SOA Solution Stack 2.0

A Solution View of SOA Reference Architecture

CommunityPaper Version 2.0, November 7, 2006 (Version 1.0, September 2004)

Core Team Members:
Ali Arsanjani (Editor, V1)
Liang-Jie (LJ) Zhang (Editor, V2)
Michael Ellis
Abdul Allam
Kishore Channabasavaiah

Acknowledgements

With thanks to the SOA Community Contributors:

Abdul Allam, Ali Arsanjani, Kishore Channabasavaiah, Raj Cherchatil, Arnauld Desprets, Yaroslav Dunchych**,** Michael Ellis, Donald Ferguson, Rolando Franco, Biffle French, George Galambos, Stephen Graham, Rob High, Kerrie Holley, Joe Hardman, Rao Kormili, Min Luo, Emily Plachy, Siddarth Purohit, Robert Rafuse, Rachel Reinitz, Rick Robinson, Raghu Varadan, Dan Wolfson, Olaf Zimmerman, Jamshid Vayghan, Karrem Yusuf, Liang-Jie Zhang.

With special thanks to the Application Services R&D (ASRD) SOA Application Factory (SOA-AF) Project Team members:

Jia Zhang, IBM Watson Research
Teresa Aguilera, AS, GBS

Editors:

Ali Arsanjani, Chief Architect, SOA Center of Excellence, SOA Community of Practice Leader, arsanjan@us.ibm.com (S3 V1)

Liang-Jie Zhang (LJ), Research Staff Member, Founding Chair of Services Computing Professional Interest Community (PIC), IBM Research, zhanglj@us.ibm.com (S3 V2)

Table of Contents

Table of Figures

0 Executive Comments (A Note on "CommunityPapers" [1])

This CommunityPaper is available in three formats. First is the current digital snapshot of the CommunityPaper. Second, a version of this CommunityPaper will be maintained on the SOA CoP Wiki which will be focal point for community comments, updates, and feedbacks to maintain this as an ongoing, vital document supported by a community of practice. This will ensure on-going feedback and vitality as the community updates the topics herein. Third, this CommunityPaper will act as the knowledge transfer that will occur over ongoing lecture series and sustained bi-weekly calls of the core team.

[1] This notion was introduced by the SOA Community.

1 Overview

This paper describes the on-going results of an SOA & Web Services Community of Practice (SOA CoP) Special Project called SOA Solution Stack (S3), which major goal aims at defining a conceptual view of a service-oriented architecture. This CommunityPaper does not include an SOA method but can leverage the Service-oriented Modeling and Architecture (SOMA) method for the analysis and design of activities, roles, and work products needed to build an SOA-based solution.

In response to the question "what is an SOA solution?", we would like to offer this CommunityPaper as an answer from IBM. In short, a service-oriented architecture decouples service providers and consumers using a service description that covers both the interfaces (what the services can do and how to access the services) and policies (e.g., requirements and constraints) needed to describe an abstract yet executable specification of the functional and non-functional attributes of a service.

1.1 The SOA Architectural Style

To a software architect, an *SOA is an architectural style with a set of highly reusable components with standard interfaces, as well as connectors, constraints, composition criteria, and containers* [].
Its contained *components* are the Service Consumers, Service Providers (providing functionality based on the service-level agreements specified by the service consumers within its declarative policy) decoupled by a Service Description that can be discovered ("find" or "search"), bound to ("bind"), and invoked via a mechanism such as the Web Services Description Language (WSDL). Note that WSDL and the associated Web Services "stack" are not the only way to implement an SOA, but they are recommended, standards-based approaches to realize SOA.

The *connectors* between providers and consumers are their Service Description which includes a Service Interface (a mechanism for a loosely coupled functional interface, e.g., WSDL) and a Policy (a mechanism to declaratively specify non-functional requirements on both provider and consumer sides) (e.g., WS-Policy and WS-Policy Attachments). A dedicated Integration Layer can serve as the connector between the consumers and the provider(s) who can potentially fulfill the service requests. Note that an Enterprise Service Bus (ESB) is a recommended implementation of this Integration Layer as a best-practice for deriving the maximum value and flexibility out of this architectural style. However, note that an ESB is not a mandatory construct; sometimes, more primitive forms of SOA tend to employ point-to-point connections between consumers and providers. On the other hand, although not a mandatory part, an ESB helps achieve decoupling and appropriate routing and transformation needed to match and invoke endpoints of a service consumer's request with actual service providers who can fulfill that request.

The *constraints* are special requirements attached to the connectors that have to be enforced (sometimes they are also called the principles of SOA). Constraints should have the following three features:

1. declaratively specified
2. capable of being published, found and bound to
3. Capable of having their bindings altered at run-time for optimal adherence to service level agreements (beyond the scope of SOA but a key factor in their negotiated success in a service eco-system)

The *compositions* consist of choreographies or orchestrations (if the distinction is critical) of atomic or composite services (those service whose implementations rely upon other service components (implementations) or other service interfaces.

The *containers* may include application servers with support for SOAP over various protocols (e.g., HTTP/HTTPS, JMS, SMTP, CICS, and FTP) and choreography-based standards (e.g., BPEL engines). Such an entity can be found in the IBM Websphere Process Service product as examples.

1.2 SOA Solution Stack Reference Architecture: A Solution View

SOA Solution Stack (S3) includes a logical reference architecture that describes a Service-Oriented Architecture. It provides a high-level abstraction of an SOA factored into layers, each addressing specific value propositions within SOA.

An S3 is a "partially-layered" architecture in that a layer is not strictly hidden from the layers above it. For example, a consumer may choose to access a service through either the business process layer or the service layer. A service may also choose to leverage two styles of service implementation: namely, service components like EJBs or .NET components or packaged applications such as SAP or Siebel or other SOA-enabled legacy applications.

Our logical view of an SOA depicts it as consisting of a set of layers, ABBs (Architectural Building Blocks), architectural and design decisions[2], and so on. The solution space is viewed from four perspectives: an SOA reference architecture (see

[2] See the **meta-model in the section entitled** The SOA Reference Architecture Metamodel

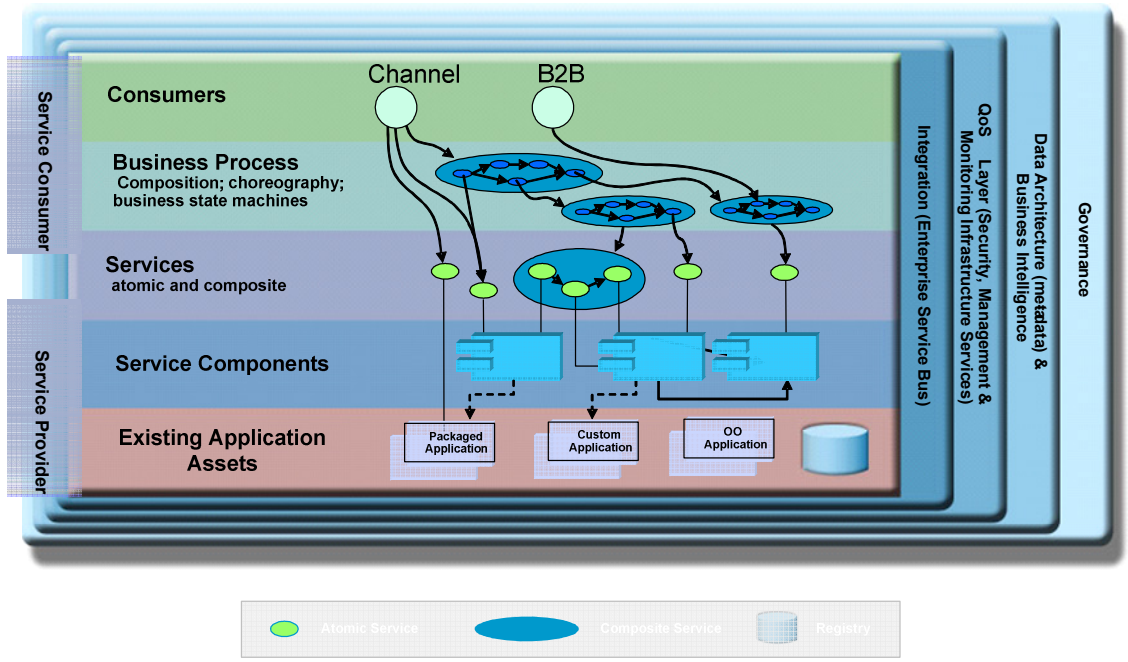

Figure 3), IBM recommended software and hardware (e.g., the implementations of the ESB Gateway pattern are available in both hardware and software by multiple vendors), Service Offerings and Assets (e.g., Order to cash services, SOMA, and SOA Assessments) and Industry-specific models and applicable standards.

This helps in the understanding and implementation of separation of concerns and assists in the process of creating an SOA in conjunction with methods such as the Service-oriented Modeling and Architecture (SOMA) method. The S3 defines a blueprint that can be used to define the layers, ABBs within the layers, options available at each layer, and typical architectural decisions that need to be made (see meta-model in Figure 4. Metamodel for the SOA Reference Architecture).

We recognize that there are various paths to realize SOA [1] in addition to S3, including business process driven (business services), tool-based model-driven, message-based application integration through SOI (integration services), data access-based (information services), legacy encapsulation and wrappering, legacy componentization and transformation, and so on.

Note: Finally, the content of this paper is generally restricted to topics that are salient in the discussion of SOA. At times judgment calls were made to include/exclude topics; these calls were made based upon the balance of how important those topics were vs. the ability to contain the task of authoring the paper. Specifically, comments on the use of software products and technology are not intended to be comprehensive of their use.

1.3 SOA Solution Stack

The basis of the SOA Solution Stack Project is the current SOA Reference Architecture Overview Diagram, which is an abstraction of the fundamental building blocks and architectural style of an SOA solution. It is not a strictly layered architecture, but rather a partially layered one in which a set of functional layers (0-5) collaborate via the SOA Infrastructure to deliver business values through a set of business-aligned IT services that collectively fulfill an organization's business processes and goals.

S3 is an answer to the question "If I build an SOA, what would it look like conceptually and what abstraction would be present?" It enumerates the fundamental elements of an SOA solution. Moreover, it presents the architectural foundation of an SOA solution, from which various perspectives can be derived: tooling, standards, industry specification, and so on.

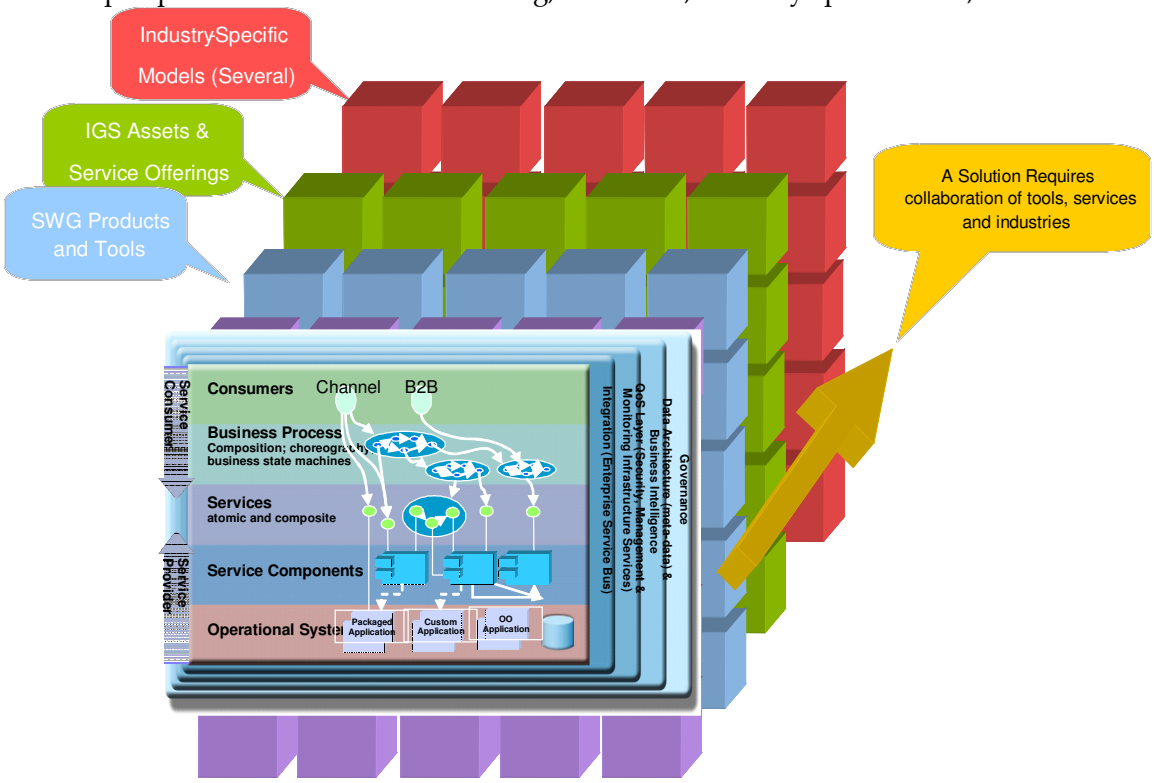

Figure 1. Multi-Dimensional SOA Solution Stack

As shown in Figure 1, the base model (reference architecture: solution view) presented in this CommunityPaper forms the foundation for the rest of the three stacks: the software products stack, the services stack and the industry specific stack. Three categories of stacks are recognized: a services (IGS) stack, a tools and products stack (SWG) and an Industry-specific SOA Reference Architecture for each major industry and eventually for all industry in which IBM is engaged.

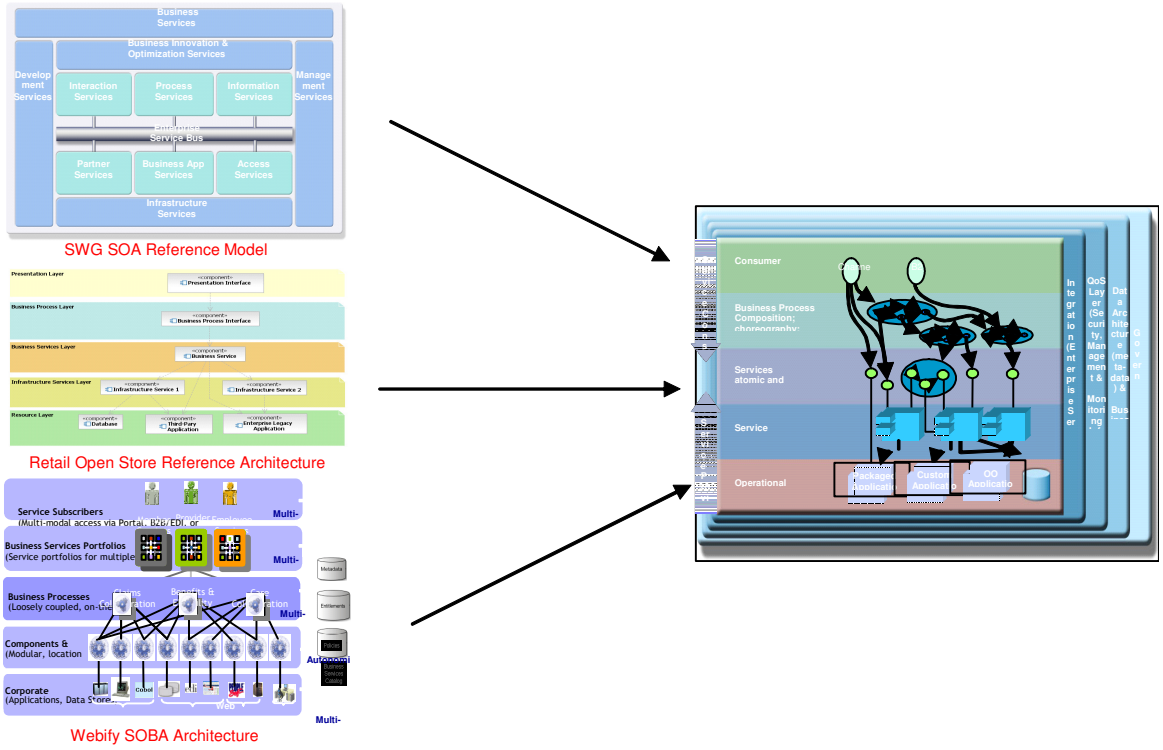

Figure 2. The SOA Reference Architecture: Solution View

Figure 2 shows that SOA Reference Architecture – SOA Solution Stack (S3) seamlessly incorporates various SOA structural models, typically from SWG SOA Reference Model, Retal Open Store Reference Architecture, and Webify SOBA Architecture.

