a reusable service. The same payment gateway service is invoked for payments to be made via credit cards (VISA, MASTER, AMEX etc.), debit cards from various banks, direct debit from various banks etc. This is a completely commoditized service and any application across the globe which needs this functionality can directly use this service without having to incur any major development.

Exceptions
The reusability index can be different for different services, for e.g. infrastructure services may be the most reusable across the enterprise, some business services may be more reusable than others and some very specific business services may be used only once. The process services explained earlier are another case where the reuse may not be very high. But, irrespective of the current envisaged reuse of a service, the service should be designed and implemented in the manner described above so that future reuse can be facilitated.

Services should have a Well Defined Contract
This is an extension of the standard OO paradigm of programming to an interface. Service contracts provide a formal definition of:

- The service endpoint
- Each service operation
- Every input and output message supported by each operation
- Rules and characteristics of the service and its operations
- Performance and SLA considerations

Service contracts therefore define almost all of the primary parts of an SOA. Good service contracts may also provide semantic information that explains how a service may go about accomplishing a particular task. Either way, this information establishes the agreement made by a service provider and its service requestors.

Because this contract is shared among services, its design is extremely important. Service requestors that agree to this contract can become dependent on its definition. Therefore, contracts need to be carefully maintained and versioned after their initial release.

Within the web services framework, service description documents (such as the WSDL definition, XSD schemas, and policies) can be viewed collectively as a communications contract that expresses exactly how a service can be programmatically accessed.

Exceptions
There is probably no complete exception to the above principle. The rigor in defining the contract can be different for different situations. For a small internal application the rigor can be much less as opposed to that required for a cross-enterprise service. Highly reused services should have well thought of interfaces as they get invoked from multiple consumers and may have different variations of inputs. Taking the same example of the payment gateway service, any slight change in the interface contract definition for this service will impact a huge number of services across the globe. Hence the interface for such services should be very well thought out.

Services should be Loosely Coupled
Loose coupling is the phrase used to describe the impact of change in a service on the service consumers. No one can predict how an IT environment will evolve. How automation solutions grow, integrate, or are replaced over time can never be accurately planned out because the requirements that drive these changes
are almost always external to the IT environment. Being able to ultimately respond to unforeseen changes in an efficient manner, is a key goal of applying service-orientation. Realizing this form of agility is directly supported by establishing a loosely coupled relationship between services.

Loose coupling is a condition where a service acquires knowledge of another service while remaining independent of that service. Loose coupling is achieved through the use of service contracts (service descriptions) that allow services to interact within predefined parameters; this is similar to the age old concept of ‘programming to an interface’. Another important aspect of loose coupling is the design of services to be sufficiently coarse grained so as to encapsulate a cohesive amount of functionality within the service. If multiple service consumers have to make multiple fine grained calls to a service provider for achieving an atomic functionality, the consumers have built a dependence on the provider by virtue of their invocation of these operations in a specific sequence. On the other hand, fine grained services are more amenable to be reused or composed as opposed to a coarse grained service. Service Definition has to tread the thin line between both these approaches.

Loose coupling is also important from the perspective of platform and transport protocol independence. Service consumer should not be tied down to a particular way of binding to and invoking a service. Service consumers should also not be tied down to a particular platform.

It is interesting to note that within a loosely coupled architecture, service contracts actually tightly couple operations to services. Once a service is formally described as being the location of an operation, other services will depend on that operation-to-service association.

Exceptions
Service implementations today being based on the web services architecture automatically achieve the platform and transport protocol independence described above. But if an enterprise so wishes, for internal applications, it can decide upon a different standard such as Java RMI invocations or JMS based invocations with a custom defined message format etc. The disadvantage is that these applications will not be interoperable with industry standard applications and services. The enterprise also has to expend additional effort in coming up with such standards and this is not advisable.

Services should abstract Underlying Logic
Also referred to as service interface level abstraction, it is this principle that allows services to act as black boxes, hiding their details from the outside world. The scope of logic represented by a service significantly influences the design of its operations and its position within a process. There is no limit to the amount of logic a service can represent. A service may be designed to perform a simple task, or it may be positioned as a gateway to an entire automation solution. There is also no restriction as to the source of application logic a service can draw upon. For example, a single service can, technically, expose application logic from two different systems.

Operation granularity is therefore a primary design consideration that is directly related to the range and nature of functionality being exposed by the service. Again, it is the individual operations that collectively abstract the underlying logic. Services simply act as containers for these operations.

Service interface level abstraction is one of the inherent qualities provided by web services. The loosely coupled communications structure requires that the only piece of knowledge services need to interact with is each others’ service descriptions.

Services should be Composable
A service can represent any range of logic from various types of sources, including other services. The main reason to implement this principle is to ensure that services are designed so that they can participate as effective members of other service compositions, when required. This requirement is irrespective of whether the service itself acts as the composer of others.
A common SOA extension that underlines composability is the concept of orchestration. Here, a service-oriented process (which essentially can be classified as a service composition) is controlled by a parent process service that composes process participants.

The requirement for any service to be composable also places an emphasis on the design of service operations. Composability is simply another form of reuse and therefore operations need to be designed in a standardized manner and with an appropriate level of granularity in order to maximize composability opportunities. Composability makes reuse also easier as new services can be built quickly by assembling existing services without having to write too much glue code.

Exceptions
A service can be classified as non-composable if it has internal dependencies on availability of some resources or other resource constraints (e.g. cannot participate in a global transaction context).

The web services architecture standards and the BPEL4WS standards ensure that services can be composed into processes using intuitive GUIs. This attribute is essential for services which are highly reused and can be sacrificed in case of services which are very peculiar and unique and would not need to be so composed into other services.

Services should be Stateless
Services should minimize the amount of state information they manage and the duration for which they hold it. State information is data specific to a current activity. While a service is processing a message, for example, it is temporarily stateful. If a service is responsible for retaining state for longer periods of time, its ability to remain available to other requestors will be impeded. Statelessness is a preferred condition for services and one that promotes reusability and scalability. In order for a service to retain as little state as possible, its individual operations need to be designed with stateless processing considerations.

The primary quality of SOA that supports statelessness is the use of document style messages. The more intelligence added to a message, the more independent and self-sufficient it remains.

Exceptions
This principle as explained above is essential from the perspective of scaling up to meet large load requirements. But there may be requirements such as long running business processes involving timed interactions with other systems or human intervention which may demand that the state information is preserved by the service throughout the execution cycle spanning hours or days. Advancements in implementation techniques such as persistence mechanisms and activation-passivation mechanisms ensure that such processes are executed in a stateful manner with minimal locking of resources.

Services should be Discoverable
Discovery helps avoid the accidental creation of redundant services or services that implement redundant logic. Because each operation provides a potentially reusable piece of processing logic, metadata attached to a service needs to sufficiently describe not only the service’s overall purpose, but also the functionality offered by its operations.

Note that this service-orientation principle is related to but different from runtime SOA discoverability. On an SOA level, discoverability refers to the architecture’s ability to provide a discovery mechanism, such as a service registry or directory. This effectively becomes a part of the IT infrastructure and can support numerous implementations of SOA. On a service level, the principle of discoverability refers to the design of an individual service so that it is as discoverable as possible.
Building A Service Mosaic

Service Identification
The service identification process is driven primarily by the business processes modeled based on the individual business domain requirements. Once the business functionality is realized, we go about identifying the services, which are inline with the design principles stated in the earlier section.

Technically any piece of functionality can be defined as a service, but doing so will lead to a service proliferation resulting in difficulty to comply with QoS and maintainability. Service should be well organized set of functionality representing a tangible business concept. For example, get Account Balance is a tangible functionality representing a portion of business process but convertStringToNumber is not an identifiable business concept and hence not a candidate for service.

TCS envisages the following three approaches as the key for building a service oriented platform.

- Top-Down
- Bottom-up
- Middle-Out

The above three approaches will be detailed out in terms of identifying services at different layers and keeping in mind some of the key business drivers for a service oriented platform. The end goal is to ensure that the business can measure the value of the platform in terms of improvement in business processes and how it impacts the revenues and the stakeholders in the enterprise directly.

Top-Down Approach
In top-down view, a blueprint of business use case provides the specification of business services. The top-down process if often referred to as domain decomposition consists of decomposition of the business domain into its functional areas and subsystems, including its flow or process decomposition into processes, sub-processes and high level business use cases. These use cases are good candidates for business services exposed at the boundary of the enterprise or can be used across lines of business within the enterprise.