Figure 2 also shows the convergence of models/architectures to the IBM SOA Reference Architecture. As a brief example, the layers of the software stack in Figure 2 can be mapped to the following products and software entities: (Source: Don Ferguson's presentation on SOA Solution Stack Summit, Keynote, Dec 2004)

1. Existing applications
2. Adaptors, JCA
3. SCA, SDO
4. BPEL4WS, Adaptive Entities, Business Rules, Policy
5. WSRP, JSF, WCT
6. ESB, Mediations
7. Policy, Containers
8. (New Item)

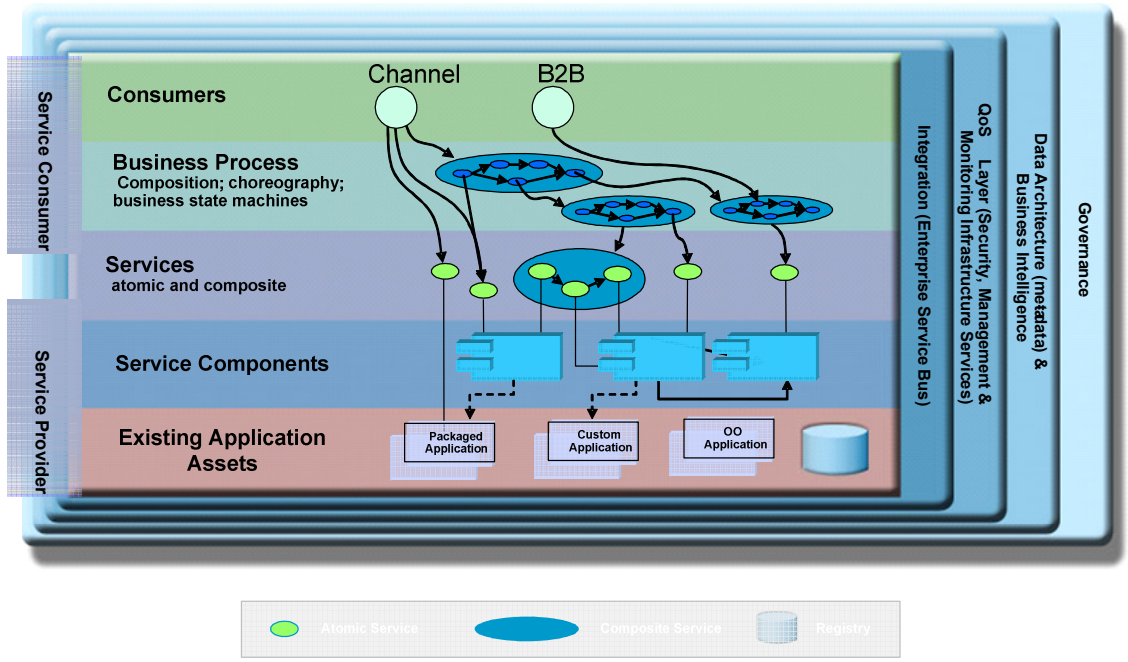

Figure 3 shows an abstraction of how to construct a Service-oriented Architecture (SOA) as a set of logical layers. Note that this representation is not designed to imply a strictly layered architectural style in which one layer solely depends upon the layer below it. Rather, an S3 is a *partially* layered architecture in which a company has a set of layers that are more service consumer-oriented (consumer layer, business process layer, and service layer) and a set of layers that are more service provider-oriented (service layer, service component layer, and Existing Application Asset layer). Cutting across both the consumer layers and the provider layers are a set of non-functional layers: integration layer, quality of service (QoS layer), data architecture layer, and governance layer.

A service consumer may access a service through the service layer directly or through the business process layer. A service provider will provide an implementation of a service in a service component or through the wrappering of an existing system or packaged application. Further, the Service provider will ensure quality of service through the quality monitoring and management supplied by the QoS layer. The communication between a service and its service component implementation or operational system will occur via the integration layer. If there is a point-to-point connection (discouraged), it will come through this layer; if there is an Enterprise Service Bus (ESB), it will also reside in the integration layer.

Let's take a brief look at the rationale behind the layers. The Existing Application Asset layer can be thought of as an abstraction of the physical runtime. Consider the runtime of a company's infrastructure operating as a set of interconnected nodes with programs and processes running on them, collaborating and functioning to provide the IT capabilities of an enterprise.

Then, let us take a snapshot of those operational systems (applications and data) in time; abstract and expand the runtime nodes that reside in the Existing Application Asset layer. This expansion of the logical layers, ABBs and SOA infrastructure (the fabric or foundation that the layers depend upon at runtime) is depicted in

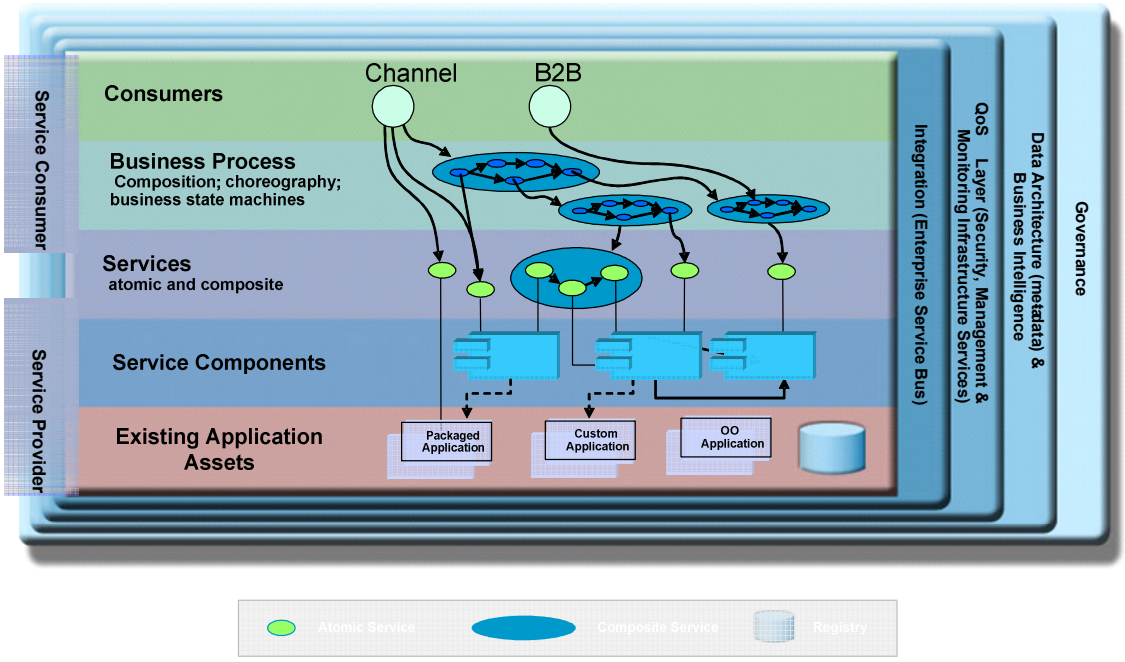

Figure 3 in terms of multiple separation of concerns shown in the 9 layers of this architectural reference model.

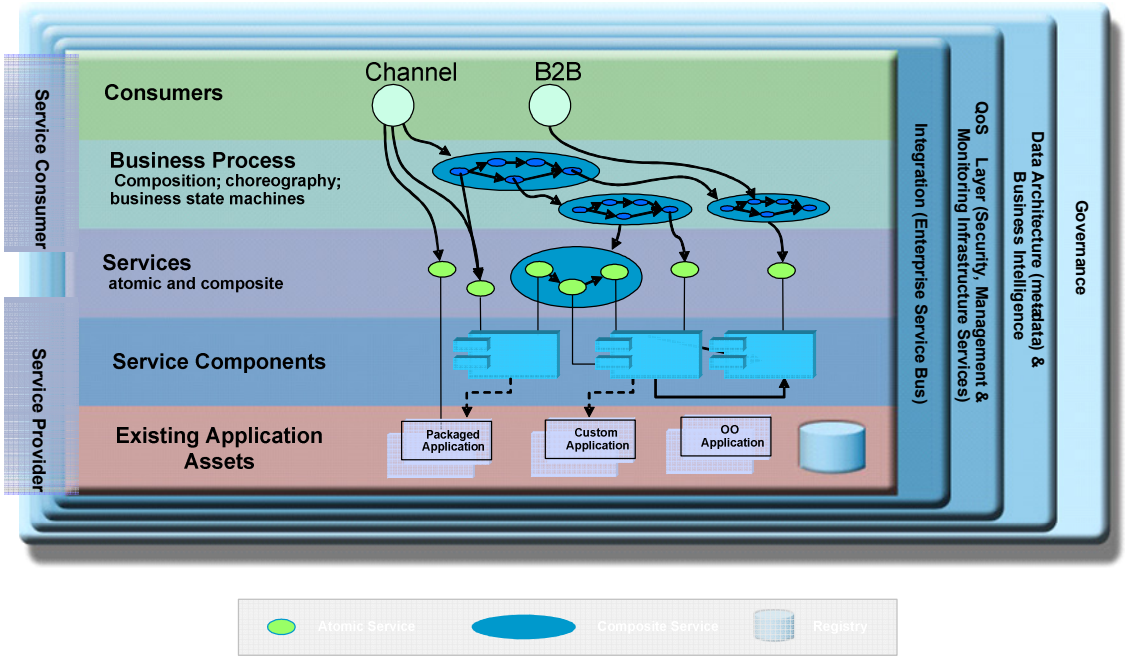

Figure 3. Layers of the SOA Solution Stack Reference Architecture

Note that the provider and consumer arrows in

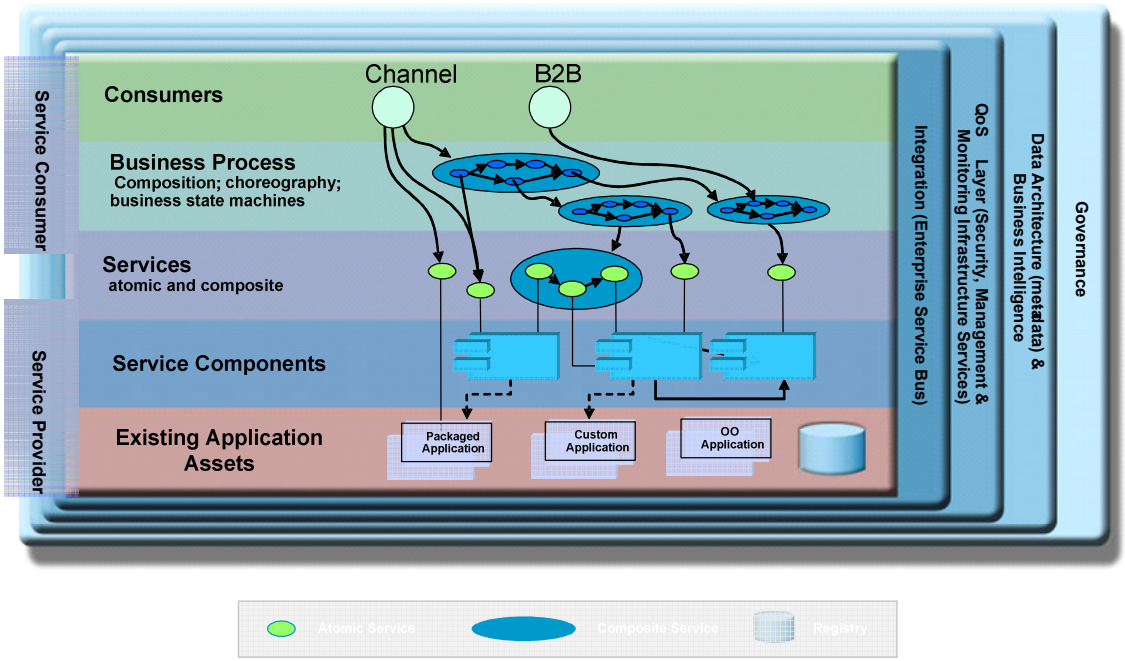

Figure 3 are used to depict the fact that layers 1-3 (6-9) are for Service Provider concerns and layers 3-5 (6-9) are for Service Consumer concerns. As shown in

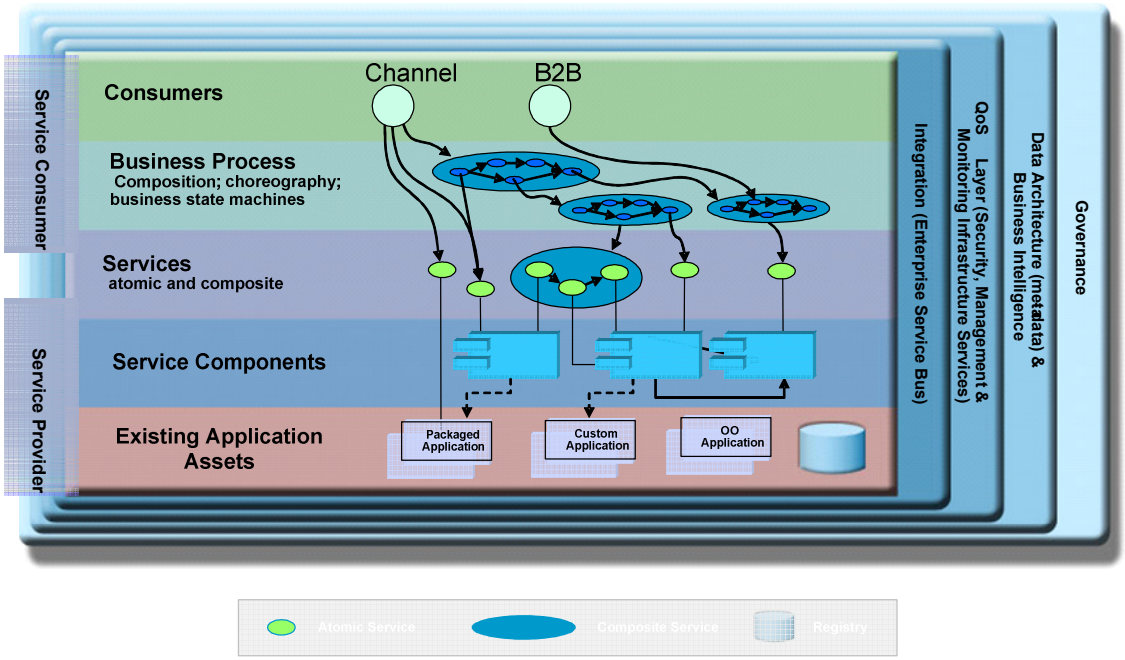

Figure 3, the lower layers are concerns for the provider and the upper ones for the consumer.Note that an organization can assume multiple roles. For example, in a supply chain, a consumer can leverage the services of a provider who in turn may depend upon another service. This multi-role paradigm of SOA roles is as valid within the confines of a single organization as it is within a larger service (SOA) ecosystem. A project or business line can be a consumer of services offered by another project or line of business. Alternatively,

and especially in the earlier stages of SOA maturity, the same project or line of business will be both provider and consumer. (See the Service Integration Maturity Model [5] for a more in-depth discussion of the adoption, transformation and maturity on the path to realizing SOA).

The SOA reference model is an enterprise architectural template that guides the creation of SOA solutions at the enterprise level by defining a reference architecture. Organizations may start with this reference model, customize and apply it for developing solutions for one or more lines of business. Thus, the reference model is used as an architectural template, customized and applied based on the needs of a service domains that integrate and interact among themselves. For example, S3 helps us design SOA solutions, leveraging SWG ESB-based SOA reference architecture. Using the same S3 we can deliver our SOA business services, using the same deployment framework.

We recognize, also, that there are two distinct groups of services within a given organization: external services and internal services. The first is the set of services that have been discussed thus far. They are business-aligned services that fulfill an organization's business processes and goals. These services can be tied back to business processes. They are services that can be exposed to other lines of business or to the outside world of partners and the SOA eco-system. These are called external or business services. The second group contains those internal services addressing IT integration and infrastructure problems. We do not necessarily apply the same rigor in the identification and exposure of this type of services. Though these internal services are designed to meet non-functional or quality of service requirements, they may directly represent an important aspect of the SOA value proposition.

1.4 The SOA Reference Architecture Metamodel

The conceptual model that underlies the rationale behind the SOA Reference Architecture is depicted in the metamodel as shown in Figure 4. This is a model of the S3 model, which illustrates the comprising elements of an S3 model and their inter-relationships.

Figure 4 uses a UML 2.0 class diagram to show the metamodel elements and their relationships between each other. Each element is represented by a UML class; each identified relationship between a pair of elements is represented by a UML 2.0 relationship.

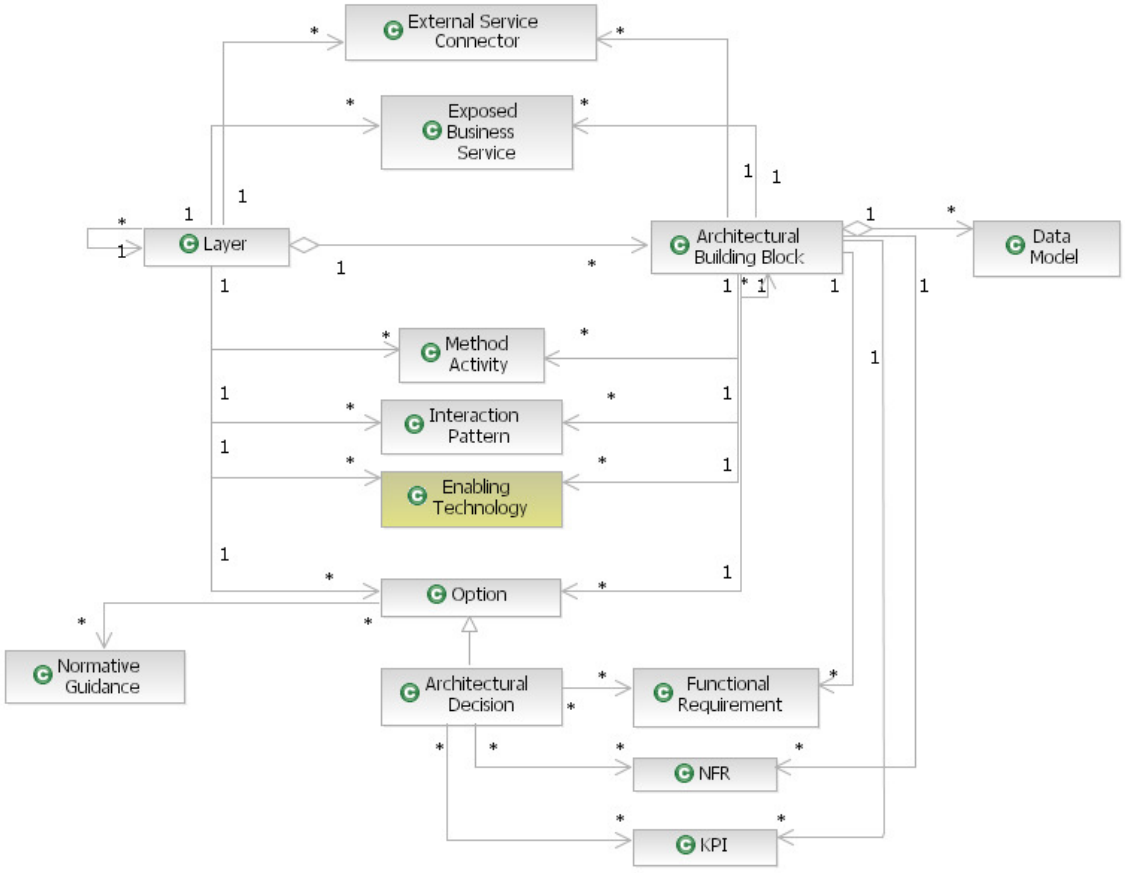

Figure 4. Metamodel for the SOA Reference Architecture

The SOA metamodel thus consists of the following fourteen elements:

1.4.1 Layers

It is an abstraction of the nine layers of S3, which contains a set of components such as ABBs, architectural decisions, interactions among ABBs and interactions among layers.