Any enterprise with following objectives can embark on a Top-Down approach for service definition:

- Broad business transformation of existing business models or the deployment of new business model.
- An enterprise architected implementation enabling integration across business functions throughout the enterprise.

Bottom-up Approach
In the bottom-up approach, existing systems are analyzed and candidate business functions are identified as services for realizing the business process defined by the business analyst.

In this approach existing business systems are analyzed to unearth business functionality which can be exposed as services for supporting the business processes. In this process the existing APIs, transaction, and modules are analyzed from existing legacy and packaged applications. In some cases componentization of legacy systems is needed to re-modularize the existing assets for supporting service functionality. This may call for changes in the existing applications to meet the service oriented needs of the enterprise. However the change to these existing applications has to be kept to a minimum so that existing business processes are not impacted.
**Middle-out Approach**

Top down and bottom up approach both have some limitations. In the top down approach services defined at the process layer may not match with services exposed by the existing applications. In the bottom up approach combining smaller services into bigger ones may not rollup to the business functionality as required in the process layer.

This middle-out process consists of a combination of top-down and bottom-up techniques of domain decomposition and existing asset analysis. In the top-down view, a blueprint of business use cases provides the specification for business services.

Identifying services typically requires an organization wide analysis. However, certain analysis patterns can be applied to find out potential services. Here are some aspects to consider when identifying services.

- Analyze a certain part of the enterprise business process and decompose them into several small business processes. For example, Order Processing System of an enterprise can be decomposed into smaller business processes such as Sales order management, Inventory management and Dispatching processes.

- Identify if any of the smaller business processes are reused or potentially can be reused in other business domains of the enterprise. For example, checkInventory() can be used by the warehouse management system to check stock of inventory. The reusable business services are candidates for defining them as services.

- Define the inputs and the output required for the business processes. The key point here is that these parameters should be generic so that services are reusable and they can support changing business needs over a period of time. For example, the processPayment() service may now support payment by check but tomorrow the introduction of other forms of payment like credit card should be easily incorporated into this service.

- Once the business processes are modeled, figure out if this business functionality is already implemented as part of the existing application portfolio. Analyze the business critical factor of all these applications to identify which of this functionality can be converted as business services. This is more to do with platform revitalization of the existing portfolio of applications.

- Identify the services which can be composite services to realize some activities in a business process and the dependencies of these services.

- If services need to be exposed to the external world, then the definition of the services needs to account for how messages can be exchanged and security protocols be followed. More important would be the response time the service takes to respond to any consumer request.

These are some of the broad guidelines based on our past experience with SOA implementation. This is by no means a complete list of guidelines and there are no predefined set of formulas to define and implement a set of services.

This service definition approach can be demonstrated by means of a ticket reservation process in the travel industry. The ticket reservation process has multiple legs before the passenger can print his ticket. There can be potential activities within the reservation process which can be realized by invoking services from end applications or services defined outside the enterprise.
The entire ticket reservation process can be broken down into the following steps:

1. Search Availability (Criteria based on date of travel, sector of travel, and class of travel)
2. Make a Reservation (Confirm the ticket)
3. Make Payment (Mode of payment could be credit, cash, cheque, mileage points)
4. Issue Ticket

**Business Process Definition**

In this step the process models and the existing services are documented along with a refinement in the mapping between business processes and existing services.

For the example cited above, the ticket reservation process is modeled which has four sub-processes viz. check for ticket availability, make the reservation, make the payment, and issue the ticket.

**Decomposing Process Activities into Business Services**

At this stage the sub-processes identified are split into a set of business functions which can be used as business services across business domains. The key factors would be achieving the right granularity of these business services and meeting all the service level requirements in terms of the overall business processes in accordance with the design guidelines spelt out in the earlier section.

In the above example we have identified the search service, confirmation service, payment service and ticket processing service. These business services put together help us realize each of the process activities involved.
**Mapping of Business Services and Existing Application Services**

In this step we map the existing application services to the functionality required of the business services in the business processes. There are four possible options for the mapping results.

- A service is mapped to an activity in the process. It's functionally and the function defined in the activity exactly match with each other.

- A service is mapped to part of an activity. The function defined by the activity covers the service and contains more.

- A service is mapped to more than one activity. Its function covers more than the function described in an activity.

- A service cannot be mapped to any activity in the process.

In this step the focus is to refine the process model and achieve a 1-to-1 mapping between services and the activities in the business process. The rationale is that 1-to-1 mapping can leverage both existing services and they can be easily managed and maintained.

If a 1-to-1 mapping cannot be realized, we try to find n-to-1 mappings i.e. a service may support more than one activity in the process especially if the use of an activity is multiple.

In the above business process composite services like search and confirmation of ticket availability can be directly mapped to the search and confirmation activity of the Ticket reservation process.

**Service Definition**

Having identified the various services, the next important step is to define these services. This is a very important step as future reusability, configurability, and flexibility of services are dependent on how well the service interfaces are designed.

A service definition can be broadly classified into the following four categories:

- Abstract
- Concrete
- Supplementary Description
- Policies

Abstract, also termed as service interface definition, primarily defines the following:

- Service name
- Operations in the service
- Input and output data elements for each operation and their formats

An abstract of the service definition provides sufficient information for a service consumer to be designed to invoke the service. The operation and the input/output definition form the basis of how coarse grained and cohesive a service will be. Inefficiency of the input/output definitions to cater to future variations, will lead to the development of multiple services which have minor variations or lead to modifying the service interface, thus impacting a host of service consumers.

Concrete, also termed as service implementation definition, primarily defines the following:

- Service location in terms of physical address
- Service access protocol e.g. HTTP, RMI etc
- Protocol for passing service data e.g. SOAP
This part of the service definition provides the necessary information to invoke the service in runtime.

The above abstract and concrete definitions help in designing and building services. The services realized can be invoked by the consumer. It is a good practice to represent these definitions in a machine processable form (e.g. XML) so as to achieve automatic service composition and invocation.

A supplementary description can be used to provide further information about the functionality of the service. This is more from the perspective of categorizing and classifying the service and is typically used by prospective service consumers to search for a service. Some examples of such data include documentation of the service; target consumers, specific business rules supported/not supported in the service etc. The following table is a sample service definition template.

<table>
<thead>
<tr>
<th>Name</th>
<th>Manage Appointments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td></td>
</tr>
<tr>
<td>Related Business Process</td>
<td>Work management</td>
</tr>
<tr>
<td>Business Owner</td>
<td></td>
</tr>
<tr>
<td>Design Owner</td>
<td></td>
</tr>
<tr>
<td>High Level Description</td>
<td>This capability allows appointments to be fully managed initially for the provision and repair of Customer products and services</td>
</tr>
<tr>
<td>Scope &amp; Assumptions</td>
<td>Service will provide the appointment facility to customers for repair/maintenance of utility (gas/electricity) connections. This capability will not manage appointments required by sales personnel</td>
</tr>
<tr>
<td>Key Information Entities</td>
<td>Appointment, Customer Contact Person, Place, Customer Order, Customer Problem, Component Order, Service Fault, Job and Task</td>
</tr>
<tr>
<td>Dependencies</td>
<td></td>
</tr>
<tr>
<td>Deployments</td>
<td>The business units where service is deployed</td>
</tr>
<tr>
<td>Issues</td>
<td>Data replication issue</td>
</tr>
<tr>
<td>Description of Event based messages</td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td>public boolean bookAppointment(Appointment appointmentDetails, Customer customerID, CustomerContactPerson contactID, Place address, CustomerOrder customerOrderID, CustomerProblem customerProblemID)</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
</tr>
<tr>
<td>Exposure Methods</td>
<td>Web Service, J2EE</td>
</tr>
<tr>
<td>Message Type</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Performance Statements</td>
<td>Throughput:</td>
</tr>
<tr>
<td></td>
<td>Latency:</td>
</tr>
<tr>
<td></td>
<td>Availability:</td>
</tr>
<tr>
<td></td>
<td>Recovery Time:</td>
</tr>
<tr>
<td>Delivery mechanism</td>
<td>Guaranteed delivery etc</td>
</tr>
<tr>
<td>Validation/Error Handling</td>
<td></td>
</tr>
<tr>
<td>Configurability</td>
<td></td>
</tr>
<tr>
<td>Special Security Considerations</td>
<td></td>
</tr>
<tr>
<td>Service Variants and physical location</td>
<td>Details of the other versions of the service</td>
</tr>
<tr>
<td>References</td>
<td>Details on requirements, design, and testing documentation</td>
</tr>
</tbody>
</table>
Policies are instructions for processing and are used to provide various infrastructural and QoS definitions such as:

- Actions – to encrypt, decrypt requests, check access/signature etc
- Process message events – before request, after request, before response, after response, on error etc
- Process timed events – scheduled instructions
- Calculations which are stored for further usage like SLA management (e.g. to count number of orders processed by a service)
- Response time and Availability SLA

Policy definitions provide visibility to the service consumer on the QoS characteristics of the service (e.g. service will respond within 200 ms) as well as provide information to service management tools on managing adherence to SLAs.

From an implementation perspective, in web services architecture, the abstract and concrete parts of service definitions are published as a WSDL file. The supplementary descriptions are captured as part of a service registry. The policies are also captured within a service lifecycle management tool. These tools provide the flexibility of defining a set of policies independently and then dynamically binding the policies to various services e.g.

- policy X is applicable to all services in production
- policy Y is applicable to all services used by service A

**Service Realization**

Various considerations go into the realization of identified services. The existing business functions in legacy systems can be “wrapped” as services using technology-specific or platform-specific tools or adapters from the vendors. The new services can be implemented on a chosen technology platform like an Enterprise Service Bus, J2EE, .NET, or even existing legacy platforms. The most common way of realizing the services in SOA today is using web services. The various aspects of realizing these services are described below.

**Adopting Standards**

One of the most important aspects of SOA implementation has been the emergence of technology-agnostic industry standards that govern service description (WSDL), and message structure and data (XML, XML Schema, and SOAP).

**XML and XML Schema**

XML is now universally accepted as a data representation format that works with most technology platforms. XML data representation is fundamental to SOA. XML Schemas preserve the validity and integrity of message data and XSLT is used to transform the XML data from one schema to another.

Simple Object Access Protocol (SOAP) - SOAP is now an established standard for exchanging messages in a web services environment. Every SOAP message is packaged within SOAP envelope which consists of a header and message body. Header is a dedicated area to host meta information and practically all WS-* standards are implemented through SOAP header blocks. The actual message body typically consists of XML data. SOA emphasizes on creating self-sufficient and intelligent messages that promote loose coupling and message independence. SOAP headers are capable of containing a variety of supplementary information related to the delivery and processing of message content.

WSDL - WSDL is a standard used to describe web services. A WSDL document provides a definition of the endpoint interface and the physical location of the service. A WSDL consists of abstract and concrete parts that collectively describe a service. An abstract description lists the operations, and the input and output messages for each operation. Each operation is a specific action that can be performed by the service. Abstract description establishes the interface characteristics of a web service without reference to technology used to host the service or transmit messages. Concrete description binds the abstract interface to the physical
address, at which the service can be accessed (URL) with a specific protocol (e.g. HTTP or JMS). WSDL definitions rely on XSD schemas to formalize the structure of input and output messages of a service.

There are other standards, commonly known as WS-* extensions, that are emerging in the arena of web services. These standards are at a varying degree of maturity and address specific challenges in the web services domain which are described in later sections.

**Service Development Approaches**

The services can be realized or implemented in 3 ways:

- Building a new service from scratch
- Composing a service as an orchestration of existing granular services
- Reusing existing business functionality within the organization by developing a wrapper around it or a technology adapter to access that function or application

**Building a New Service**

Developing a new service from scratch may include implementing an off-the-shelf packaged product or developing a custom-build service. Many new packaged solutions provide the business functions as Web Services. These can be directly used to assemble the requisite new business or application service. For building a new custom service, by far J2EE and .NET are the most popular platforms in the distributed world. Both the platforms provide various tools to develop components that encapsulate business processing logic, XML schema definitions, and automatic creation of WSDL files describing the interfaces of the services.

**SOA** is comprised of the following participants:

- **Service Consumer**
- **Service Broker**
- **Service Provider**

Service Broker mediates a Service Request from the consumer and routes it to the actual Service instance. It accepts a request from a consuming application which specifies a logical name for the service and service operation. This information is used to locate a service that has been described in the Service Repository. The Service Manager then delegates the request using an appropriate protocol (e.g. Web Service, JMS etc.) for service implementation. Service Broker functionality is typically implemented using an ESB. Alternatively, a custom Service Manager component can be developed which serves as the broker between the Service Consumer and Service Provider.

---

**Fig 3** Relation between Service Consumer and Service Provider
A Service Provider actually builds the Service. It implements the business functionality of the service just as operations or methods develop the interface to access these methods.

A Service Consumer makes use of the service functionality by invoking the methods. Typically, a Service Consumer has to implement a native client which enables invocation of the service methods. (Note that the client is created in the consumer’s native technology environment which can be different, say .NET, from the environment in which the Service has been implemented, say J2EE)

The most common way to implementing services is through web services. On J2EE platform, web services are typically implemented as servlets or EJBs (stateless session beans). Provider needs to implement Service Endpoint Interface and Service Implementation Bean based on JAX-RPC API that governs SOAP messaging and JAXP API for XML document processing. The runtime environment for hosting the services consists of J2EE Web Container (for servlets) and EJB Container (for EJBs) along with JAX-RPC runtime which provides support for SOAP communication and WSDL processing.

A Service Consumer needs to create a service proxy client to be able to access the service operations. Most IDEs today provide a mechanism to auto-generate the stub from the WSDL provided by the Service Provider at the time of design. The result is a proxy class which can be invoked by the requestor as any other Java component. Dynamic proxy generation at runtime is also supported on most vendor platforms.

.NET framework is another popular platform to implement the services. ASP.NET environment is the web technology layer within .NET environment and Assemblies are the components that encapsulate the business logic. ASP.NET Web Services are typically implemented as ASMX endpoint and a compiled Assembly separately housing the business logic. .NET Web Services primarily use System.XML API for XML processing and System.Web.Servives for SOAP communication and WSDL processing. .NET framework provides a unified support for a set of languages like VB.NET, C++, and C# with a common runtime environment called Common Language Runtime (CLR).

Service Requestor needs to create a .NET proxy class that duplicates a service provider interface. The code behind a proxy class can be auto-generated using Visual Studio or WSDL.exe command line utility.

**Composing a Service**

A Service can also be composed by orchestrating the existing Business Services or Application Services. Process Services are typically implemented this way. With the advent of Business Process Management tools, the existing services can be orchestrated in a specific sequence to implement new processes. This process itself may be exposed as Service and be utilized in other processes. This provides enormous flexibility in creating services at various levels of granularity and based on the service consumers requirements. Various modeling tools are now available to model the business processes.

**Wrapping Legacy Functions**

Most organizations have existing enterprise applications like Legacy Systems (e.g. Mainframe CICS, IMS), ERP Systems (e.g. SAP, Oracle Applications), CRM systems (e.g. Siebel). The existing business functionality in these applications needs to be re-used as far as possible to meet new business requirements. This can be done by “exposing” these business functions as Services. One of the common techniques is to develop or implement a technology adapter that will enable connectivity to these native applications and make available the business functions of these applications through APIs. Conventional EAI platforms made use of various technology and application adapters to connect to a variety of applications. Adapters themselves can be made available as Service on the SOA platform.

Several techniques have evolved to wrap Legacy functions as Services. There are 3 main ways of integrating with legacy applications. We consider an example of Mainframe z/OS environment.
Session integration

Many z/OS applications are only accessible through terminal data streams, typically referred to as “green-screens” and 3270 or 5250 terminals. The applications are written in such a way that business logic and presentation interface are not cleanly separated through the use of callable routines. Re-engineering these applications may involve high risks, time, and money. However, it is still very useful to avail these functions as re-usable services by new applications. Session integration is the ability to intercept and interpret the screen information that is passed back and forth between a client and the server (for example, z/OS, AS/400 and UNIX). The terminal session or screen information can be packaged using different protocols, such as 3270, vt100 and 5250. These protocols describe the data related to the user interface and how that data should be interpreted and rendered by the receiving application (terminal emulator).