The reason why layers are important is that the separation of concerns groups the responsibilities, interaction patterns and options at each layer, in terms of software platforms to be used, protocols, design and architectural patterns and other design principles and guidelines.

1.4.2 Options

It is a collection of high-level solution-level patterns that impact other artifacts. For example, options may guide to choose appropriate architectural decisions and interaction patterns. There are options in each layer, as to what the ABBs are, what their instantiations are in terms of software and platforms, what best practices are in terms of their interaction patterns.

Note that we use *options* to represent high-level design decisions, for example, how many layers should be used for a given SOA solution, which question can neither be answered and covered by architectural decisions that focus on ABB identification and selection, nor be covered by interaction patterns that focus on interactions among ABBs.

In summary, options focus on solution-level patterns or principles that an SOA solution architect should consider and make decisions upfront. Key options include deciding which SOA methodology to use, for example, starting from legacy transformation from bottom-up, starting from business process re-engineering, starting from message-driven (B2B) methodology (using a dedicated integration layer with message routing), starting from top-down business process decomposition, starting from user-centric solution design (define channels to be used), or starting from open composition approach (starting with the Service layer to identify services to be used first).

Note that options are different from architectural decisions, and governance. The former is oriented to low-level decisions, while the latter is oriented to specific scenarios (e.g., industry-specific scenarios require specific governance as normative guidance). In contrast with architectural decisions and governance, options represent high-level decision making points. In other words, options can be viewed as generalization (i.e., super class) of architectural decisions and governance.

In addition, options have many perspectives, which also include selection of enabling technology that decides which products and technologies to be used to realize an SOA solution. The latter selection is also knows as product and technology mapping.

1.4.3 Architectural Decisions

It is one type of options specific for architectural designs of an SOA solution. The architectural decision involves ABBs, functional requirements, Key Performance Indicators (KPIs) and Non-Functional Requirements (NFRs) to provide information on configuration and usage of ABBs. Existing architectural decisions may also be reusable by other layers or

ABBs. As discussed earlier, comparing to options that consider high-level decisions, architectural decisions consider low-level decisions specific to architectural designs, for example, how to choose each ABB. Architectural decisions have to be made as to which one we will adopt for a given set of circumstances. This context and the forces that reside in the problem space shape a pattern that defines the architectural decisions typically encountered in a layer, across layers and across ABBs.

Note: As a starting point, in S3 Version 2.0, we will provide architectural design decision points and recommended options and solutions for the Consumer layer. Once the S3 model is used in practice, we will keep on adding more architectural design decision sections for each layer in the S3 model.

1.4.4 Normative Guidance

It is one type of options specific for scenario-specific normative guidance. In general, normative guidance may cover several layers of meanings: ABB-level guidance, layer-level guidance, and high-level guidance. ABB-level guidance refers to low-level guidance at ABB levels, such as how to implement an ABB. Layer-level guidance refers to guidance at S3 layer levels,such as how to integrate several layers into an SOA solution. High-level guidance refers to solution-level guidance such as how to linkage business service architecture and IT service architecture to into a specific SOA solution.

In general, we provide normative guidance in two levels, one as generic normative guidance that serves as high-level over all guidance to an entire SOA solution development; the other one as layer-specific normative guidance that serves as specific guidance for a specific layer in the S3 model.

It should be noted that it is not required that every layer in the S3 model has normative guidance. For some layers, we do provide layer-level normative guidance. Once the S3 model is used in practice, gradually we will keep on adding more normative guidance associated with the corresponding architectural design decisions into the "normative guidance" section of each layer.

Note: As S3 Version 2.0, we provide high-level solution-level normative guidance in Section 1.6.

1.4.5 Method Activities

It is a collection of steps that involve ABBs to form a process to populate components in a layer.

1.4.6 ABBs

A set of Architectural Building Blocks (ABBs) reside in a layer that contains attributes, dependencies and constraints as well as relationships with other ABBs in the same layer or different layers.

These components of each layer are the constituent elements of that layer that provide the essentail functionality and deliver the responsibility assigned to that layer. They depict the "types" of things that can reside in that specific layer.

1.4.7 Interaction Patterns

It is an abstraction of the various relationships among ABBs (patterns and diagrams) within and across layers.

1.4.8 KPIs

It is a set of key performance indictor constraints that involve in ABBs and put concerns on making architectural decisions.

1.4.9 NFRs

It is a set of non-functional requirement constraints that involve in ABBs and put concerns on making architectural decisions.

1.4.10 Enabling Technology

It is a technical realization of ABBs in a specific layer by selecting which technology or product, e.g., whether an IBM technology or an Independent Software Vendor (ISV) technology, should be selected to fulfill the enablement. It shouls be noted that unlike other components included in the metamodel as logical concepts, enabling technology indicates actual physical realization of a speicific SOA solution.

1.4.11 Exposed Business Services

Exposed business services (a.k.a. externalized business solution elements) refer to entities that expose business processes or composite business applications as business services that can be reused as service assets. This construct enables a new way to quickly compose existing service assets using business processes and expose the composed business processes as new business services for future reuse.

1.4.12 External Service Connectors

External service connectors refer to adaptors (e.g., transformers) for exploiting external services for business connections and business integrations. Business services developed by different business entities may use different technologies so that there might exist inconsistency between them, including interface inconsistency such as parameter inconsistency and method name inconsistency. In order to link together these individually developed business services, adaptation is necessary and required. These adaptors are meant for enable business service interoperability between different business entities. Of course, adaptors are typically designed for selected solution scenarios.

1.4.13 Data Models

It models data contents associated with ABBs including data exchange between layers and external services.

1.4.14 Functional Requirements

It models the functional requirements that one layer or ABB must fulfill.

From section 2 (Layer 1: Existing Application Asset Layer) to section 10 (Layer 9: Governance), we will use the above elements of the metamodel to describe the characteristics of the SOA Reference Model, the layers and the building blocks and architectural decisions about each layer. Later, in section 13 entitled *The SOA Infrastructure: "Platform 9 ¾ "[3]*, we will describe the underlying aspects of the infrastructure model that defines the contracts between layers and the fabric that enables the functionality of the SOA Reference Model.

1.5 Assumptions

For each layer, there is a specific mechanism by which the service requirements influence that layer.

A service requirement is the documented capability that the service needs to deliver or is expected to deliver. Each service requirement has two views, a provider view and a consumer view. The provider view of a service requirement is the business and technical capability that a given service **needs** to deliver. The consumer view of a service requirement is the business capability that the service is **expected** to deliver to fulfill the business functionality either in the context of a business process or in the consuming application including a portal solution. The fulfillment of the service requirement may be delegated to different layers of the S3 and a service requirement may have to be met by the capabilities and responsibilities of each of the layers of the S3. In addition the contracts between the layers and the inter-layer dependencies may influence the fulfillment of the service requirements both in the consumer and the provider layers of the S3.

The service requirement is comprised of both the functional and non-functional requirements that must be met by a given service. Non-functional requirements include aspects of Quality of Service (QoS), security and manageability during the lifecycle of a service. It is a key aspect in developing a service-oriented architecture for an enterprise, by identifying the service requirements desired to deliver a given business functionality and mapping that to each of the layers of the S3 based on the layer capabilities and responsibilities.

Service quality assurance and testing is largely influenced by the service requirements defined upfront in developing and implementing an SOA solution. Meeting the criteria set-

forth in the overall service requirements both in terms of the functional and non-functional capabilities of a given service is key to successful deployment of a service in an SOA.

The next section describes in detail the different layers of the S3 which is a generic reference model for SOA. Service requirements, when applied to the S3, result in a specific SOA for a given enterprise.

It should be noted that each layer comprising logical and physical parts. The physical part of each layer is part of the overall infrastructure of SOA solutions, including (1) the operating and runtime environments such as the application containers and the runtime engines required by the other layers of the S3, (2) the runtime engines and operating environments that enable the wrappering or packaging of legacy applications to be delivered as a set of one or more services, (3) the hardware and software components including ISV software package solutions. In more detail, the infrastructure of SOA solutions provide: an operational environment for the services of the SOA solution, and an operational environment for the other layers of the SOA solution including application containers and runtime engines. Therefore, our SOA reference architecture layers do not cover physical parts.

The following core principles define this architectural description:

- If a functionality or capability required is not found in a specific layer n, then look for it in layer n-1, if not then look for it in Integration or Data Architecture layer.
- S3 Layers provide a logical and centralized view of functionality/capability. Physical implementation of these functionalities can be distributed through Business Process Choreography, Service Component, Integration Component, and so on.

2 Layer 1: Existing Application Asset Layer

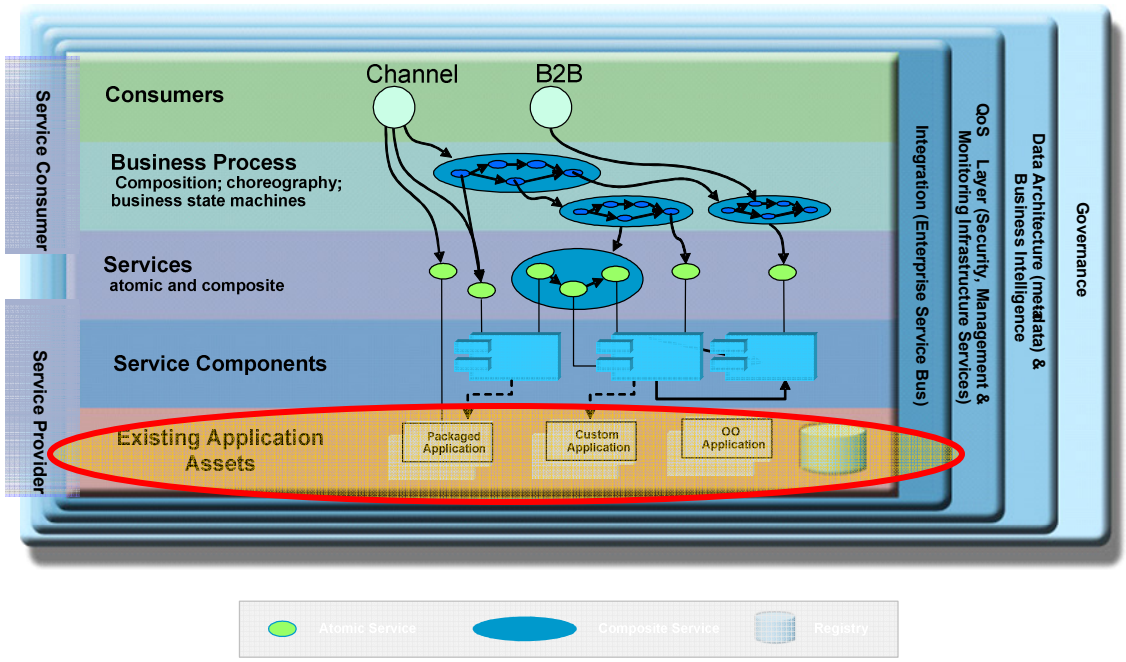

Figure 5. Existing Application Asset Layer in S3

The Existing Application Asset layer contains existing packaged applications (applications typically provided by individual software vendors (ISVs)), customer applications (applications developed in-house or to-be-developed), and legacy systems (existing applications typically developed in traditional technologies) that can be reused as assets for further service composition. Traditionally, these applications could only be used for one purpose and serve one specific user. With the aid of SOA, they can be exposed as services with standard interfaces; so that they can be reused by other upper-level services.

It should be noted the supporting physical infrastructure (hardware), runtime environment, operating system environment, netrworking equipment, and database are part of SOA supporting infrastructure, which is used to realize SOA solution by using middleware and supporting infrastructure.

2.1 Layer Description

This layer includes all custom or packaged (e.g., CRM, ERP, etc.) applications and systems in the application portfolio running in an IT operating environment to support business activities.

The Existing Application Asset Layer or Layer 1 is the foundation of the SOA Solution Stack or the Reference Model architecture that is defined to virtually represent an SOA solution. The Existing Application Asset layer also includes the operational and run-time environment required to deploy an operational SOA within an enterprise.

The layer is comprised of different software systems including the custom monolithic applications, legacy applications, transaction processing applications, database software systems, Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM) packages and solutions and the runtime environment required for these systems. The operating system environments and the associated hardware platforms are all part of this layer.

2.2 Value Proposition

This layer is a fundamental layer of the SOA Solution Stack and includes all the run-time environments required for services to be operational in an SOA. The characteristics and responsibilities of the Existing Application Asset layer influence the success of a deployed SOA to a large extent.

The Existing Application Asset layer is typically comprised of existing software systems thereby leveraging existing IT investments in implementing an SOA solution. This directly influences the overall cost of implementing SOA solutions within enterprises and also frees up some parts of the overall budget for newer initiatives and development of business critical services.

A number of existing software systems typically are part of this layer. Those systems include:

- Existing monolithic custom applications such as J2EE and .Net applications
- Legacy applications and systems such as IMS applications
- Existing transaction processing systems such as CICS systems
- Existing package applications and solutions such as ERP and CRM packages, including SAP and Siebel solutions

2.3 Characteristics

A set of key characteristics are integral to the Existing Application Asset layer. These characteristics include a number of attributes and responsibilities which are specific to this layer. These characteristics influence the behavior and the characteristics of the other layers. The non-functional requirements and service level attributes of the layers are also influenced by the characteristics of this layer.

2.3.1 Attributes and Responsibilities

As the foundational layer of the SOA Solution Stack, a set of attributes and responsibilities are identified as integral to Operational Systems layer, including:

- Provide infrastructural foundation for the realization of the SOA solution
- Host the applications and the application functionality required to deliver the service functionality in the Service layer of the S3.
- Support the functional and non-functional requirements of the implemented services in the SOA solution.

The Existing Application Asset layer requires a set of inputs influencing the realization of the SOA solution and delivers outputs which can be leveraged by the other layers of the SOA Solution Stack for their definition and realization.

The inputs of the Existing Application Asset layer include:

- Required service functionality
- Service non-functional requirements including SLAs, QoS Attributes, Service performance, service availability, etc.
- Runtime engines and environments required to deliver the functional and non-functional requirements of all the services in the SOA solution

The outputs of the Existing Application Asset layer include a set of existing applications that can be used by upper layers.

2.3.2 Key Performance Indicators

The Existing Application Asset layer, as a key foundational layer of the SOA Solution Stack, includes a set of generic KPIs that influence the definition and realization of this layer. These KPIs include:

- Service Performance
- Service Availability
- Service QoS
- Service Uptime
- Service Resource Requirements

2.4 Architectural Building Blocks (ABBs)

As shown in

Figure 6, we identify nine fundamental Architectural Building Blocks (ABBs) in the Existing Application Asset layer: (1) packaged application, (2) customer application, (3) legacy system, (4) operating system controller, and (5) federated security enablement.

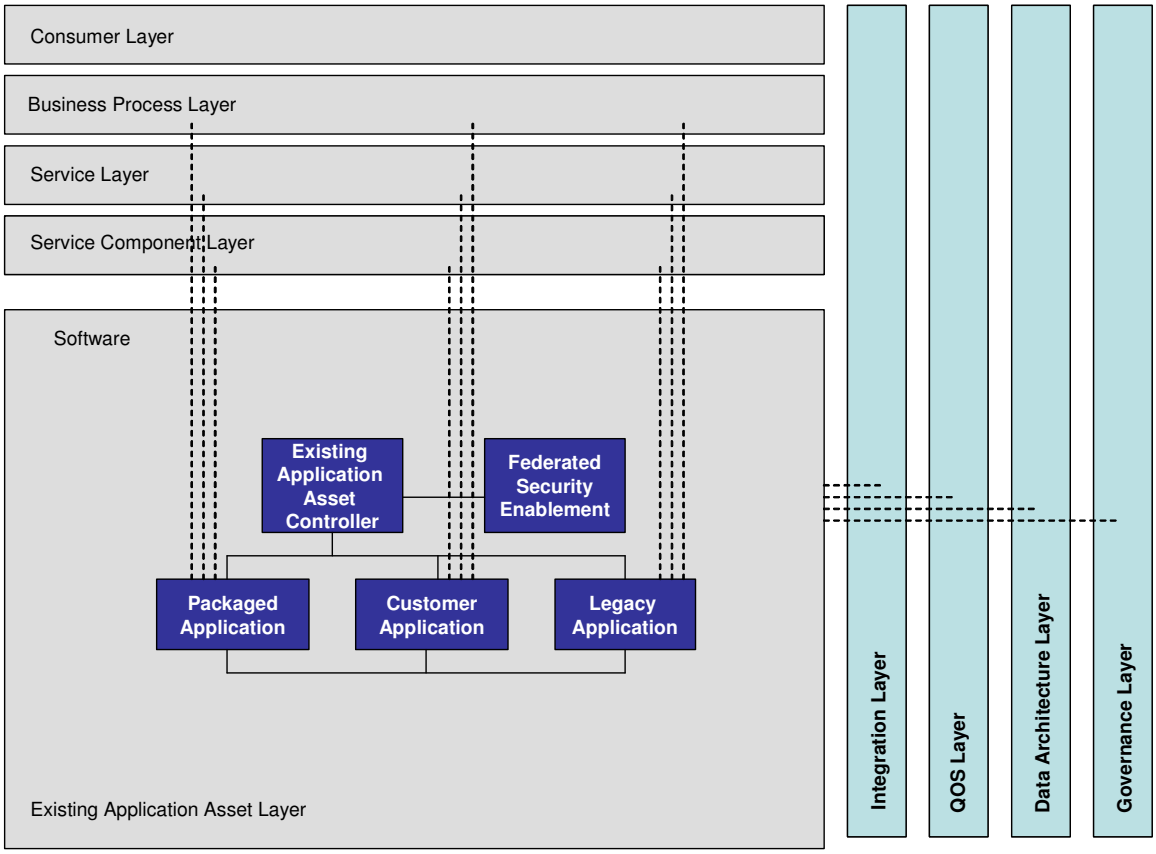

Figure 6. *Existing Application Asset layer ABBs.*

2.5 Component Descriptions – Existing Application Asset Layer

This section describes in detail each ABB in terms of its responsibilities.