With session integration, it is possible to intercept the terminal emulation protocol data and render it in non-traditional ways. For example, the 3270 session data can be displayed in HTML format to be rendered within a web browser. This is more efficient than the “screen scraping” technique, which is tightly coupled to the screen layouts. The client application has a problem interpreting the data correctly if the screen layout is changed. Session integration, on the other hand, can interpret the data received more efficiently via the emulation protocol. Rather than looking at the data in a fixed or positional manner, it actually parses the data and recognizes patterns (or data, fields, constants and screen identifiers) by locating them wherever they are within the data stream. This allows more flexibility in how the data can be manipulated and represented to the consuming application. Session integration can also be used to expose z/OS applications as web services or XML documents.

IBM Host Access Transformation Server (HATS) is one such tool that provides session integration capabilities. It has:

- Rules-based Web-to-Host transformation engine which dynamically converts 3270 screens to HTML pages
- The ability to generate Web services or other Java objects from host transactions
- Server based on WebSphere and 3270 HOD Connector
- HATS Studio based on Rational Application Developer

Though session integration offers value in a number of scenarios, it does have some notable limitations:

- Session integration is possible in “one way” only. In other words, the z/OS applications can be exposed as services; but they cannot participate fully in an SOA by consuming other services that are not on the z/OS platform.
- Session integration can expose and encapsulate only a functionality that exists in the original application. There is no ability to extend the application with this approach.
- Session integration cannot access BATCH applications or processes. It can access only those that are exposed through a user interface or callable routine.
**Transaction Integration**

In some scenarios, z/OS applications may in fact be well structured with separate and distinct presentation logic and business logic. In such cases, the transactions that contain the business logic could be accessed as a web service in an SOA. Transaction integration refers to a style of integration in which existing transactions, such as BATCH programs or online CICS transactions on z/OS, are wrapped as web services without disturbing the original state of the application and can be accessed from distributed platforms. A typical implementation will look as follows:

Transaction integration is also valuable because it offers the potential for “two-way” integration. Instead of only offering z/OS functionality to the outside world, transaction integration allows z/OS applications to consume external services without having to know they are web services. Just as a CICS transaction can be wrapped to look like a web service, a web service can be wrapped to look like a legacy transaction. Following transaction connectivity technologies are available for z/OS regions:

**CICS**
- Direct SOAP access into CICS
- J2EE application connectivity to CICS via J2C (JCA) using CICS Transaction Gateway
- Websphere MQ connectivity to CICS transactions via Bridge or Trigger Monitor

**IMS**
- Direct SOAP access into IMS
- J2EE application connectivity to IMS via J2C using IMS Connect server
- Websphere MQ connectivity to IMS transactions via MQ-IMS Bridge

Sometimes, CICS transactions may be too fine-grained. Tools like CICS Service Flow Feature provide an ability to compose a coarse-grained business service by aggregating a series of CICS transaction calls. A service flow is a non-interruptible micro-flow that is constructed from a collection of nodes that represent the invocation of CICS resources. The flow describes navigation of the nodes and allows data mapping between them. Thus, a single service request may cause the execution of many CICS resources.
Despite the benefits of this approach, transaction integration has its own set of limitations that must be considered:

- Transaction integration assumes that existing applications are well structured with callable subprograms or procedures. If not, these applications must be re-engineered to use this approach.
- Organizations may want to get directly to data sources without having to call existing business logic. Transaction integration allows access only to applications, not data sources.
- Depending on the nature of the original application, services created using transaction integration may be too fine-grained to be of value to an SOA.

**Data Integration**

The distributed applications may be required to directly access the operational and transactional data residing on z/OS databases without going through business logic. This can be achieved in two ways:

- By providing standard connectivity (via ODBC or JDBC) to the z/OS databases and data sources
- By encapsulating SQL statements or stored procedures as Web services.

Again, there are tools, which provide such capabilities. For example, DB2® Web services object runtime framework (WORF) provides an environment to create simple XML based web services that access DB2. WORF uses Apache Simple Object Access Protocol (SOAP) 2.2 or later and the Document Access Definition Extension (DADX). A DADX document specifies a web service using a set of operations that are defined by SQL statements or XML Extender Document Access Definition (DAD) documents. Web services specified in a DADX file are called DADX web services, also referred to as DB2 web services. DADX lives in a J2EE Web application.

![Diagram](image.png)

**Fig 6** Database function (SQL Query, Stored Procedure) exposed as Web Service

There can as well be legacy functions implemented on distributed platforms such as J2EE and .NET. For example, EJB components in an existing J2EE application can encapsulate business functionality. Such components can be wrapped as services by using techniques described in the earlier section (Building a New Service).
Testing Services
When an organization is taking a bottom-up approach to SOA the typical approach is to build web service interfaces to their existing legacy systems. QA is mainly confined to testing these service interfaces. This requires a combination of web services testing and Integration testing capabilities. But both, web services testing and integration testing, concentrate on meeting current requirement specifications. SOA, by nature, is designed for continually changing requirements. Thus QA must also focus on testing SOA metadata because this forms the core of any SOA implementation’s ability to respond dynamically to changing business requirements. The changes that occur during change time are changes in metadata like configuration changes, policy changes, and service contract changes. Thus, testing strategy and tools must provide for the testing of changes to these metadata.

In an SOA environment where business processes are realized through a combination of services, testing needs to be complete in all aspects, functional and non-functional. Capturing all the test cases in such a dynamic environment and dependencies with multiple components/systems makes SOA testing more challenging.

The test strategy for an SOA implementation must cover the different phases of testing like constructional and unit testing, integration testing, and stress testing.

The following sections talks about the different phases of testing in an SOA implementation.

White Box Testing: Testing is done once the construction phase is complete. The objective of this testing is to ensure complete coverage of functionality of the component and optimize performance.

This testing can be carried out using an open standards frameworks like Junit.

Installation Testing: This test is to ensure that the component works on a brand new machine and when there is an upgrade to the application.

Configuration and Interface Testing: Configuration is interface testing helps prove that the service components work on required hardware/software and the protocols.

Functional Testing: Testing of the functionality of the service component ensures the meeting of business requirements, business rules. This testing is based on black box techniques that verify the application and its internal processes by interacting with the application through integrated test stubs, messages and analyzing the output as described in the business process management layer

Data and Database Integrity Testing: This testing verifies all database access methods and checks processes for any kind of data integrity issues when service components update databases.

In Service Oriented Architecture it’s imperative that apart from aligning IT with business the performance of the services meet the required Quality-of-Service. The QoS parameters are response time, throughput, and availability of service. Worst case scenarios are tested to ensure that the services exit out gracefully.

Non-Functional Load Testing: This includes testing the service components under varying workload conditions to ensure compliance with functional and service level requirements.

Non-Functional Testing-Performance Profiling: Performance profiling tests the component for performance behavior under normal and worst case scenarios to ensure compliance with functional and throughput requirements. This test is done to figure out at what loads a components breakdown and this can be reversed by either augmenting the computing resources or making architectural changes.
Non-Functional Testing- Stress Testing: Stress testing, tests the components under stress conditions to see if the component does a graceful exit.

Non-Functional Testing-Volume Testing: Volume testing tests the service components by injecting large volumes of messages and also with database to full capacity of data.

Development Principles of a Services Mosaic
Following is an indicative list of points that need to be considered while developing the services:

**Business Services**
1. Use process choreography for defining business processes in terms of coordinating service calls across several services. This maximizes the visibility and control of business processes and leads to greater enterprise agility.

2. Implement incremental integration of legacy application using adapters, which invoke legacy business functions as services thereby allowing them to participate as service providers, and coordinated by process choreography.

**Services Implementation**
1. Leverage both synchronous and asynchronous web services to decouple different components of an enterprise. This results in greater reuse and enables maximal leverage of common communications architecture.

2. Use a common service model based on web services and WSDL. The services can be accessed over a variety of protocols including SOAP/HTTP/JMS. This helps the enterprise to leverage modern infrastructure, development and operational tooling.

3. Leverage a common infrastructure model to enable process flows to be designed using a common semantic representation of data objects (Canonical Data Model) even though the services accessed in the process use a sub set of the data from the canonical data object. This reduces the number of distinct data transformation that must be defined for services to interact, hence reducing development time. Canonical data objects represent generic entity / message definition based on the commonality across the applications within the enterprise, typically superset of application-specific information. Canonical objects can be based on in-house enterprise standard or based on industry-specific standards. All possible set of consumers and their needs for information must be considered while designing the canonicals.

4. The Enterprise Service Bus provides a common infrastructure for connecting the service providers and consumers with the capability to manage and monitor services against some defined SLAs.

**Business Process Monitoring**
5. A common event model allowing service providers to send anonymous events, which are routed, based on content to appropriate services for processing. This allows for real time business process monitoring against a set of defined KPIs.

**Deployment Considerations For Services**
SOA architects should consider the following as they determine the appropriate platform and language to choose for an individual service:

- **Proximity to the data** – Placing the service close to the source data store or book of record will result in better performance for those services that are data intensive or require frequent interactions with source data.
Proximity to the service consumers – Placing the service close to the service consumers will result in better performance for those services that require frequent interactions between the consumer and the service. This is especially true for those services that are primarily read-only and can benefit from data caching.

Read-only data vs. Dynamic data – Services and source data stores that are read-only or services that do not require real-time access to data can benefit from caching data local to the service. A service’s data cache could be comprised of data previously retrieved or provide “just-in-time” access to frequently accessed data. It does not imply that a service would host a replicated copy of the source data store. Caching service data allows more flexibility for placement of the service as data intensive services can be placed closer to the service consumer.

Scalability requirements - Scalability refers to the capability of expanding capacity available to a service hosted on its target platform without interruption or degradation of the Quality of Service (QoS) to an increasing number of users. Since a successful service implementation is one that is reused by a large number of users, it is important to design, develop and deploy services with scalability in mind. The runtime environment of each service must be capable of scaling to meet the consumers’ needs today, and in future.

Availability requirements – Availability refers to the capability of ensuring a service is operational during the time period specified by an SLA. In the event of a system failure, a service must transparently fail over to a secondary service. Consideration for availability requirements of the service must be given when selecting the platform/runtime environment for a given service.

Systems Management and Monitoring Capabilities – System monitoring services are required for managing capacity and to provide appropriate alerts when thresholds are reached or in event of processing errors. Consideration for the existing systems management capabilities that are available for a given platform must be given when selecting the platform/runtime environment for a given service.

Service Catalog
The key requirement for these services to be used, is the ability for prospective consumers to be able to unearth these services from a common repository through a standard protocol. This is also in line with the design principle on services being Discoverable.

To start with service catalogs were simple web sites where services were described and people could read about the services and their functionality. Subsequently, the need was felt for such discovery to be done programmatically and hence standards based Service Registries evolved for cataloging and publishing service information. Universal Description, Discovery, and Integration (UDDI) is one such registry specification for maintaining standardized directories of information about services.

The key information required for service discovery is:

- Source of Service
- Intended Target Consumers of a Service
- Service Functionality Description
- Service Operations
- Input and Output Data and Formats for Service Operations

As the service usage matured, a need was felt to have more policies defined around the behavior and QoS of services. Service Registries evolved from being a mere publisher of service information to repositories of service information, service contracts, and service policies.

It is important to note that registries today, perform highly sophisticated functions like:
• Routing to the most relevant service, based on various characteristics
• Maintenance of multiple service versions and routing to appropriate version
• Service Policy enforcement in terms of security and QoS
• Change management of services
• Operational and Business reporting on service usage and performance

The service policies and the role played by Service Registries will be discussed in more detail in the subsequent section on Service Governance.

**Challenges in Implementing Services**

There are many challenges while implementing services in an SOA environment in addition to the ones described above. There are various standards and specifications being developed to address these specific challenges. Choice of development and deployment platforms should be based on the specific needs of a particular SOA implementation, which may need addressing one or more of the following challenges.

*Separation of Contract and Implementation*

One of the fundamental tenets of an SOA implementation is that the service requestor should be oblivious to the implementation logic and technology platform of the service. Service provider exposes a standard interface to the external world through which the consumers can invoke the business functionality that is provided by the services. The service provider can also include some metadata that provides additional information about the interface and messages that are expected by the service requester. The service interface and metadata are together referred to as the contract between the consumer and the provider.

*Contract Management and Registry*

As seen above, one service can only access another through the service description. Thus there is a need to store and access the service descriptions in a standardized way. These repositories are known as service registries. UDDI is an acceptable standard to structure a service registry and provides a standard API to programmatically access the service descriptions. Registries can also be searched manually, typically at design time.

*Interoperability*

In a heterogeneous environment where services are defined on different technology stack like Java, .NET etc it's important that these services communicate with each other seamlessly. Web Services Interoperability Organization (WS-I) has released a Basic Profile; a recommendation of the available standards must be used collectively to form the most desirable interoperability architecture. WS-I Basic Profile formally positions specific versions of WSDL, SOAP, UDDI, XML, and XML Schema. Compliance to Basic Profile is gaining industry-wide acceptance including product vendors and it is recommended to be considered while preparing an implementation roadmap and choice of tools and products for SOA implementation.

*Addressing*

In a service oriented solution services can be invoked through different protocols (JMS, HTTP,RMI). When there is a java client requesting for a service it can use the native Java call to invoke the service i.e. RMI or if it's a web based client then it can be HTTP or JMS. These parameters can be addressed as part of WS Addressing standards which identifies the request through the SOAP headers. Addressing empowers a message with the ability to self-direct its payload and allows services receiving the message to respond based on the metadata information within the message. WS-Addressing specification provides SOAP headers, called Message Information (MI) headers, which establish message exchange characteristics within the message itself. MI headers include destination, source endpoint, destination endpoint, reply endpoint, message id, relationship, action etc.
**Reliable Messaging**

In a loosely coupled, message-based environment, a service requestor has no immediate way of knowing whether the message arrived at its intended destination or whether it failed and needs re-transmission or whether the series of messages arrived are in the intended sequence. WS-Reliable Messaging provides a framework which guarantees that a service provider is notified of the success or failure of message transmission and also that messages intended in a particular sequence do so or generate a failure condition. The reliability rules are implemented as SOAP headers within the message itself.

**Correlation**

In a message-based communication, there is no intrinsic way of associating messages as part of a common activity or context. This is achieved by way of correlation. Correlation generally involves embedding a context id, usually original request message id, as part of a message body (typically a SOAP header) in all the related messages. This also helps preserve the service autonomy and statelessness.

**Security**

Security considerations form a very important aspect of a robust SOA implementation especially when cross-enterprise services from business partners are accessed. These considerations include authentication and authorization of the service requestor by the service provider, maintaining confidentiality, and integrity of the messages being exchanged, and non-repudiation of the requests. WS-Security and associated extensions provide a framework to address these security needs in a web services environment.

The messages may need to be secured, either at the transport level or at the message level. SSL is a very popular standard to protect HTTP channel of message transport. However, it provides message security only between the service endpoints. For complete end-to-end security, message-level security needs to be provided. XML-Encryption provides features to encrypt XML messages, either in part (such as password) or as a whole, both header and body. XML-Signature provides features to protect the message from alteration through use of digital signature. XML-Signature can reside in the SOAP header.

Message encryption or transport encryption has an impact on the performance, especially response time and therefore needs to be weighed against the performance requirements.

**Policy Management**

Services can be governed by various policies like response time, privileged customer request (gold, silver etc) which needs appropriate metadata around services cater to such policies. WS-Policy framework governs the structure of policy description and association of policies to the web services. A policy assertion is simply an XML statement that can be programmatically processed. Policies allow better interoperability as more information of service end-point can be published in the contract and thus also help in dynamic discovery and binding of the services. Policies can be accessed programmatically by service requestors to understand the requirements and restrictions of service providers at runtime.

**Transactions Management**

In an enterprise, many a times a group of services (internal as well as external to the enterprise) need to interact to complete a specific task or activity. Individual services interact within a context to complete that activity. The complexity of such an activity or the context information depends on the number of services that need to interact with each other and the duration of the activity. WS-Coordination specification provides a framework to coordinate these services and manage the context of the activity. WS-Coordination framework is based on coordinator services model. Two of the most common coordination types are atomic transactions and business activities or long-running activities. Atomic transactions are coordinated on principles of two-phase commit and rollback whereas long-running activities implement rollbacks by way of compensation transactions.
Service Versioning

The reuse of services across multiple consumers leads to a more complex problem related to versioning of services and release management. A service consumer can be impacted by changes in either the service description (interface) or the end point (implementation). We can hence define the following types of releases based on the complexity of change:

- **Major Releases**: Releases excluding the first release, that require change to interface/endpoint that impacts existing consumers e.g. adding an additional parameter to the input of an existing operation
- **Minor Releases**: Releases that may change interface or endpoint, but the change to interface is limited to addition of some operations which do not impact existing consumers
- **Revisions**: Releases typically involving bug fixes that do not change the end point or interface

While releasing a new version of a service, care has to be taken that the old version of the service is not required by any consumers. It may happen that there could be in-state long running processes which require the old version of the service to be available. Elsewhere different consumers may require different versions of the service depending on other input data parameters e.g. new tax processing rules may be defined for some states while the others continue to use the old rule; in such cases both versions of the tax processing service need to be available and the decision of which service to use is taken on the basis of the input state code. This kind of version related decisions can be implemented as policies within a service repository.