2.5.1 Packaged Application

A *packaged application* building block refers to existing applications typically provided by Individual Service Vendors (ISVs), e.g., Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), SAP applications, Siebel solutions, database packages, imaging solutions, content management solutions, knowledge management solutions, financial solutions packages, human resources package solutions, and collaboration suites such as SameTime and Lotus Email.

2.5.2 Customer Application

A *customer application* building block refers to custom monolithic applications. Typically, these are home-built applications or to-be-built enterprise-specific applications. Examples

include any "homegrown" system or application that supports business or infrastructure functionality.

2.5.3 Legacy Application

A *legacy application* building block refers to existing applications typically using traditional technologies.

2.5.4 Existing Application Asset Controller

An *existing application asset controller* building block provides central management over other ABBs within the Existing Application Asset layer: packaged application ABB, customer application ABB, legacy system ABB, and federated security enablement ABB. In other words, the existing application asset controller acts as a coordindator among reusable existing application assets.

2.5.5 Federated Security Enablement

A federated *security enablement* building block provides federated security control and enablement over other ABBs within the Existing Application Asset layer through the existing application asset controller ABB. Each existing application may have its proprietary security enablement mechanism. The federated security enablement ABB provides a unique solution-specific security enablement coordinating different security access control. Single-Sign-On and Microsoft passport are two typical techniques implementing federated security enablement. Theoretically, other ABBs could utilize the federated security enablement ABB directly. However, S3 strongly recommend that they go through the federated security enablement ABB.

2.6 Dependencies and Interactions (patterns)

In this section, we will discuss the relationships between the identified ABBs within the Existing Application Asset layer and across other layers.

2.6.1 Relationship Diagram within the Existing Application Asset layer

Figure 7 uses a UML component diagram to illustrate a static view of the relationships between the ABBs within the Existing Application Asset layer. All identified ABBs are represented as components in the diagram, with "ABB" as stereo type. The Existing Application Asset layer is represented as a package containing all identified ABBs.

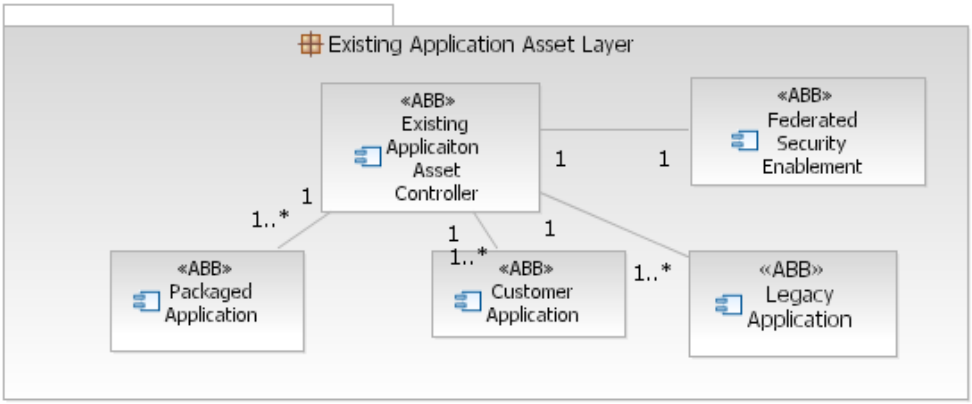

Figure 7. *Component Relationship Diagram – ABBs within the Existing Application Asset Layer.*

Certain relationships exist between the twelve types of ABBs:

- There is a one-to-many relationship between the existing application asset controller ABB and the other four ABBs: (1) packaged application ABB, (2) customer application ABB, (3) legacy application ABB, and (4) federated security enablement ABB.

Theoretically, there maybe an optional relationship between the customer application ABB and the packaged application ABB. However, S3 strongly recommends that they interact through the operational system controller ABB.

2.6.2 Relationship Diagram across Other Layers

Figure 8 uses a UML component diagram to illustrate a static view of the relationships for the ABBs across layers. Various layers identified in the S3 are represented as packages; ABBs are represented as components.

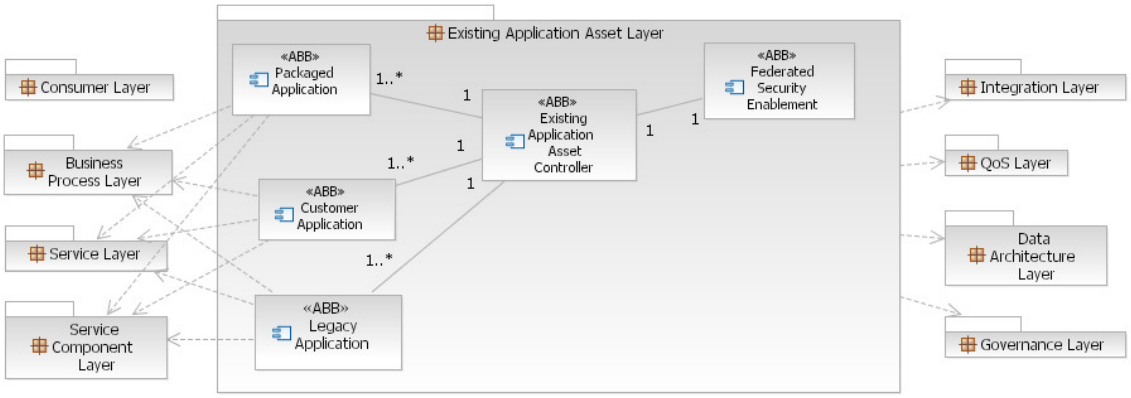

Figure 8. *Component Relationship Diagram – ABBs in the Existing Application Asset layer across layers.*

Certain relationships exist between the ABBs in the Existing Application Asset layer with other layers:

- A *packaged application* ABB, a *customer application* ABB, and a *legacy application* ABB may interact with the Business Process layer, the Service layer, and the Service Component layer.
- The entire Existing Application Asset layer interacts with the Integration layer, the QoS layer, the Data Architecture layer, and the Governance layer.

2.7 Options and Design Decisions

The major option and design decision for this layer is how to map service requirements to a specific application function (a realization decision).

2.8 Activities

A number of activities influence the definition, organization and structuring of this layer within the overall S3. These are related to the different SOA lifecycle activities including model, assemble, deploy and manage steps of realizing an SOA solution.

Some of the activities that are integral to the Existing Application Asset layer include:

- Leveraging existing software and hardware components within the overall SOA solution
- Componentization of existing applications to implement the services that will deliver the business functionality required in the SOA solution
- Wrapping existing applications with a Web services wrapper to implement the services that will deliver the business functionality required in the SOA solution
- Leveraging existing package solutions using techniques and technologies to implement the services of the SOA solution
- Leveraging the existing and new run-time environments that are the building blocks of this layer to host components of the other layers of the S3 including service components of the service component layer.
- Leveraging the infrastructure capabilities of the hardware components to realize the non-functional requirements of the different services and the overall SOA solution.
- Consolidation of redundant and duplicate systems thereby realizing the business value of the SOA solution.

3 Layer 2: The Service Component Layer

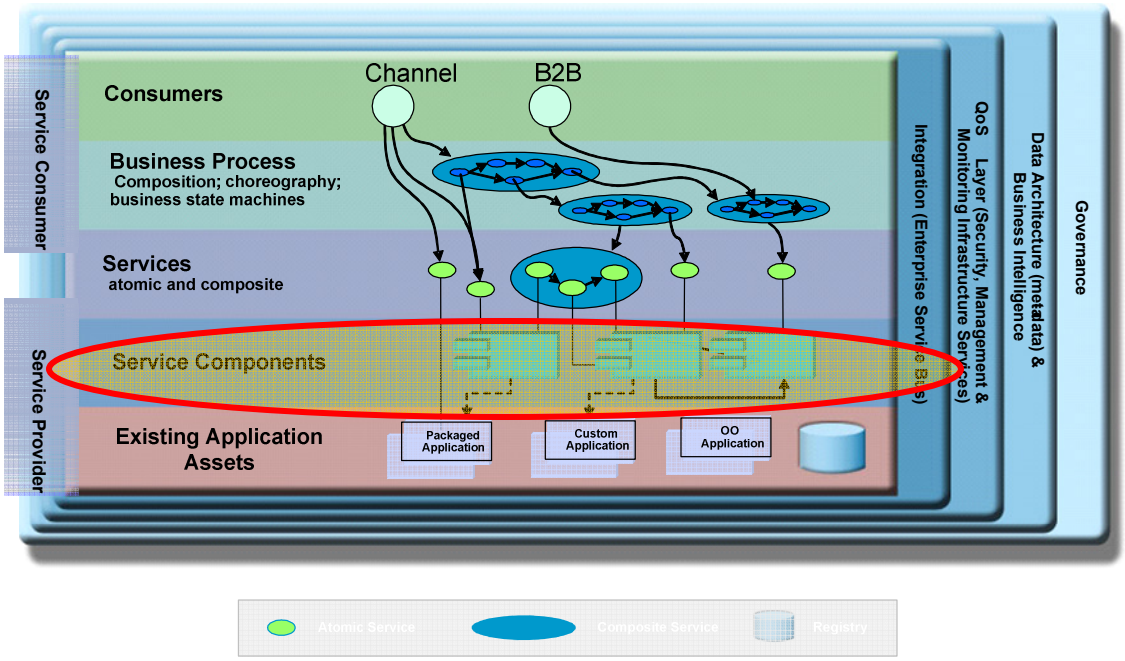

Figure 9. Service Component Layer in S3

3.1 Layer Description

The Services Component layer provides code containers that implement services (i.e., service interfaces) defined in the Service layer. A service component may rely on some existing service assets (e.g., packaged applications, customer applications, and legacy applications) from the Existing Application Asset layer, some services from the Service layer, and some business processes from the Business Process layer. A service component can be implemented in a Java Class, EJB, .Net component, and so on. In addition, a service component may include the implementations of multiple methods, while some methods are exposed as services in the Service layer. Moreover, the Service Component layer is also responsible for automating input transformation and output adaptation (e.g., parameter adaptation) for services.

3.2 Value Proposition

The Service Component layer manifests the IT conformance with each service contract defined in the Service layer; it guarantees the alignment of IT implementation with service description.

In detail, each service component fulfills the following two goals:

- Provides an enforcement point for service realization
- Enables IT flexibility by strengthening the decoupling in the system, by hiding volatile implementation details from service consumers.

Refer to the example in Figure 10, where "Service A" is implemented using a combination of the behaviors from 3rd parties "Package X" and "Application Y". "Application B", the consumer of "Service A", is coupled only to the description of the exposed "Service A". Note that the consumer must assume that the realization of the service is faithful to its published descriptions (service compliance) and it is the service provider's responsibility to ensure that it is. The details of the realization, however, are of no consequence to Application B. Service Component A acts as a service implementation façade; it aggregates available system behaviors and gives the servcie provider an enforcement point for service compliance.

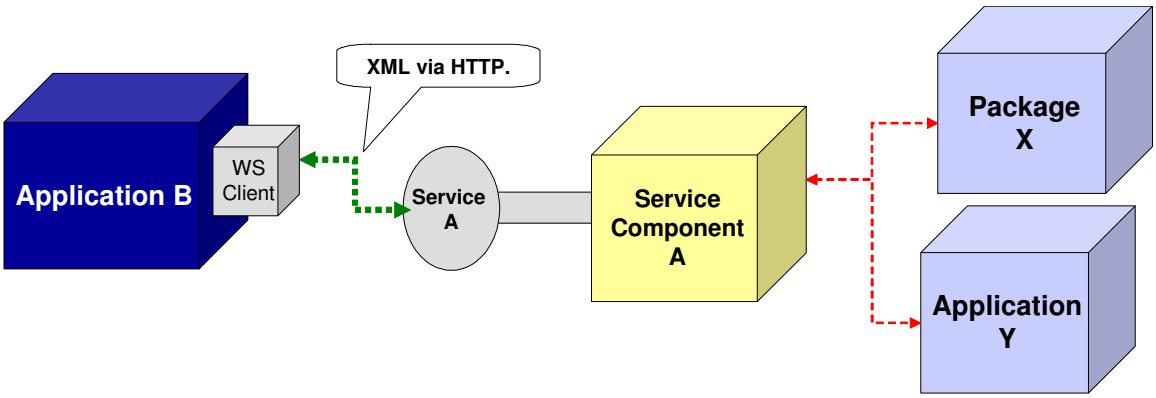

Figure 10. Service Abstraction: Service Component as a Service Router or Façade.

Subsequently, the provider organization may decide (for reasons that are independent of a change in the service contract) to replace "Package X" with "Package M", as shown in Figure 11. When this occurs, the required modifications are encapsulated in service component A with the result that there is no impact on any consumers of service A[3]. This example illustrates the value of the Service Component layer in supporting IT Flexibility through encapsulation.

[3] This assumes that the deployment of the altered service components and the subsequent binding of requests to the new implementation could be achieved in a transparent manner.

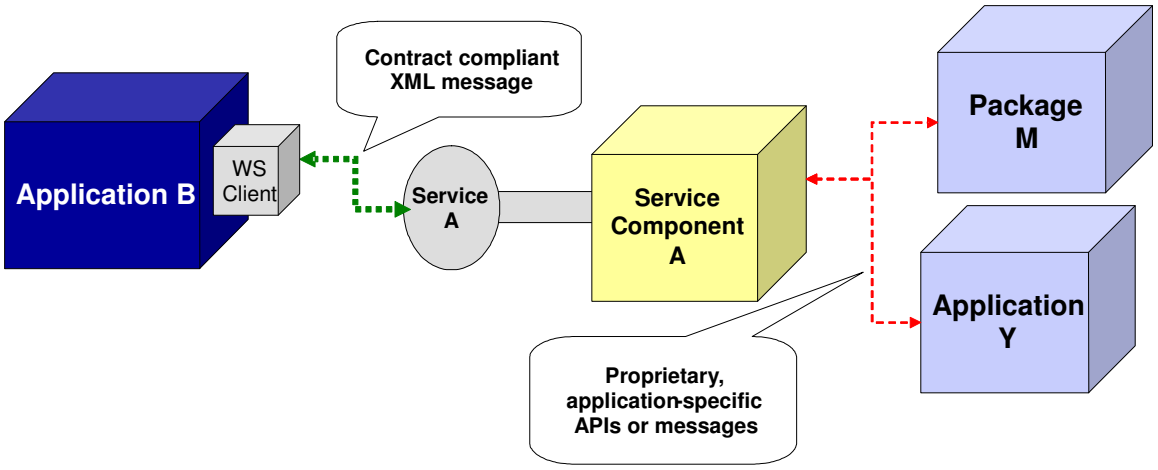

Figure 11. Flexibility to Replace Functionality

3.3 Composition Scenarios

Composition of existing application assets occur frequently in the context of transforming legacy systems into services. Figure 12 shows an example of how to construct service components using existing application assets. Assume two existing application systems – *Applicaiton 1* that maintains customer addresses, and *Applicaiton 2* that validates postal codes. These two existing applications were developed on proprietary platforms using proprietary technologies. In other words, these two applications are legacy systems. A new project intends to build two Web services that can be accessed via SOAP prototol. First is a "safe" address updating service that would validate postal codes before making changes to addresses. Second is to build a dedicated "validate postal code" service.

As shown in Figure 12, there is a need of having an "adapter" component that represents particular existing application assets and provides an API for service components to consume and expose through necessary functions of this particular application assets. For example, both legacy applications have their own dedicated adaptors. "Adapter" is owned by the same organization who owns the application asset system and is legislated to be the only way to access this application asset. Such an adapter is provided as an API to the legacy system.

In order construct a SOAP-enabled service component, a "service enabler" is a typical instrument, as shown in Figure 12. Each service component contains a service enabler component if it contains access to legacy systems through adapters. As shown in Figure 12, service component 1 composes two legacy systems through dedicated adapters, and enables the composed service to SOAP access through an enabler 1; service component 2 composes two legacy systems through dedicated adapters, and enables the composed service to SOAP access through an enabler 2. Note that application 2 only has one adapter, which is reused in both service component 1 and service component 2.

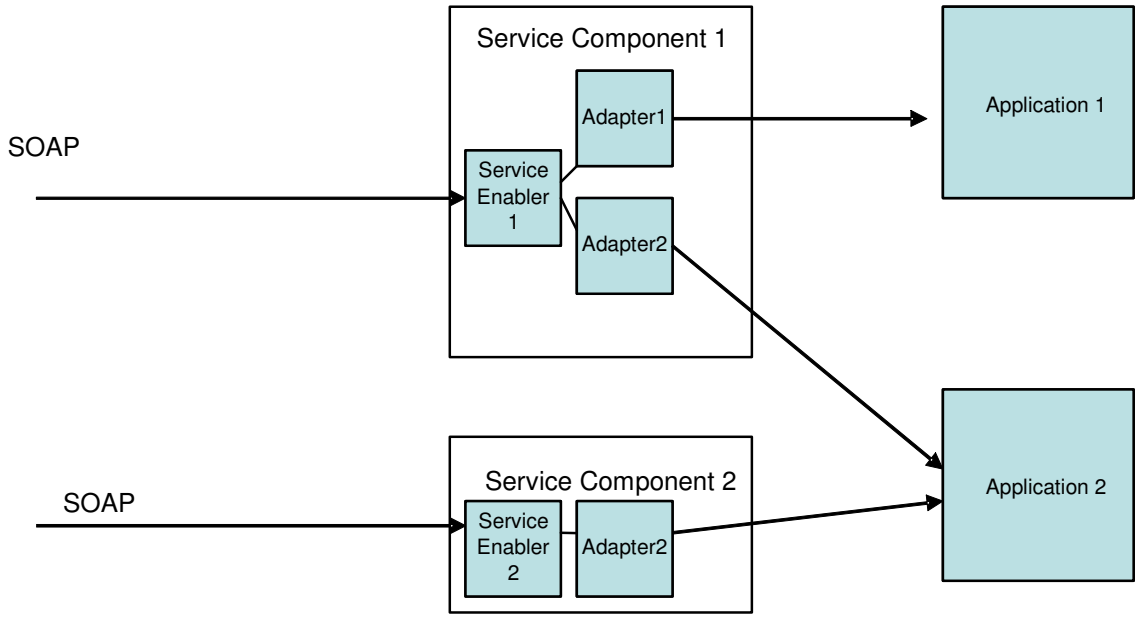

Figure 12. Implementation Scenarios for Service Components

3.4 Characteristics

The Service Component Layer is:

A collection of implementation of services exposed in the Service layer. In other words, this layer is comprised of software components (e.g., EJBs or Java classes) that implement the service definitions in the Service layer.

Service Components:

- should employ architectural/design best practices
- are often implemented with container–based technologies, e.g., application servers that enables the implementation to take advantage of high-availability, load-balancing, etc.
- enable aggregation and migration of existing application assets for a given service realization.

3.5 Key Performance Indicators (KPIs)

Given that this layer provides the implementations of the various services defined in the Services layer, the salient KPIs are those that are of common concerns in enterprise systems, such as latency, availability, scalability, reliability and security. Latency refers to the delay of accessing a specific service due to internal implementation details and processes. Availability refers to the percentage of how a specific service can be available during a specific time frame. Scalability refers to the ability of a specific service that supports various scales of consumer groups. Reliability refers to the ability of a specific service that stays no

fail during a specific time frame. Security refers to the ability of a service that provides authorization and authentication facility to ensure secure access.

3.6 Architectural Building Blocks

As shown in *Figure 13*, we identify eight fundamental Architectural Building Blocks (ABBs) in the Service Component layer: (1) service implementation module, (2) method input transformation, (3) method output adaptation, (4) service deployment module, (5) service publishing module, (6) service invoker, (7) application adaptation module, and (8) service component repository.

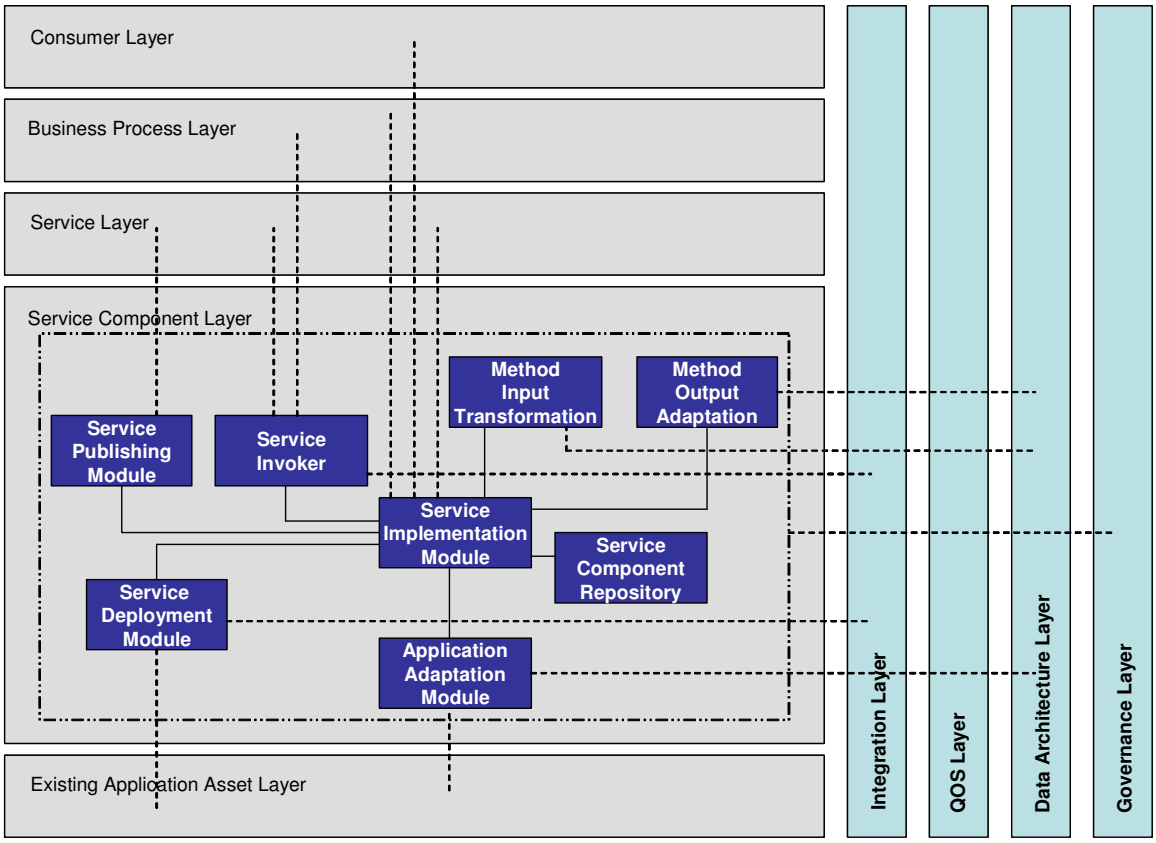

Figure 13. *Service Component layer ABBs.*

3.7 Component Descriptions – Service Component Layer

This section describes in detail each ABB in terms of its responsibilities.

3.7.1 Service Implementation Module

A *service implementation module* building block manages the real implementation (i.e., actual programming code in various languages, e.g., Java, .NET, C++, etc.) of service components in the Service Component layer.

3.7.2 Method Input Transformation

A *method input transformation* building block is responsible for transforming the input parameters of method (i.e., operation) signatures, between the real service components and published service components.

3.7.3 Method Output Adaptation

A *method input adaptation* building block is responsible for transforming the output parameters of method (i.e., operation) signatures, between the real service components and published service components.

3.7.4 Service Deployment Module

A *service deployment module* building block is responsible for deploying the actual implementation of a service component to corresponding deployment platforms. This ABB typically manages the deployment descriptor of a service component.

3.7.5 Service Publishing Module

A *service publishing module* building block is responsible for publishing a service component to the Service layer. In more detail, it directly interacts with the service interaction manager ABB in the Service layer to publish a service component to the service registry ABB in the Service layer.

3.7.6 Service Invoker

A *service invoker* building block is responsible for invoking service components (including services from the Services layer or business processes from the Business Process layer).

3.7.7 Application Adaptation Module

An *application adaptation module* building block is responsible for adapting a bottom-up packaged application from the Existing Application Asset layer to the implementation module ABB in this layer, so that it can be treated as a reusable service component.

3.7.8 Service Component Repository

A *service component repository* building block is responsible for storing information of service components retrievable from (registered with) the Service Component layer. The purpose of this ABB is for reusability. Before creating integration with legacy services, the Service Component layer needs to find available service components.

3.8 Dependencies and Interactions (patterns)

3.8.1 Transformation

To comply with the definition of "Service A" in Figure 14, "Service Component A" must perform any necessary translations to/from XML that are required due to dissimilar information formats that are consumed/emitted by "Package X" and "Application Y". To accomplish these translations, the Service Component layer may make use of the translation services defined in the Integration layer.

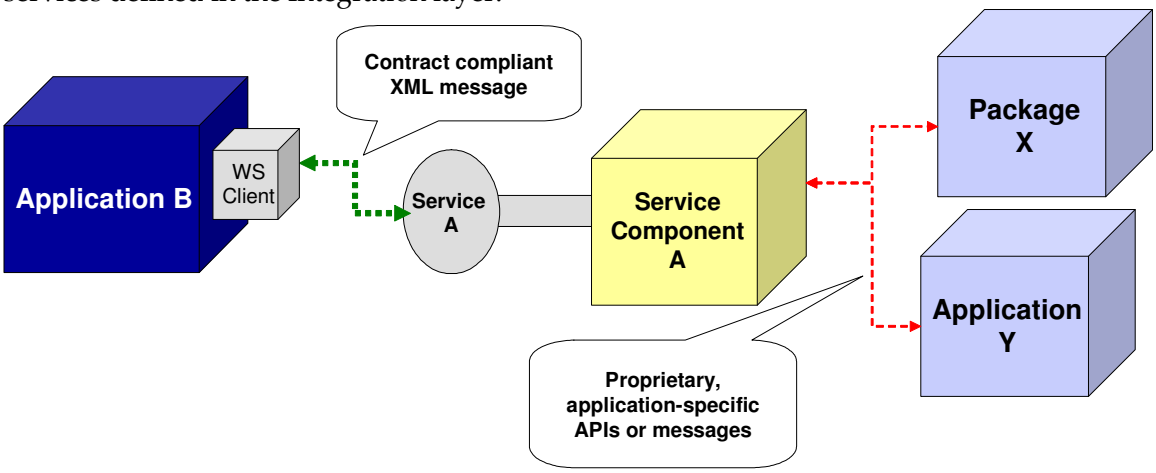

Figure 14. Message Transformation

3.8.2 Service Component Aggregation

A service can be realized through the aggregation of existing service components and/or additional behaviors from the Existing Application Asset layer.

Example: The "Profile Service Component" shown in
Figure 15 is realized by aggregating the behaviors of 3 other components. In this example, "Name & Address" is already exposed as a service but "Beneficiary" and "Preferences" are not exposed as services; they reside in the Existing Application Asset layer.

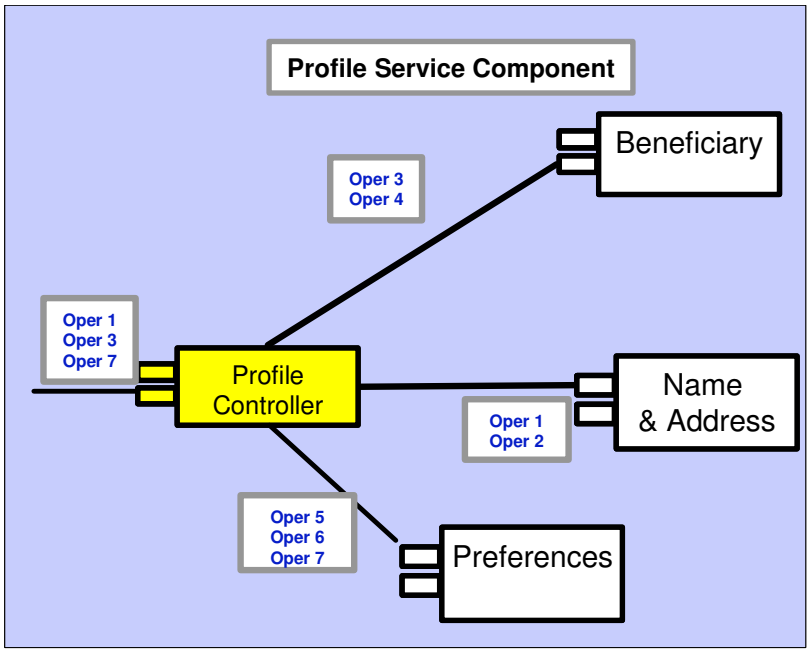

Figure 15. Service Component

The Profile Service Component delegates its behaviors to a coordinating component, the Profile Controller, which has knowledge about the other components, interacts with other components, and sequences the order in which the operations belonging to various components will be invoked. The operations of the collaborating components may also be exposed through the controller component interface.

It is important to note that the role of controller may also reside in the Profile Service Component itself, particularly if the implementation platform provides suitable encapsulation mechanisms, e.g., Session EJBs in J2EE.

3.8.3 Interaction with the Governance Layer

IT Governance has a significant influence on the Service Component layer. The choice of implementation technology, the manner in which Service Components may/may not consume behaviors from other service components and decisions about where to place integration logic, are two examples of where this layer may be influenced by IT Governance. For another example, the implementation options for a service component may include BPEL, a Session EJB, a Message Broker Flow, a SOAP/CICS operation, etc. Some of these alternatives may eliminate/involve a Governance exception, because the Technology Roadmap established by IT Governance excludes them.

3.8.4 Interaction with the Service Layer

By its nature, this layer is coupled to the Services layer of the S3 model. A service definition change is likely to cause a direct side-effect on the service component in this layer. For

example, if a service is retired from the Services layer, the corresponding service component will also be retired[4]. Finally, it is the responsibility of service components to faithfully reflect the definition of one or more services. To ensure that this relationship is maintained, a service component **must not** exhibit behaviors not defined in a service description.

3.8.5 Interaction with the Existing Application Asset layer

Service components often consume behaviors from the Existing Application Asset layer. This relationship creates a dependency on the behaviors consumed. If a decision is made to change the manner in which the Existing Application Asset layer behavior is realized, there are likely side-effects on the service components that consume it. For this reason, traceability between Service Component layer and the Existing Application Asset layer is an important element of an SOA.

Also, it is often the case that Existing Application Asset layer behaviors required by a service component is not available in a convenient fashion. In such circumstances, the refactoring of the behaviors in the Existing Application Asset layer may be necessary. This is an example of why the implementation of SOA may result in or require changes to existing Existing Application Asset layer.

3.8.6 Relationship Diagram within the Service Component Layer

Figure 16 uses a UML component diagram to illustrate a static view of the relationships between the ABBs within the Service Component layer. All identified ABBs are represented as components in the diagram, with "ABB" as stereo type. The Service Component layer is represented as a package containing all identified ABBs.

[4] It can be argued that the service component may continue to provide values when the service has been retired, as it may continue to be used in the implementation of other service components or services. While this is true, it is important to recognize that the significance of the software component as a service-aligned component has been lost and as such it is no longer subject to the same governance as a service component.

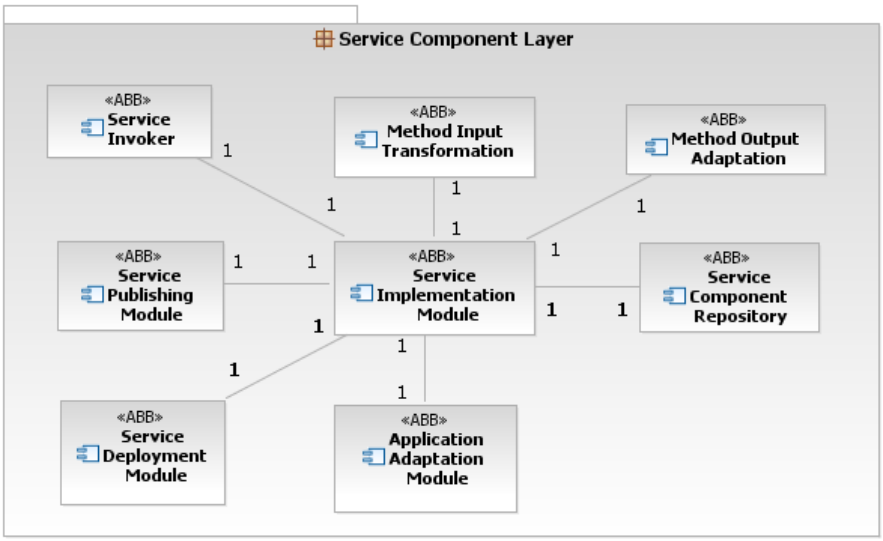

Figure 16. *Component Relationship Diagram – ABBs within the Service Component Layer.*

Certain relationships exist between the eight types of ABBs:

- Each method input transformation ABB, method output transformation ABB, application adaptation module ABB, service deployment module ABB, service component repository ABB, and service invoker ABB needs to interact with the service implementation module ABB. In other words, the service implementation module ABB is the controller of the Service Component layer.

3.8.7 Relationship Diagram across Other Layers

Figure 17 uses a UML component diagram to illustrate a static view of the relationships for the ABBs across layers. Various layers identified in the S3 are represented as packages; ABBs are represented as components.

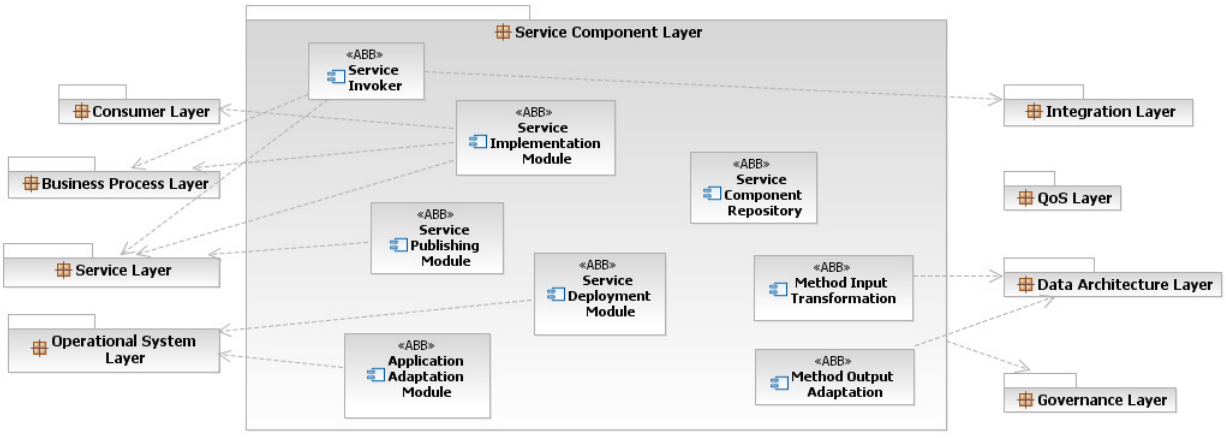

Figure 17. *Component Relationship Diagram – ABBs in the Service Component layer across layers.*

Certain relationships exist between the ABBs in the Services Component layer with other layers:

- A *service implementation module* ABB needs to interact with the Consumer layer, the Business Process layer, and the Service layer.
- A *service invoker* ABB needs to interact with the Business Process layer, the Service layer, and the Integration layer.
- A *service publishing module* ABB needs to interact with the Service layer.
- A *service deployment module* ABB needs to interact with the Existing Application Asset layer.
- An *application adaptation module* ABB needs to interact with the Existing Application Asset layer.
- A *method input transformation* ABB needs to interact with the Data Architecture layer.
- A *method output adaptation* ABB needs to interact with the Data Architecture layer.
- All ABBs in the Service Component layer interacts with the Governance layer.