SERVICE MOSAIC GOVERNANCE

Traditionally in an application based IT environment, the applications gets defined by the business units. They are realized using the IT department and the applications get jointly owned by the business unit and the IT department. Any change to the application gets initiated either by the business unit if it is a change due to business requirements or by the IT department if the change is driven by technology requirement.

In a service oriented solution the governance covers design, implementation and hosting of the services to ensure consistent implementation and reliable performance of the services. Establishing an effective governance mechanism across the entire lifecycle and implementing it matters a lot in successful sustenance of a SOA landscape. Starting an SOA initiative is easier than sustaining it. This is due to the reason that as the SOA initiative moves forwards, the common shared services needs to be owned, maintained and made available for reuse across business units and needs to be controlled. A delicate balance needs to be struck between a tight control of the services and the right degree of flexibility to accommodate changes to services as required by consumers belonging to different business units. This makes it imperative for having an established governance process in place that spans across the service lifecycle.

The scope of service governance includes managing the set of commonly shared, reusable services and related assets through their entire lifecycle. Service governance encompasses two distinct phases of the service namely design time governance and runtime governance.

- **Design time governance** of the service will include service identification, definition, realization and publication of service details for usage.
- **Runtime governance** includes access control to the service instance for consumption, monitoring service consumption and other business reporting of the service.

Service Governance Organization

In order to realize effective service governance, each role must be realized through separate well defined teams. The Figure 1 below represents a typical organization for service governance.
**Service Architecture Team (SAG)**

This team will comprise of both technology architects as well as business architects. This team will be the leadership team for Service Governance. This team will set the technology and business strategies, define standards and guidelines, build the enterprise data model, define common business semantics and own the service catalog. As explained in section 2, the enterprise functions are represented as a Service Mosaic. This is realized as a service catalog that contains the list of services, usage scenarios, business functionality, access policy and other service metadata.

**Service Oriented Business Engineering Team (SOBET)**

SOBET will be responsible for defining new services or making changes to the existing services. This will comprise of people drawn out of the business and IT. The internal organization of this team should be around business domain. As discussed in the section on service identification, the services needs to be grouped based on the business domain and owned by a single person or team. This will ensure that the services are consistent and address all business requirements expected from the service. All changes to the services are channeled through the owner who will then approve and build into the existing service implementation.

**Program Management**

Program Management team will do the program management for the implementation of new services or changes to existing services. This team will comprise:

- Project management people who will manage the IT Delivery resources, schedule changes, prioritize work on services
- Release Management people who schedule production/test releases in discussion with the business/IT
- Test Management people who do the test strategy for services/release comprising of multiple services (new and old) and test the services during the testing phases of service realization
Service Delivery
IT Delivery team will comprise of the actual service realization team made up of technology designers, developers, and deployers.

Service Governance in Action
Once the service governance team and processes are in place it is critical that the governance process works as planned to effectively sustain a SOA organization. This section describes the information flow in such an organization assuming the governance organization described in Figure 7 is in place.

- The SAG defines the architecture standards, enterprise data model, data semantics and design guidelines.
- The requirements for new functionality or changes to the existing functionality (both business and IT) come from either the business or IT to the SOBET.
- The SOBET decides on the creation of new service/changes to service. If it is a new service then the business requirement might need to be refined to make it a generic reusable business service which can be shared in the enterprise. This team also defines the service interfaces, analyses the interdependency between business services, impact of changes to a service etc.
- The SOBET draws out the design specification or the service change specification and passes it to the Program Management team.
- The Program Management team creates the work item, allocates work based on prioritization requirements, does the release plan, and test plan.
- The IT Delivery team then does the actual service creation or change to service and deploys them for use in the enterprise.

SOA being a highly distributed, heterogeneous and dynamic it is very important that SOA artifacts are governed by business, technical and regulatory policies. An SOA policy defines configurable rules and policies that effect services both during design and runtime. This means that the policies must be used to validate services before they are published, and as a basis for enforcing standards and behavior at runtime. This linkage between service and policy ensures automatic validation and enforcement of services both during design and run time environment.

Design Time Governance
Design time governance primarily involves two aspects – Service Metadata Management and Service Life-cycle Management.

<table>
<thead>
<tr>
<th>SERVICE POLICIES</th>
<th>POLICY DEFINITION</th>
<th>POLICY ASSOCIATION</th>
<th>POLICY ENFORCEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA Policies will be defined by the SAG team</td>
<td>Reusable policy definition which is independent of the application logic and can be stored in a system of record like a service repository.</td>
<td>Policies can be associated with business services used to realize business processes. Once the policies are published they are associated with business services which render specific functionality.</td>
<td>This is done both during design time and run time using service repository or some web services management component like monitoring, logging, and SLAs of services.</td>
</tr>
</tbody>
</table>

- Service Metadata Management
Service metadata consists of business information like business focus (banking, insurance) and service type (claims processing), technical information like input parameters, protocols, authentication etc. and finally governance information like policies and contracts described below along with their relationships and dependencies.
**Lifecycle Management**

Business value of a SOA implementation is related to the quality and predictability of the services. This can be achieved by managing services and their artifacts in a complete lifecycle. The key activities in a SOA lifecycle management are as follows:

- Providing a means for consumer to discover and reuse services and other artifacts
- Ensuring the quality, performance and applicability of services that are published
- Assessing and managing the impact of change across a network of consumers
- Managing versions and state change of services and other artifacts in a SOA implementation

**Run Time Governance**

The infrastructure management framework service governance helps deliver the business quality of services by monitoring and managing the workload of service requests from the businesses.

Run time governance also includes an understanding of workload performance and resource performance characteristics – along with the ability to forecast the impact of workload or resource changes.

Run time governance entails provisioning, virtualization and resource management at run time to ensure optimal utilization of resources and execution of services in a SOA implementation.

**Service Mosaic And The Value Measurement Framework**

The key aspirations for Service Orientation and Process Management in an enterprise are defined as follows:

- **Agility** – The responsiveness of enterprise and the IT solutions to emerging opportunities and challenges
- **Efficiency** – The cost reductions due to optimized use of infrastructure including hardware and software solutions
- **Adaptability** – The resilience to change in delivering business centric technology solutions so that risks associated with change impacts are well managed