3.9 Options and Design Decisions

There are several technology alternatives for the implementation of the service components. The selection criteria employed when choosing the technology should be a balance of the following criteria:

- Capability: the capability to realize the value proposition of service components and to realize the required behaviors of a given service.
- Familiarity: whether the capability of using the technology already exists in the organization.
- Strategic: whether the technology inline with the technology roadmap of the organization.
- Manageable: whether the technology allows for the effective management of service components with respect to the KPIs defined.

Often the selection requires the prioritization of these criteria. One alternative may offer features that are suited to the needs of a particular service component (e.g. message mediation) but not offer the management capabilities that other service components require (e.g., high availability).

Some of the common architectural decisions in this layer include:

- Should integration always occur through an integration broker? Can it be performed inside a service component?
- Should collaborating service components consume each other through the Services layer or through a platform-specific interface? (e.g., EJB<- ->EJB or EJB<- ->Service)
- Where are transformations performed? In service components or in the Integration layer?

- Should component implementations be portable?

3.10 Types of Composition

See Appendix A Composition and Flexibility.

3.11 Activities

The major activities performed in this layer can be organized into four categories: model, assemble, deploy, and manage.

3.11.1 Model:

- Perform IT Asset analysis to identify Existing Application Asset layer features that can be leveraged for a given service component;
- Perform Service Oriented Modeling and Architecture(SOMA) and Business Process Modeling (BPM); and
- Create a machine-readable service description (e.g., using WSDL or BPEL).

3.11.2 Assemble:

- Select service component implementation technology;
- Create a service-compliant service component skeleton;
- Develop service components;
- Instrument the service components for monitoring; and
- Test service components (e.g., functional test, regression test, load test, etc.)

3.11.3 Deploy:

- Configure non-functional requirements (NFRs) on the service components; and
- Register a service in a service registry.

3.11.4 Manage:

- Monitor salient characteristics of a service (e.g., latency and request failures); and
- Analyze service component behaviors and make improvements in the implementation.

4 Layer 3: Service Layer

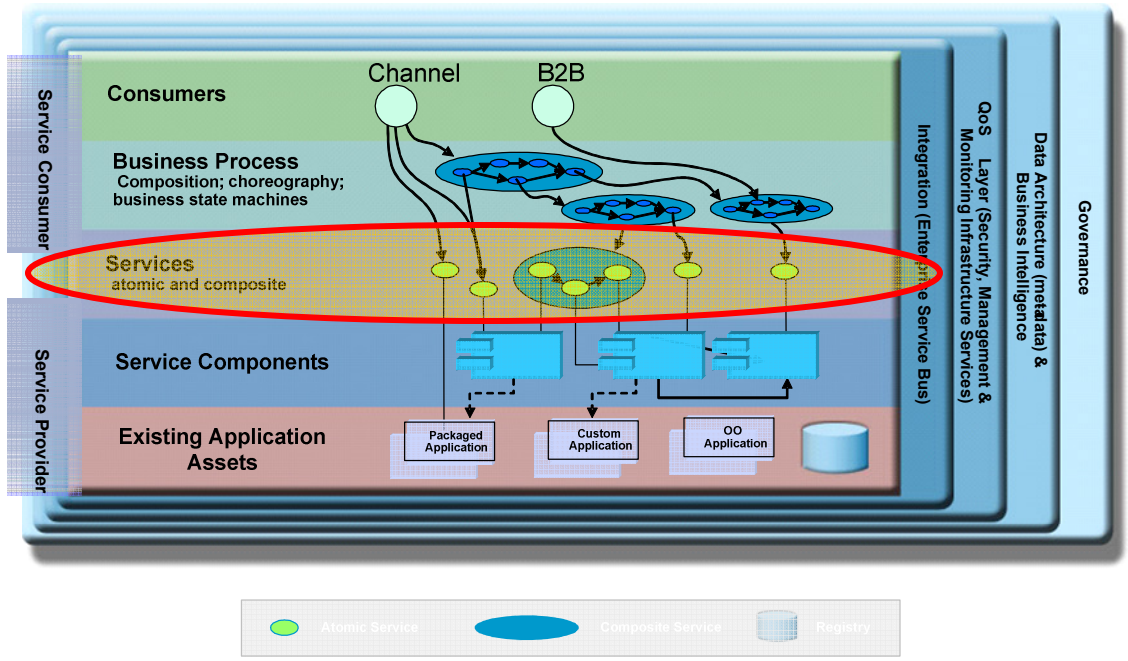

Figure 18. Services Layer in S3

4.1 Layer Description

As shown in **Figure**, the Service layer extends the known triangular SOA model into a comprehensive logical layer enabling and facilitating service registration, decomposition, discovery, binding, interface aggregation, as well as service life cycle management. The Service layer introduces a concept of *service cluster*, which refers to a collection (category) of Web services conceptually serving a common business function. These Web services may be published by different service providers, and differentiated with each other by specific features. For example, a shipping service cluster may include various services such as UPS, USPS, and FedEx. For each service cluster decomposed from a business process in the Business Process layer, the Service layer is responsible for locating appropriate service provider and binding to the target Web service interface. In addition, the Service layer is also in capable of aggregating multiple service interfaces into a new service.

This layer consists of all services within an SOA solution. It consists of service descriptions, policies, versions of services, as well as SOA management descriptions and attachments that categorize or show service dependencies.

Exposed services are collected in this layer. They can be "discovered" and invoked, or possibly choreographed into a new services composition. Services are "functions" that are

accessible across a network via well-defined interfaces of the "Services layer". The service layer also provides mechanism to take enterprise-scale components, business unit-specific components and in some cases project-specific components and externalize a subset of their interfaces in the form of service descriptions. So the components provide services through their interfaces. The interfaces get exported out as service descriptions in this layer, where services exist in isolation (atomic) or as composite services.

This layer contains the contracts (service descriptions) that bind corresponding providers and consumers. Services are offered by service providers and are consumed by service requestors. It is important to note that services can be mediated by service brokers to provide transformation, aggregation (aggregating information from finer-level services), auditing, metering, and so on.

The services defined within this layer are not a product of SOI (Service Oriented Integration), though there may be some serendipitous benefit from SOI.

As one example enabling technology, the services in the Service layer can be identified using the Service Identification activities defined in SOMA [4] through three complementary techniques: domain decomposition, existing asset analysis and goal-service modeling. They represent the heart of the SOA value proposition – improved agility via the decoupling of business and IT. The quality of these service identifications and specificiations (see service litmus tests in the Specification phase of the SOMA method) will have a significant impact on the benefit of a given SOA effort.

Services are accessible independent of implementation and transport. This allows a service to be exposed consistently across multiple customer-facing channels such as Web, IVR, Seibel client (used by Customer Service Rep), etc. The actual transformation of response to HTML (for Web), Voice XML (for IVR), XML string (for Siebel client) can be done via XSLT functionality supported through ESB transformation capability in the Integration layer.

Note that a service can encapsulate a process and that a service component can be a service consumer.

4.1.1 Service Cluster

The Service layer leverages a concept of *service cluster* [8]. A Web service cluster refers to a collection (category) of Web services serving a common business function. These Web services may be published by different service providers, and differentiated with each other by specific features. For example, a generic shipping service can be considered as a conceptual service cluster. For the same shipping purpose, many service providers exist, such as UPS, USPS, FedEx, and so on. For the same service provider, say UPS, it may also provide a variety of shipping services with different time frames and guarantees. For example, a UPS service can provide overnight delivery, the second day delivery, 2-day delivery, 3-day delivery, 5-day delivery, and 1-week delivery. All of these types of shipping services can be considered as actual implementations of a conceptual shipping service

cluster. As a best practice, in this layer, services should always be categorized into service clusters based upon some business functions, e.g., reporting services and purchase order management.

A business process only cares about the level of Web service clusters, instead of individual Web services. The reason is apparent. A selected Web service may become unavailable at invocation time; therefore, it should be replaced by another available Web service in the same Web service cluster without being known by the users of the corresponding business process. Each potential Web service is kept in the logical Service layer.

The Service layer performs both top-down and bottom-up service-level handling. From top-down direction, the layer provides facilities to locate actual Web service interfaces for business processes; from bottom-up direction, the layer provides facilities to expose Web service interfaces to the outer world.

The actual top-down mapping from business processes into real Web services is handled by the Service layer. As discussed in the section of the Business Process layer, a business process is decomposed into service clusters. Then for each service cluster identified, the Service layer is responsible for: (1) finding appropriate service provider and service consumer, (2) locating at which the target service resides and accumulating other requirements such as access control, (3) binding to the target Web service interface.

From bottom-up, the Service layer exposes Web service interfaces for service components. Note that one service component may be exposed into different formats by different service interfaces. As shown in Figure 19, one service component is exposed into four service interfaces. In other words, one service component may implement multiple services defined in the Services layer. Therefore, the number of services in the Service layer may exceed the number of service components in the Service Component layer. In the Service Component layer, multiple service components may be used by a wrapper that implements the interface defined by the Service layer.

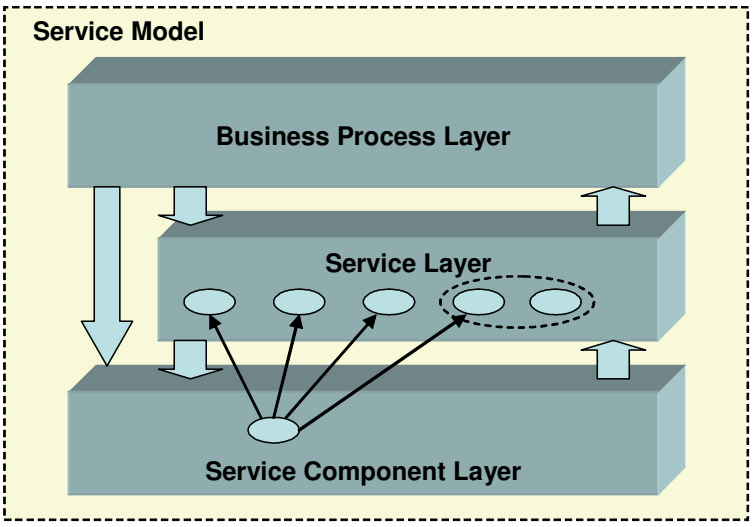

Figure 19. One-to-Many relationship for a service component to multiple service interfaces.

4.1.2 Composite Service

The Service layer may also perform some high-level interface-level service aggregation. Figure 19 illustrates two categories of services: individual services and composite services. An individual service is shown as an individual oval; while a composite service is shown as a dotted oval comprising several individual services. Here we define the two concepts. An individual service refers to an atomic service that does not depend on other services. A composite service refers to a service depends on more than one individual services. The concept of a composite service is illustrated in Figure 20.

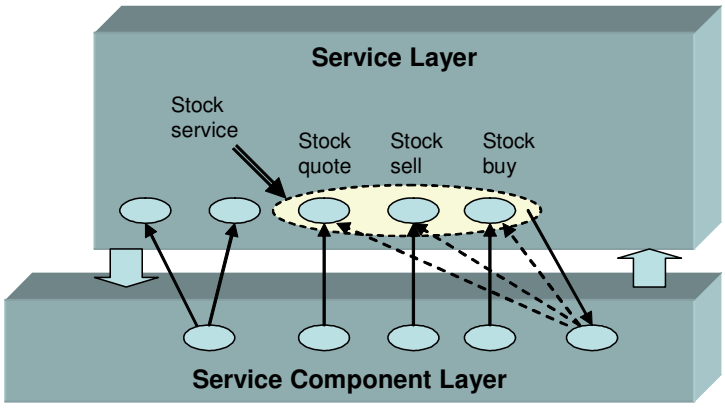

Figure 20. Concept of a composite service.

Figure 20 shows a composite service *stock service*. Three individual services exist: stock quote, stock sell, and stock buy. Each of these services is wrapped from corresponding service

component from the Service Component layer. Each of these three services represents one stock-related activity: quote stock price, sell a stock, or buy a stock. The *stock service* aggregates the three individual services into a composite service with new service interface. Note that this kind of service aggregation is based upon interfaces only (e.g., some packaging), without business logic involved. As shown in Figure 20, there is no control flow between the three aggregated services. As a matter of fact, each of the three services may become a separate operation of the aggregated WSDL interface. It should also be noted that a composite service definitely contains a corresponding single service component, which we called technical service component, in the Service Component layer to implement the interface aggregation. As shown in Figure 20, the composite service *stock service* comprises a technical service component to invoke the three services (service interfaces) from the Service layer.

Note that a composite service must comprise more than one service (interface); otherwise, it should be considered as an individual service. As shown in Figure 21, the service is implemented by a service component in the Service Component layer, which is in turn implemented by two legacy systems in the Operational System layer. Since only one service (interface) from the Service layer is involved, the service should be considered as an individual service.

In summary, a composite service must include a composite interface and an implementation through a service component, which in turn invokes more than one service in the Service layer and/or other technical component(s) such as SAP or legacy systems.

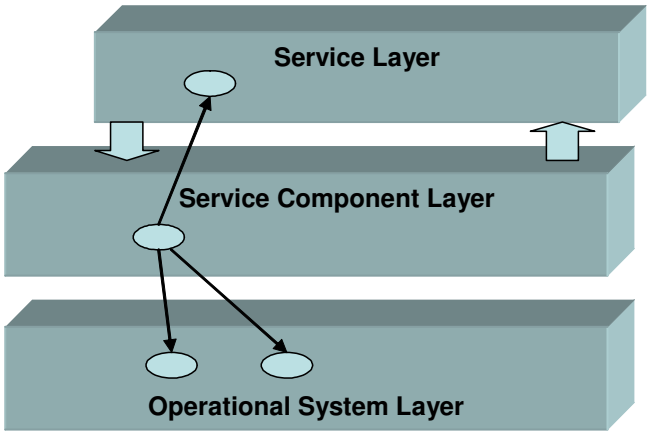

Figure 21. Concept of an individual service.

Note that we use the term "aggregation" instead of "composition" as in the Business Component layer. The difference is apparent. The service composition refers to integrating services into a business process with business logic. It should be noted that with service

composition, there is business flow between services, which can be represented by business flow description language such as BPEL4WS. On the other hand, the service aggregation refers to turning services into individual operations in a new WSDL interface without adding any business logic between them. There is no business flows defined between the services to be aggregated. In other words, service aggregation is merely a new way to release existing services. Note that a service composition may not result in a composite service.

In general, there are two distinct groups of services within a given organization: external services and internal services. The first group is also known as common business services. This refers to the set of services that are business-aligned services fulfilling an organization's business processes and goals. These services can be tied back to business processes. Business services can be exposed to other lines of business or to the outside world of partners and the SOA ecosystem. The second group internal services contain those that address IT integration and infrastructure problems. Typically, one organization does not necessarily apply the same rigor in the identification and exposure of this type of services. Although these internal services are designed to meet non-functional or quality of service requirements, they may not directly represent an important aspect of the SOA value proposition.

4.2 Value Proposition

Consolidate service descriptions, policies, management and versions including governance of services and their dependencies.

4.3 Characteristics

This layer captures a business IT-aligned service model and renders it as WSDL.

The service specification includes the input and output message formats. These are the Data Architecture aspects of this layer: i.e., the intersection between the Services layer and the Data Architecture layer.

An implication of this characteristic is that only types obtainable from a service model can be explicitly defined on services.

A service can be instantiated in a Service Catalog (Service Registry)

- Functionality
  - A Service Specification includes the definition of the messages exchanged
    - e.g., WSDL : xsd: AnyType is not acceptible here
- Policy and
- Management information will be found in the registry.

4.4 Scenario: Transfer Funds Service

Under construction.
"Service Catalog includes GetCustInfo, but type returns are not IVR types"
"You now need a new service to convert the output of GetCustInfo into my desired format"

4.5 Key Performance Indicators

Under construction.

4.6 Architectural Building Blocks

As shown in Figure 22, we identify thirteen fundamental Architectural Building Blocks (ABBs) in the Service layer: (1) service cluster, (2) service, (3) services registry, (4) service interaction manager, (5) service binder, (6) policy, (7) access control, (8) state manager, (9) service provider, (10) service aggregator, (11) service consumer, (12) service broker, and (13) interface aggregator.