Applying these three aspirations to the three perspectives of business, technology and infrastructure, TCS has defined a value measurement framework to measure, monitor and report the value derived out of the Service Mosaic. The matrix below describes the expectations, capabilities and measures for each dimension of the value measurement framework for e.g. under the business-agility cell, ‘expectation’ defines what is expected by the business from an agility perspective from Service Orientation; ‘capabilities’ defines the capabilities that are needed to realize the expectations and the ‘value measures’ define the attributes that need to be measured to derive and understand the value of service orientation.
<table>
<thead>
<tr>
<th>Business</th>
<th>Technology</th>
<th>Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Agility</strong></td>
<td><strong>Expectations</strong></td>
<td><strong>Expectations</strong></td>
</tr>
<tr>
<td></td>
<td>Responsiveness to Business Needs</td>
<td>plug-n-play service capabilities – to meet the responsiveness need of the business, technology should provide plug-n-play ability</td>
</tr>
<tr>
<td></td>
<td><strong>Capabilities</strong></td>
<td>Qualities-of-Services (QoS) Assurance – to meet the responsiveness need of the business infrastructure should ensure that services are available and perform as per needs</td>
</tr>
<tr>
<td>1. Faster adoption of changes to business model</td>
<td>1. Faster adoption of new business &amp; process initiatives</td>
<td>1. Service’s QoS Monitoring and Management (the infrastructure should support QoS defining, monitoring and managing e.g. standards such as WS-*)</td>
</tr>
<tr>
<td>2. Faster adoption of new business &amp; process initiatives</td>
<td>3. Optimize Service Time to Customers</td>
<td>2. Ability to deploy services anywhere - location independence</td>
</tr>
<tr>
<td>3. Optimize Service Time to Customers</td>
<td>4. Optimize cycle time of internal business processes</td>
<td>3. Ability to configure at runtime usage capacity for services depending on seasonal variations</td>
</tr>
<tr>
<td>4. Optimize cycle time of internal business processes</td>
<td>5. Ability to quantify optimal response, service and cycle time</td>
<td>4. Virtualization (ability to configure computing resources dynamically, support HA and load distribution)</td>
</tr>
<tr>
<td><strong>Value Measures</strong></td>
<td><strong>Value Measures</strong></td>
<td><strong>Value Measures</strong></td>
</tr>
<tr>
<td></td>
<td>% reduction in time required to respond to business changes and needs.</td>
<td>1. % reduction in time to respond to service outages and apply corrective measures</td>
</tr>
<tr>
<td></td>
<td>2. % reduction in business process cycle time</td>
<td>2. % reduction in time to deploy new infrastructure or replace existing infrastructure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. % reduction in time to notify infrastructure warnings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Number of services</td>
</tr>
<tr>
<td>Business</td>
<td>Technology</td>
<td>Infrastructure</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>Adaptability</strong></td>
<td><strong>Expectations</strong> Ability to absorb and manage changes effectively – a change is inevitable, but the business expects that this change should not drastically impact the existing IT application space and it should be possible to assimilate any changes gracefully into the existing IT landscape</td>
<td><strong>Expectations</strong> 1. Standards driven flexible architecture – the technology should provide widely accepted standards such that incorporating additional components is simplified</td>
</tr>
<tr>
<td></td>
<td><strong>Capabilities</strong> 1. Ability to create/modify/integrate services/processes with minimal impact to the rest of business (as per changing market needs, trends, compliance etc.)</td>
<td><strong>Capabilities</strong> The capabilities required here are from the perspective of design/development time flexibility rather than runtime agility 1. Service Discovery 2. Service Definition 3. Service Creation (as defined by coding new services and as opposed to assembling a service dynamically at runtime) 4. QoS Policy Definition i.e. Interface details – the technology should provide the capability of standardized service interface definition 5. Open standards and standardized processes (e.g. eTOM, Accord) (interoperability, platform independence, vendor independence, canonical data formats, protocol independence) 6. Service traceability and Impact Analysis – the technology should provide this capability so that service change impacts can be analyzed and service proliferation avoided</td>
</tr>
<tr>
<td></td>
<td><strong>Expectations</strong> 1. Predictable and scalable infrastructure – the infrastructure should support horizontal and vertical expansion</td>
<td><strong>Capabilities</strong> 1. Standards based tools for design, development and deployment of services 2. Ability to scale 3. Enterprise wide standards-based middleware infrastructure (this is more from a perspective of reducing point to point interfaces so that flexibility is more)</td>
</tr>
<tr>
<td>Business</td>
<td>Technology</td>
<td>Infrastructure</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>Value Measures</strong>&lt;br&gt;1. % reduction in resource spent on other impacted services/processes due to creation/modification/integration of services/processes&lt;br&gt;2. % reduction in resource spent to manage the disturbance</td>
<td><strong>Value Measures</strong>&lt;br&gt;1. % reduction in resource spent on impacts to existing services on change of a service&lt;br&gt;2. % reduction in resource spent in integrating new services to existing services&lt;br&gt;3. % reduction in resource spent on determining impacts to existing services on change of a service</td>
<td><strong>Value Measures</strong>&lt;br&gt;1. % reduction in resource spent on adding new applications and integrating them into the infrastructure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Efficiency</th>
<th>Expectations</th>
<th>Capabilities</th>
<th>Value Measures</th>
<th>Expectations</th>
<th>Capabilities</th>
<th>Value Measures</th>
<th>Expectations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value Measures</strong>&lt;br&gt;1. Reuse productivity index of services – defined as sum (number of times service is reused by immediate service consumer)/total no of services.&lt;br&gt;2. % increase in productivity for service/process development&lt;br&gt;3. % reduction in number of services changed</td>
<td><strong>Expectations</strong>&lt;br&gt;1. Achieving desired agility and adaptability with optimal resource management (infrastructure, human capital, finance)</td>
<td><strong>Capabilities</strong>&lt;br&gt;1. Ability to reuse existing services/processes&lt;br&gt;2. Define KPIs for a process with measurable criteria&lt;br&gt;3. Build new services/processes with minimal effort and investment (business users to choreograph instead of IT team, reduced Software Development Process cycle effort)&lt;br&gt;4. Ability to decide quickly on build vs buy&lt;br&gt;5. Ability to reduce changes as opposed to minimizing the impact of changes – e.g. a configurable system will have reduced changes</td>
<td><strong>Value Measures</strong>&lt;br&gt;1. % improvement in availability of a service&lt;br&gt;2. % reduction in mean time to wait for a service&lt;br&gt;3. % reduction in idle time for an event completion&lt;br&gt;4. % reduction in resource utilization for service execution</td>
<td><strong>Expectations</strong>&lt;br&gt;1. Measure, Control and Optimize Resources</td>
<td><strong>Capabilities</strong>&lt;br&gt;1. Tools for design, development and deployment of services&lt;br&gt;2. Leverage existing application/infrastructure&lt;br&gt;3. Service and infrastructure monitoring and analysis</td>
<td><strong>Value Measures</strong>&lt;br&gt;1. % improvement in availability of a service&lt;br&gt;2. % reduction in mean time to wait for a service&lt;br&gt;3. % reduction in idle time for an event completion&lt;br&gt;4. % reduction in resource utilization for service execution</td>
<td></td>
</tr>
</tbody>
</table>
Service Mosaic Usage And Optimization

Based on the value measurement framework described in section 5, TCS proposes the following principles for an optimized service mosaic. These principles can be categorized as static or design time principles and dynamic or runtime principles. The static principles for an optimized service mosaic are:

- It should be possible to implement a business change or incorporate new processes for a new event by reusing existing services and creating some new services without altering the behavior of any existing services. This signifies that the services have been designed in a right-grained manner with the right amount of functionality so that they do not require to be frequently altered. The same applies to composing a new service using other existing infrastructure, application or business services.

- Services should be well documented with rich metadata so that they can be searched and discovered. As far as possible, there should not be any duplicate services with similar functionality. Strong service mosaic governance policies should be prescribed and implemented to avoid the problem of service proliferation.

- Services should be designed such that they can be deployed in multiple physical servers and share execution load without any dependency on the specific server or service instance. Services should be stateless.

- The reuse index of services that is the number of services reused across multiple business processes vs. the number of services used uniquely for a specific business process should be optimal. If services are designed in a very coarse grained manner, all business processes may have to invoke the service. If services are designed in too fine grained a manner, multiple business processes may end up invoking these services through other services. If service functionality is overlapping or duplicated in multiple services, different processes may invoke the functionality from different services and the reuse index will be very low. The reuse index of services will vary with the layer of the service that is reuse index will increase from business services to application services to infrastructure services.

- Over the time, the number of services being modified should tend to zero and the ratio of existing services reused to new services built should increase.

- The number of application services will progressively reduce. As explained in section 2 (Types of Services), the definition of an application service is based on the existence of pre packaged or custom developed applications. As an organization matures along its SOA path or in a completely ground up SOA implementation, there would be a minimal set of application services. The Service Mosaic would consist of a collection of process services which would in turn use business services and these business services would in turn use other business services or infrastructure services. Hence instead of existing applications exposing application services, we would have a logical application using process and business services from the mosaic.

- The number of infrastructure and business services is large while the number of process services is minimal. Process services are the highest level of services in an enterprise and would be equivalent to the number of business processes defined in the enterprise. Individual process services use a number of business services and infrastructure services to achieve the end result. Hence the number of process services in the service mosaic is minimal. Consequently, as a sequel to the first principle above, the infrastructure and business services will be large.

- The number of infrastructure services increases with respect to the number of business services as the enterprise moves towards maturity or steady state. Over time, as an industry or enterprise matures, its business services may become infrastructure services for example the payment gateway is a very widely accepted, standards based and mature functionality. Hence, today, the payment gateway service has become a ‘taken for granted’ infrastructure service. Similarly within an enterprise, over the
time some services become very mature and change very infrequently. Such services are identified as infrastructure services.

Service granularity is a very debatable topic and there is no ‘one size fits all’ solution to this problem. Let us take the example of a bank and the service to create an account. Accounts can be of different types say for example, savings bank account and current account.