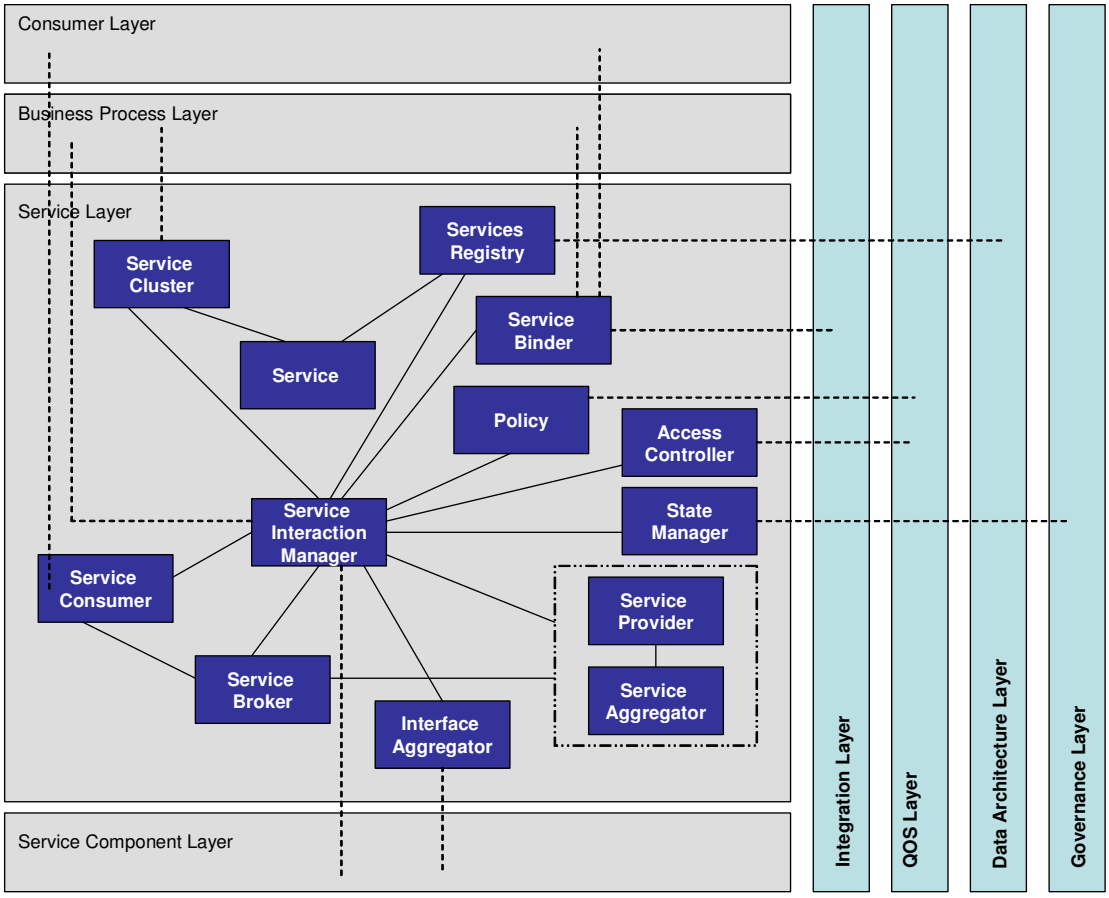

Figure 22. *Service layer ABBs.*

4.7 Descriptions of ABBs

This section describes in detail each ABB in terms of its responsibilities.

4.7.1 Service Cluster

A *service cluster* building block represents a conceptual-level concept of services. It does not represent a specific service; instead, it represents a cluster (i.e., set) of services offering the same functionalities. Using the Java terminology, a service cluster can be viewed as an interface of a set of implemented classes (i.e., embodied services).

4.7.2 Service

A *service* building block represents a published service that offers certain functionalities. Typically, a service is published to a service registry, so that service consumers can search for interested ones. A service is typically represented in a standard description language (e.g., Web Services Description Language (WSDL)) describing its accessible interfaces (e.g., function or method signatures).

4.7.3 Service Registry

A *service registry* building block refers to a service repository that stores the information of a set of published services for service consumers to query from. Typically, this information includes the service interfaces (e.g., service names and accessible operations with invocation signatures) and maybe some service providers' information (e.g., business name, contact information, and development team information). Typically, the published services' information is organized by categories to facilitate service discovery. A service consumer queries a service registry for interested services, and then uses the found network addresses of the service providers to invoke the services from the corresponding service providers.

4.7.4 Service Interaction Manager

A *service interaction manager* building block is the central managing unit of the Service layer. *As shown in* Figure 22, it manages all ABBs identified in the Service layer. The identification of this ABB not only realizes a centralized management of the Service layer, but also enables loosely coupling between ABBs in the layer in order to realize flexibility, extensibility, and reusability of ABBs.

4.7.5 Service Binder

A *service binder* building block finds proper services offered by service providers and is responsible for binding to the actual services.

4.7.6 Policy

A *policy* building block carries policies that regulate the behaviors of identified ABBs in the Service layer. Typically, services, service providers, and service consumers each carry specific policies to formalize their behaviors and enable effective management.

For example, policies for services can cover both functional and non-functional aspects of implementing services based upon a common set of ground rules, which may be derived from multiple sources, such as IT best practices, customer-specific requirements, and IBM internal practices promoting asset reuse. For example, a policy covering a functional aspect may specify a particular standard messaging format to be used by a group of services to ensure future reusability of these services. An example of a policy covering a non-functional aspect may specify a certain level of availability of a particular service over a certain period of time.

Policies for service consumers define specific constraints or patterns of how the business entity may interact with other business entities, services, or business processes. For example, customer A may not want to use services provided by company B.

Policies for service providers define patterns and constraints that capture business relationships between the service providers with other entities and services. For example, a service provider A may be form an alliance with another service provider B, so that they tend to collaborate with each other. The detailed relationship modeling can be found in [3].

4.7.7 Access Controller

An *access controller* building block provides authentication/authorization capabilities (enabled through the Security layer) to be used by the service interaction manager to grant proper access privileges for published services in the Service layer.

4.7.8 State Manager

A *state manager* building block provides stateful management of identified ABBs in the Service layer. A comprehensive business transaction typically requires a stateful environment instead of simple stateless services. A state manager allows an identified ABB to carry a set of attributes with values across conversational events.

4.7.9 Service Provider

The *service provider* building block represents the business entity who owns one or more services defined in this layer. Specifically, this building block refers to owners of primitive services. Owners of composite services are referred to as service aggregators, which will be identified as a different ABB. A service provider typically performs two key operations: (1) publish a service in a service registry and (2) provide a service invocation interface for deployed implementation module.

4.7.10 Service Aggregator

The *service aggregator* building block represents the business entity who owns one or more composite services defined in this layer. In contrast with normal service providers who offer individual services, service aggregators aggregate published individual services into

composite services. It should be noted that, as shown in Figure 22, from a service consumer's perspective, a service aggregator is a normal service provider. Therefore, the service aggregator ABB can be viewed as a subclass of the service provider ABB. It inherits the interfaces and operations defined in the service provider ABB with an addition operation that aggregates existing services.

4.7.11 Service Consumer

A *service consumer* building block represents the business entity that invokes and uses one or more services defined in this layer. A service consumer may be an entire enterprise, a department, a group of people within the enterprise identified by a particular role, or an individual. The specific granularity at which this term may be used is ultimately dependent on the nature and scope of the corresponding SOA architecture. A service consumer typically performs two key operations: (1) search for a service in a service registry and (2) bind to a specific service and consume it.

4.7.12 Service Broker

A *service broker* building block acts as a broker between service providers (including service aggregators) and service consumers. In more detail, it helps a service consumer to discover interested services and help the service consumer to locate corresponding service providers.

4.7.13 Interface Aggregator

An *interface aggregator* building block performs interface-level service aggregation. Service interfaces can be aggregated into a new service interface, without any business logic involved. Each component service becomes an individual operation in the aggregated WSDL definitions.

4.8 Dependencies and Interactions (patterns)

The relationships between the Service layer and other layers are depicted in this table.

Table 1: Service Layer Relationships

Consumer	Business Process	Services	Service Components	Existing Application Assets	Integration	Quality of Service	Data Architecture	Governance
Consumer can access /invoke services	Processes implement their activities through service invocation	Service Descriptions, Policies, management, versions, messages	Services are realized by service components	Existing application assets can be leveraged either by wrapping them with service interfaces	Invocation of services can be direct or through the Integration layer (e.g., using an ESB within	Policy Descriptions; SOA management descriptions; monitoring; events to invoke services	Input, output Messages	Service Registry; Versioning; [Policies]

				or service components that realize services leverage them	the Integration layer)			

One-to-Many relationship for a service component to one or multiple service interfaces:

- The actual top-down mapping from business processes into real Web services is handled by the Service layer.
  - A business process is decomposed into service clusters.
  - Then for each service cluster identified, the Service layer is responsible for: (1) finding appropriate service provider and service consumer, (2) locating at which the target service resides and accumulating other requirements such as access control, (3) binding to the target Web service interface.

- From bottom-up, the Service layer exposes Web service interfaces for service components.
  - One service component may be exposed into different formats by different service interfaces.
  - One service component is exposed into one or multiple service interfaces. In other words, one service component may implement multiple services defined in the Services layer.
  - Therefore, the number of services in the Service layer may exceed the number of service components in the Service Component layer.
  - In the Service Component layer, multiple service components may be used by a wrapper that implements the interface defined by the Service layer.

4.8.1 Relationship Diagram within the Service Layer

Figure 23 uses a UML component diagram to illustrate a static view of the relationships between the ABBs within the Service layer. All identified ABBs are represented as components in the diagram, with "ABB" as stereo type. The Service layer is represented as a package containing all identified ABBs.

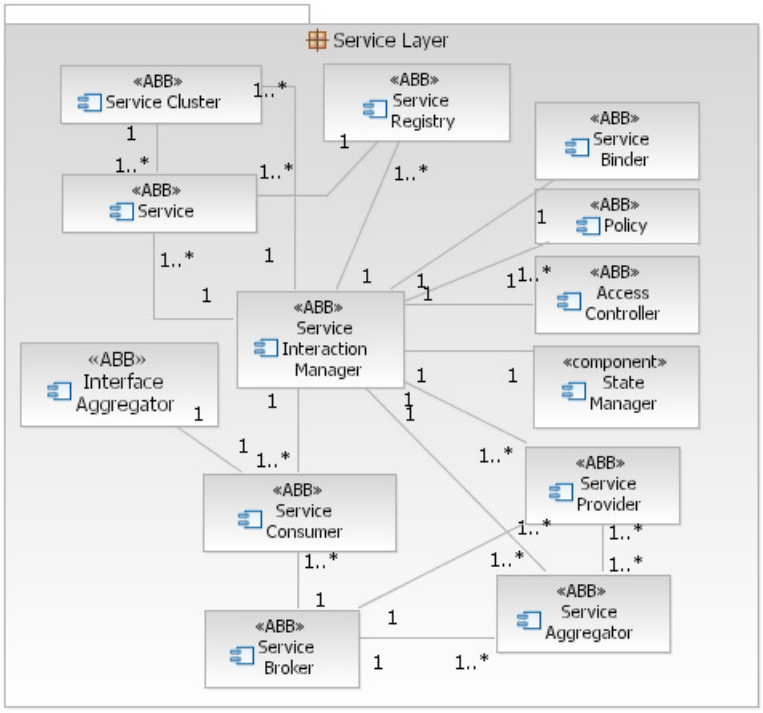

Figure 23. *Component Relationship Diagram – ABBs within the Service Layer.*

Certain relationships exist between the twelve types of ABBs:

- Each service cluster, service, service registry, service binder, policy, access controller, state manager, service provider, service aggregator, service broker, service consumer, and interface aggregator needs to interact with and under control of the service interaction manager. The relationship between the service interaction manager and its related ABBs is a one-to-many relationship, meaning that there is only one service interaction manager ABB in a Service layer that controls all other ABBs.
- There is a one-to-many relationship between a service cluster ABB and a service ABB, because a service cluster represents a set of services with the same functionalities.
- There is a many-to-one relationship between a service ABB and a service registry ABB, meaning that a service registry is a repository with multiple services registered.
- There is a many-to-one relationship between a service consumer ABB and a service broker ABB, meaning that a service broker helps multiple service consumers.
- There is a many-to-one relationship between a service provider ABB and a service broker ABB, meaning that a service brokers has access to multiple service providers.
- There is a many-to-one relationship between a service aggregator ABB and a service broker ABB, meaning that a service broker has access to multiple service aggregators.
- There is a many-to- many relationship between a service provider ABB and a service aggregator ABB, meaning that a service aggregator creates a new service from multiple service providers, and a service provider's service can be used by multiple service aggregators as a service component.

Theoretically, other relationships can exist. For example, a service consumer can directly binds to a service provider for an interested service. However, they are not recommended to interact with each other directly in an SOA solution architecture. In order to realize higher reusability, flexibility, and extensibility of identified ABBs, these ABBs should be loosely coupled and only interact with each other through other ABBs within the Service layer (i.e., service interaction manager ABB) or other layers (e.g., the Integration layer or the QoS layer).

4.8.2 Relationship Diagram across Other Layers

Figure 24 uses a UML component diagram to illustrate a static view of the relationships for the ABBs across layers. Various layers identified in a typical SOA solution architecture are represented as packages; ABBs are represented as components.

Certain relationships exist between the ABBs in the Services layers with other layers:

- A *service interaction manager* ABB needs to interact with the Business Process layer and the Service Component layer.
- An *interface aggregator* ABB needs to interact with the Service Component layer.
- A *service cluster* ABB needs to interact with the Business Process layer.
- A *service registry* ABB needs to interact with the Data Architecture layer.
- A *service binder* ABB needs to interact with the Business Process layer, the Consumer layer, and the Integration layer.
- A *service consumer* ABB needs to interact with the Consumer layer.
- A *policy* ABB needs to interact with the Quality of Services (QoS) layer.
- An *access controller* ABB needs to interact with the Quality of Services (QoS) layer.
- A *state manager* ABB needs to interact with the Governance layer.

In theory, a *service interaction manager* ABB can directly interact with the Existing Application Asset layer to packaged applications. However, in a typical SOA solution architecture, we strongly recommend that they do not directly communicate with each other. Instead, they should indirectly interact with each other through other layers, such as the Integration layer or the Data Architecture layer.

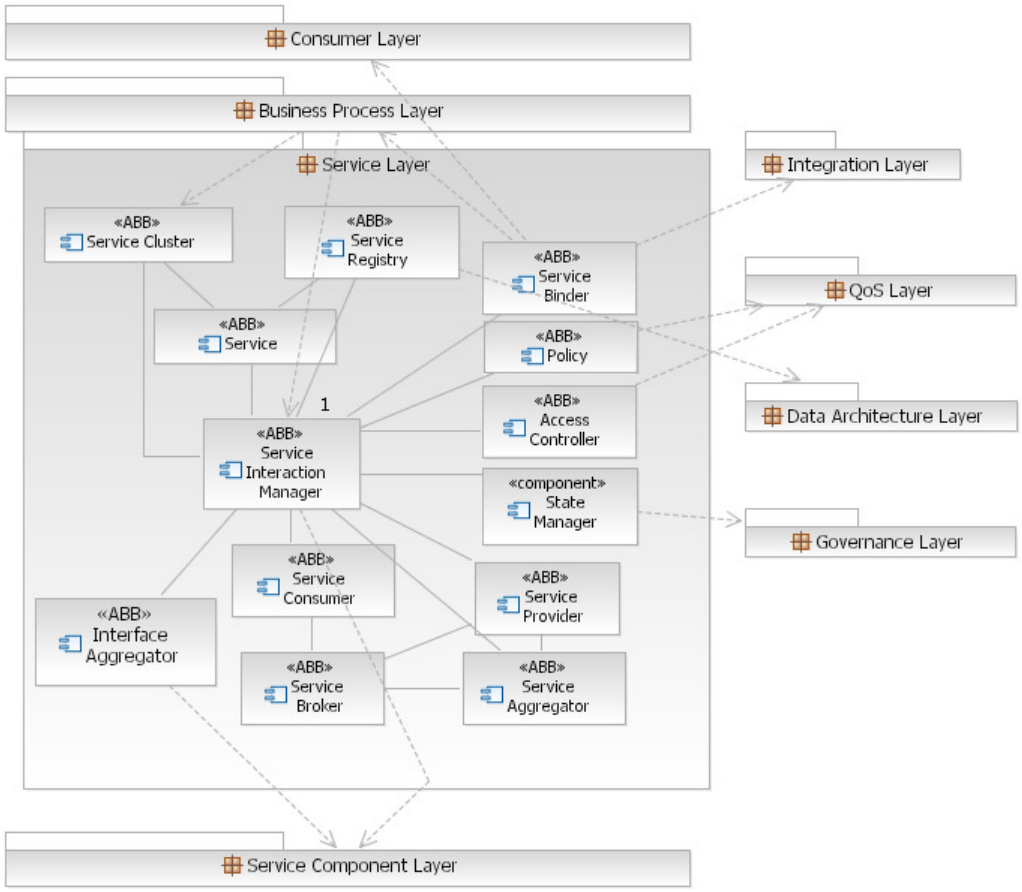

Figure 24. Component Relationship Diagram – ABBs in the Service layer across layers.

4.9 Options and Design Decisions

The major issue requiring design decision is: format conversion (between providers and consumers)

- What part of format conversion should be delegated to the Integration layer and what part should be done in this layer?
  - What are the responsibilities of the Service layer with regard to format conversion?
  - Provider specific, standards-focused conversions are done in this layer inside
  - This layer if independent of the formats expected by the consumer.

Options 1: Provider-specific transformations versus consumer-specific transformations.

Options 2: Define the distinction between a service interface and a service description.

4.10 Activities

The results of SOMA Service Model map to this layer.

During Design Time:

1. Publish Service Specification or Service Descriptions into Registries
2. Find and Select services based on functional and non-functional (SLAs) requirements
3. Compose services based on the business process, logic, policies
    - ➔ The final result of the composition is a service oriented process flow described in the Business process Layer

During Run Time: invocations to this layer need to be bound to a specific version of the service that can be found in the service registry associated with the governance aspect of this layer.

This includes outward-facing business services as well as inward-facing (usage is inside the company) IT services. Services can be simple (atomic) or composite. When a service is composite and you only wish to expose the entire composition as a service and are not interested in exposing the composition then the composite service will lie within this layer. If you wish to expose the constituent services of the composition then you can assign that composition to the business process layer.