We could create two different services to create a savings account and current account or we could create one service to create an account and internally manage the creation of both types of accounts based on contextual data provided. If the data elements required to create the two accounts are vastly different and exclusive to each other and the invoking business processes are also vastly different and mutually exclusive, it makes sense to create these processes as different processes. But, if that is not the case and the same business process with minor variations is used to create these accounts, then it makes sense to have one service called createAccount which will internally manage the creation of both these accounts.

The dynamic principles for an optimized service mosaic are:

- As far as possible, services should be designed with rich policies and metadata such that their behavior can be altered at runtime based on the specific context of the request. Service implementations should internally be based on configurable and rule engine techniques to realize the above.

- Deviations from Service QoS SLAs should be minimum. The infrastructure should be capable of being warned instantly in case of a breach of SLA and the infrastructure should be capable of immediately taking corrective action such as increasing the number of service instances, substituting other similar service instances in lieu of the failed service. QoS configurations itself should be configured based on the priority and criticality of individual services. The criticality of a service is defined on the basis of the criticality of the business process or event which triggers the service.

All the services defined in a service mosaic should be used one time or the other at runtime. Service QoS definition should be amended based on the service usage and the business processes which use the service – the higher the service usage, the more critical the service and the more strict its QoS SLA. Services that are rarely used should be evaluated for phase out or incorporating into other existing services.
Service Mosaic Realization Strategy

Business Process Management and service orientation are strategic initiatives in an enterprise in a move towards a self-optimizing enterprise. TCS may engage in different stages in the program based on the progress made by the enterprise in the above program and the overall objectives of the enterprise.

There are four entry points in a Business Process Management and service orientation of business and IT processes.

1. **Enterprise Business Transformation**: A strategic initiative to transform business and IT architecture and infrastructure with focus on long term initiatives and goals. Engagements of these kinds have high value proposition and long term benefits.

2. **Business Process Harmonization**: A focused initiative to harmonize and streamline complete business processes across units using service orientation.

3. **Service Oriented Solution**: These business driven initiatives are service orientation of the existing applications to realize the business processes. These engagements have medium value proposition and near term benefit realization. These initiatives are more like establishing the feasibility of the overall solution.

4. **Service Enabling in Existing Applications**: These programs focus on revitalizing the existing application portfolio by exposing existing business functionality as services which can be consumed by the business processes. These engagements have low value proposition and short term benefit realization.

**Enterprise Business Transformation**

This is a strategic program aimed at optimizing the business processes across the enterprise paving the way for a more agile and adaptable enterprise. A top down approach is taken to define the business processes and the processes are modeled using the standard business process management tools which follow the Business Process Execution Language (BPEL) standards. These business processes modeled helps in identifying the business services across these processes which can be implemented using the Web Services (WS) standards. The services can be of varying granularity and provides a common contract and interface that can be used by business processes.

The introduction of the business process layer not only helps enterprise to manage and monitor the business processes in real time but also promotes reuse of business functionality as services in the enterprise.

From an architecture standpoint the key is defining a Service Invocation Framework for all the process, business, and infrastructure services. This framework will be realized using a representative business process of the enterprise with sufficient touch points to end systems. The framework will provide a runtime mechanism for service consumers and provider using the WS standards. The invocation framework will follow the service design principles described in section 2.

**Business Process Harmonization**

This engagement would focus on standardizing the business processes across the enterprise resulting in consistent and standards based process in the enterprise. This will follow a top-down approach of defining business processes in the enterprise. The exercise may involve modeling of new business processes or remodel existing processes to make it consistent in the organization.

The objective of the engagement is to streamline the business processes and follow services driven approach for realizing the business processes. This helps the enterprise to adapt to changing business scenarios more rapidly.

The business and infrastructure services identified above can be realized either by using revitalizing the existing applications or built ground up using WS standards based approach. A service invocation framework will be defined around process, business, and infrastructure services which can be used to invoke the services.
Service-Oriented Business Solution
Service Orientation of business primarily focuses on a particular business unit within the enterprise. The business processes are defined for this unit and the services identified based on these processes. A combination of top down and bottom up approach can be taken to identify the services. These services can either be realized using the existing applications or built completely new using standards based approach.

For invoking the services the invocation framework will be defined and all services will use this framework.

Since the focus is limited to a business unit the amount of reusability may be limited. This is more tactical initiative to understand the feasibility of the services oriented solution and how it benefits the enterprise.

Service Enabling in Existing Applications
In this scenario the primary focus would be to streamline the existing business process and improve their efficiency by revitalizing the existing applications. This means the business functions strewn across the applications will be exposed as services. This may involve changes to these applications in order to service enable these business functions. Once the services are enabled on the end applications these services can be composed together to orchestrate a business process.

The services defined on the end application follow the design principles described in section 2 and the standards for WS.

These engagements are primarily focused on improving the efficiency of business processes result in cost benefits to the enterprise. These are tactical engagements to understand the feasibility of the service based solution and how it translates to business value for the enterprise.
Conclusion

The result of the service orientation of an enterprise is a “Service Mosaic” which is the realization of the top-down functional decomposition of business processes (enterprise business process modeling) and bottom-up modularization and encapsulation of application functionality as services. The top-down approach helps in identifying coarse-grained business services and the composition of these services into business processes. The bottom-up approach will give domain services and their mapping to applications. A middle-out approach ensures that the solutions meet business requirements of agility-adaptability even while ensuring effective leverage of existing investments.

The critical success factor for realization of service mosaic is in identifying services at the right level of granularity with right domain categorization across both the functional and non-functional (QoS) aspects of implementation. We have defined the service discovery process which enterprises must follow to identify what we have classified as Process services, Business services, and Infrastructure Services. As far as possible, services should leverage the metadata and policies such that their behavior can be altered at runtime based on the specific context of the request.

In order to build a consistent and optimal service mosaic in the enterprise it’s important to have a governance model. This paper talks about the governance team in managing the service design, service meta data, and its run time management. The SOBET and the Service Architecture Group play important roles by the virtue of being a centrally operating structure in close co-ordination with business owners while leaving the realization of the service mosaic to the IT delivery teams near to the business in respective units.

The service mosaic is not a static solution but represents a dynamic framework that adopts with changing business and IT requirements. Once an organization starts on the SOA roadmap, the optimizing principles can guide it towards higher efficiency of the services itself that is very important for effective realization of benefits. Towards this end the enterprise can look at building a Service Mosaic optimization model around process, business and infrastructure services (e.g. No. of services, Reusability of services, Cost of implementing these services etc.) enabling enterprises towards a self optimizing business and IT infrastructure.

Finally any solution will have to be measured in terms of the business benefits that it brings to the business. We have detailed out the value measurement framework that enables such analysis along three dimensions of Business, Technology, and Infrastructure for an enterprise to be Agile, Adaptable, and Efficient.
About Enterprise Application Integration Practice
With presence in 20+ countries, multi-skilled EAI consultants, a rigorous delivery methodology and thorough understanding of the local challenges, TCS EAI Practice is uniquely positioned to provide value added solutions across the world. Aided by its TIME™ methodology for streamlined delivery, partnerships with major EAI product vendors and utilizing the Global Networked Delivery Model, customers can look forward to robust and timely solutions in traditional EAI space, platform rationalization, service orientation and business process management (BPM).

About Tata Consultancy Services
Tata Consultancy Services (TCS) is among the leading global information technology consulting, services and business process outsourcing organizations. Pioneer of the flexible global delivery model for IT services that enables organizations to operate more efficiently and produce more value, TCS focuses on delivering technology led business solutions to its international customers across varied industries.

For more information contact
Ravindra Asundi
Global EAI Practice Lead
Tata Consultancy Services,
Gateway Park, Road No 13,
MIDC, Andheri (East)
Mumbai 400 093, India

Phone: 91 22 6750 6826
Fax: 91 22 6750 6855

Email: ravindra.asundi@tcs.com
Website: www.tcs.com

TATA CONSULTANCY SERVICES
All content / information present here is the exclusive property of Tata Consultancy Services Limited (TCS). The content / information contained here is correct at the time of publishing. No material from here may be copied, modified, reproduced, republished, uploaded, transmitted, posted or distributed in any form without prior written permission from TCS. Unauthorized use of the content / information appearing here may violate copyright, trademark and other applicable laws, and could result in criminal or civil penalties.

Copyright© 2006-07 Tata Consultancy Services Limited