4.11 Composition and the Component ABB

<see appendix A>>

5 Layer 4: Business Process Composition Layer.

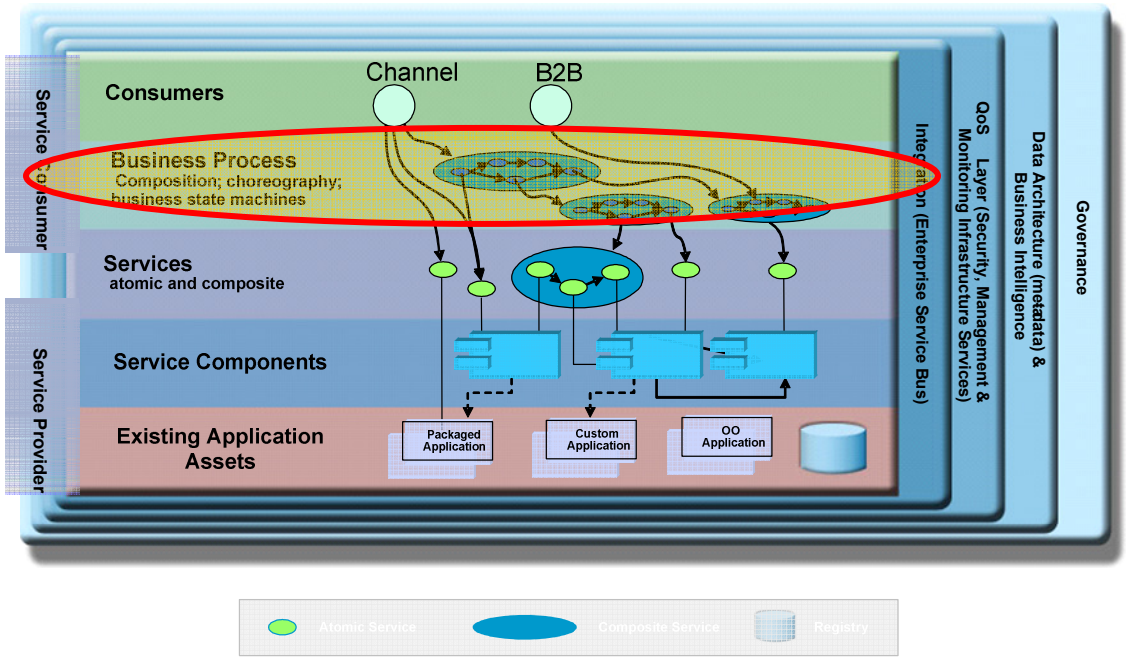

Figure 25. Business Process Layer in S3

5.1 Introduction

As shown in Figure 25, the Business Process layer leverages the Service layer to quickly compose and choreograph services and to coordinate business processes to fulfill customer requirements. Visual flow composition tools such as WebSphere WBI-Modeler can be used for design of application flow.

In more detail, the Business Process layer performs three-dimensional process-level handling: top-down, bottom-up, and horizontal. From the top-down direction, this layer provides facilities to decompose business requirements into tasks comprising activity flows, each being realized by existing business processes, services, and service components. From the bottom-up direction, the layer provides facilities to compose existing business processes, services, and service components into new business processes. From the horizontal direction, the layer provides services-oriented collaboration control between business processes, services, and service components.

The decomposition of a business process first decomposes it into smaller tasks; then each task is mapped into service clusters (i.e., conceptual services) that will be realized by actual Web services in the Service layer. In other words, the layer provides facilities to decompose a

business process into conceptual services that fulfill business functions, which are called service clusters that will be discussed in detail in the Service layer.

5.2 Layer Description

The Business Process layer handles all business logic regarding service composition and decomposition. For service composition, this layer leverages the Service layer to quickly compose and choreograph services and to coordinate business processes to fulfill customer requirements. For service decomposition, this layer provides facilities to decompose business requirements into tasks comprising conceptual service clusters, each being realized by existing business processes, services, and service components. It should be noted that the Business Process layer does not focus on individual business process representation, which can be fulfilled by workflow description languages such as Business Process Execution Language for Web Services (BPEL4WS). Rather, this layer focuses on building SOA solutions using business processes. For example, the layer may take 10 existing business processes and aggregate them into 3 big processes, while taking charge of the collaboration between them.

This layer covers the process representation, composition methods, and building blocks for aggregating loosely coupled services as a sequencing process aligned with business goals. Data flow and control flow are used to enable interactions between services and business processes. The interaction may exist within an enterprise or across multiple enterprises. This layer also includes information exchange flow between participants (individual users and business entities), resources, and processes in a variety of forms to achieve business goals. Most of the exchanged information may also include non-structured and non-transactional messages. The business logic is used to form service flows as parallel tasks or sequential tasks based on business rules, policies, and other business requirements.

From the data flow perspective, the business context and meta data are used to support the aggregation of services within an enterprise for business process orchestration or across multiple enterprises for business process choreography.

The lifecycle management for business process orchestration and choreography are also covered in this layer. In addition to the run-time process engine (e.g., WS4BPEL engine), this layer will cover all aspects of composition, collaboration, compliance, process library, process service, and invocation elements. The details will be described in the architecture building blocks section.

5.3 Value Proposition

Building process blocks on demand with reduced cost allows the supporting technology changes from being high volume/transactional to sophisticated but much smaller footprint applications. In fact, a business process captures the activities needed to accomplish a certain business goal. In today's business solutions, a business process has played a central role in bridging the gap between business and IT.

From the top-down approach, a business process can be defined by business people based on customers' requirements. In order to optimize the business process for better IT implementation, a business process should be componentized as reusable services that can be modeled, analyzed, and optimized based on business requirements such as QoS (historical data described in the QoS layer), flow preference, price, time of delivery, and customer preferences. From the bottom-up approach, after a set of assets is created, they could be leveraged in a meaningful business context to satisfy customer requirements. The flexibility and extensibility of services composition guided by business requirements and composition rules enable business process on demand for addressing different types of customer pin points by reusing services assets.

From interaction perspective, the Business Process layer communicates with the Consumer layer (a.k.a. presentation layer) to communicate inputs and results with role players (e.g., end user, decision makers, system administrator, etc.) through Web portal or business-to-business (B2B) programs. Most of the control flow messages and data flow messages of the business process may be routed and transformed through the Integration layer. The contents of the messages could be defined by the Data Architecture layer. The KPIs for each task or process could be defined in the QoS layer. The aggregation of services could be guided by the Governance layer. All the services should be represented and described by the Service Layer; and service components are represented by the Service Component layer.

From a technical perspective, dynamic and automatic business process composition poses critical challenges to researchers and practitioners in the field of Web services. First, business processes are driven by business requirements, which typically tend to be informal, subjective, and difficult to quantify. Therefore, it is critical to properly formulate the descriptive and subjective requirements into quantifiable, objective, and machine-readable formats in order to enable automatic business process composition. Second, existing Web services-based business process description languages do not adequately accommodate detailed requirement specification, which fact makes it difficult to create optimal business process compositions. Third, present Web services specifications generally lack a facility to define comprehensive relationships among business entities, business services, and operations. These relationships may be important to optimize business process composition. For example, suppose enterprise E1 needs to compose a business process including service S. Enterprises E2 and E3 both provides similar service S. However, there is a partnership between E1 and E2 that leads to a discount on services, and there is no partnership relationship between E1 and E3. If price is a requirement for party E1, this partnership between E1 and E2 needs to be taken into consideration in order to form the most appropriate business process. Fourth, nowadays more and more Web services are published to the Internet on the daily basis. How to clearly specify search requirements to discover the most appropriate Web services candidates remains a challenge. Fifth, a typical business process generally requires multiple Web services to collaborate in order to serve business requirements. Therefore, each service component not only needs to satisfy individual requirements, but also needs to coexist with other services components in order to best fit the overall composed business process. In other words, the entire business process needs to be optimized prior to execution.

In summary, the Business Process layer in the SOA Solution Stack plays a central coordinating role in connecting business level requirements and IT-level solution components through collaboration with the Integration layer, QoS layer, as well as the Data Architecture layer, the Services layer, and the Service Component layer. Addressing those challenging issues are being covered in this Business Process layer to further differentiate the proposed SOA Solution Stack with other conceptual reference models from other vendors.

5.4 Characteristics

Under construction.

5.5 Key Performance Indicators

The Business Process layer provides visibility control points and mechanisms for role players involved in a particular business process to check the status of activities and performance in different granularities. The performance metrics for business process-based service-to-service collaboration include KPIs, status, exceptions, business events, escalation processes or actions, etc.

For example, in a running business process, metrics and key performance indicators relating to business resources include:

- Operation efficiency metrics such as percentage of orders delivered to customers in full quantity at the specified time, or average credit approval cycle time;
- Key performance indicators for the value chain such as available technical resources for developing a new product, percentage of orders received electronically, percentage of customer calls taken on initial contact, and number of order input errors;
- Activity metrics such as the progress of a pricing dispute management conducted by multiple collaborators in the order-to-cash value chain (e.g., bank, service provider, and customer).

All these performance metrics can be embedded in the business process representations with annotations.

At design time, the performance metrics for a business process should focus on specifications of service flow rules, customer preferences, and business rules, which can be used for services discovery and selection. For example, flow rules specify rules for either parallel services or sequential services requested by customers. If parallel services are desired, the flow rules will specify parallel numbers and parallel tasks. If sequential services are desired, the flow rules will specify sequential numbers and sequential tasks in a business process.

There are five major aspects to define and specify business requirements: (1) service name, (2) preference, (3) business rules binding, (4) service relationship, and (5) event (for binding as well).

Service names specify a set of particular services to be used in a business process. Preferences include a set of name and value pairs, such as the preferred UDDI registry name and its location link.

Business rules specify how the various steps of a business process can be encoded in the form of reaction rules. More specifically, business rules govern the selection of Web services for optimal business processes. In more detail, a business rule is defined as a 5-tuple (business rule number, relevant service name, priority, condition, behavior). The element of condition predefines cost, time, benefit or service bonus, quality of service, and specific or preferred services. The element of behavior specifies under which circumstances a specific service should be selected. Service relationships describe the business relationships between service providers. For example, if service S1 is selected together with service S2, the combined cost is less than the sum of their costs, which affects the overall cost in a business process composition. A relationship is defined as a four-element tuple (relationship number, source service, target service, XML-based definition of the relationship).

In addition, a business process may include multiple operations that conform to a certain invocation-sequencing rule. The event list defined in the performance metrics for designing a business process is used to capture the Event-Condition-Action (ECA) mapping for event-driven business process composition. A binding event associated with these operations in a Web service triggers an event action to be performed for evaluating the service selection. An event is defined as a four-element tuple: (event name, event queue number, event condition, event action).

Finally, the extensible structure of the representation of performance metrics enables the import of existing XML files, such as FlowXML file, BusinessRuleXML file, PreferenceXML file, and ECA-XML for business flow, business rules, preferences, and event-action mappings. In other words, the performance metrics representation in business process layer not only acts as a container of these existing XML formats, but also carries the annotation information among the objects listed in different XML files.

5.6 Architectural Building Blocks

As shown in Figure 26, we identify ten fundamental Architectural Building Blocks (ABBs) in the Business Process layer: (1) process decomposition module, (2) service composition module, (3) process collaboration controller, (4) data flow, (5) process compliance module, (6) process repository, (7) process service adapter, (8) access control, (9) configuration rule, and (10) state management module.

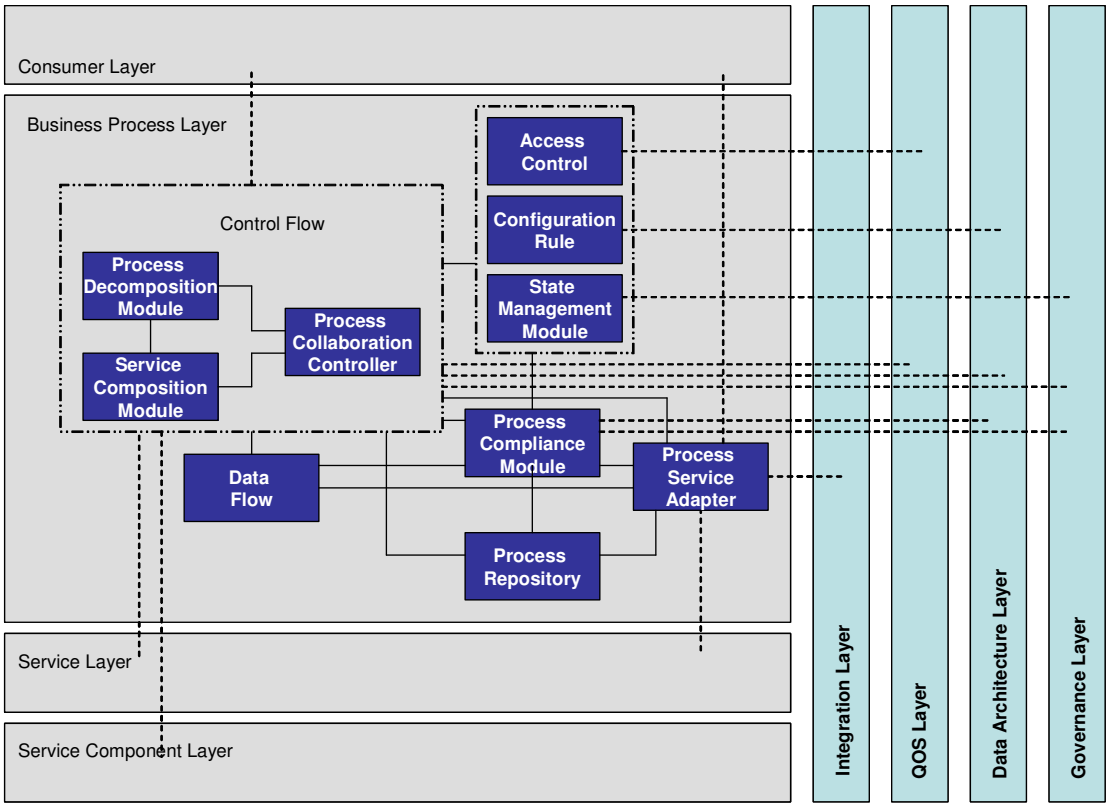

Figure 26. *Business Process layer ABBs.*

5.7 Component Descriptions – Business Process Layer

This section describes in detail each ABB in terms of its responsibilities. As shown in Figure 26,, process decomposition module ABB, service composition module ABB, and process collaboration controller ABB belong to a generic control flow ABB. Access control ABB, configuration rule ABB, and state management module ABB belong to a generic utility ABB.

5.7.1 Process Decomposition Module

A *process decomposition* building block is responsible of dividing a business process into smaller units, each being able to match to a service cluster (discussed in detail in the Service layer).

5.7.2 Service Composition Module

A *service composition* building block covers all the aspects of aggregating services as a business process. It may go through service or process discovery phase to reduce the number of available Web services candidates, guided by performance metrics derived from customer requirements. Historical QoS data will also be considered to select services. It should be noted that a combination of the locally optimal services may not be the globally optimal one;

therefore, this building block should also contain a flow optimization facility. The final aggregation of services is represented as a BPEL or other flow markup languages.

5.7.3 Process Collaboration Controller

A *process collaboration* building block handles the collaboration patterns between services and processes. Commonly used terms are defined in a dictionary or ontology. Business logic is used to define the sequences and contents of messages exchanged in the service-oriented collaboration environment. In more detail, message exchange patterns serve as business protocols that include predefined reusable messages, such as Request for Quote (RFQ), Request for Service (RFS), and a sequence of these messages. Regarding each message, it may include one or more message units defined by schema. For example, a RFQ message may include an RFQ request message unit, an RFQ acknowledgement message unit, and an RFQ response message unit. All the message units are created to carry information for business resources (e.g., site, organization, project, task, process, document, and role player) for achieving a certain business goal.

A process collaboration controller should also contain process chorography resources, which provide the vocabulary for business collaborations. Example resources include: project, task, requirement, opportunity, and virtual team. In general, they can be represented using WS-Resources standard. Process chorography resources can also contain cross-links to each other by endpoint references. For example, a project may refer to a set of tasks; a task may refer to a set of requirements. A set of relationships can also be defined for composition, aggregation, inheritance, and association for WS-Resources.

5.7.4 Data Flow

A *data flow* ABB is responsible for managing data transactions and transformations between services and processes.

5.7.5 Process Compliance Module

A *process compliance* building block is responsible for ensuring that a business process complies with predefined requirements. This ABB may cover mapping, monitoring, management, law enforcement, and versioning. Mapping means that a business process is aggregated by existing services. This functionality is analogical to using Partner Interface Processes (PIPs) in RosettaNet, which is an industry standards-based business process mapping. A business process should have embedded performance metrics or KPIs, so they can be monitored at run time. In addition, run-time exceptions need to be handled properly. Law enforcement means that a business process should comply with predefined rules or policies. For example, the annual reports of a firm must comply with the financial reporting disclosure requirements of the Sarbanes-Oxley Act (SOX) management report on internal control. In other words, all financial reporting-related business processes should be SOX-compliant. Versioning means that backward compatibility should be assured.

5.7.6 Process Repository

A *process repository* building block stores a set of business processes in a retrievable asset repository. It can be categorized by various criteria, such as by business functions or by vendors.

5.7.7 Process Service Adapter

A *process service adapter* building block is used to externalize a business process as a Web service, so it can be discovered and used as a common service in the Service layer.

5.7.8 Access Control

An *access control* building block is responsible for authorization and authentication of available business processes.

5.7.9 Configuration Rule

A *configuration rule* building block is responsible for hosting rules that dictate how the ABBs can be configured based on various scenarios. This unit allows the use of only appropriate ABBs. It should be noted that this configuration capability will not be effective if the ABBs are coarse grained, as it will reduce flexibility. On the other hand, if ABBs are fine grained, they will be more flexible to be configured based on specified rules. This configuration can be handled in the following two ways. The first is through template-based configuration, where a user can select a specific template based on the corresponding service request scenario. The system will select all the rules associated with this template and configure the ABBs to support the rules. This approach requires that scenario templates need to be created and stored in a repository to be selected when needed. The second is through dynamic template creation, where a user selects certain characteristics and the system will determine the appropriate rules and configure using relevant ABBs at run time. For instance, user may specify the following two characteristics: - data is static (i.e., does not change during an interaction) and user should not be challenged multiple times. The first characteristic implies that a cache ABB is needed; the second one implies the necessity of using a proper security token mechanism. These characteristics will be used during the run time to configure appropriate ABBs.

5.7.10 State Management Module

A *state management* building block is responsible for recording and tracking business process interactions. These data typically are maintained for the duration of an entire business process.

5.8 Dependencies and Interactions (patterns)

5.8.1 Relationship Diagram within the Business Process Layer

Figure 27 uses a UML component diagram to illustrate a static view of the relationships between the ABBs within the Business Process layer. All identified ABBs are represented as components in the diagram, with "ABB" as stereo type. The Business Process layer is represented as a package containing all identified ABBs. In order to better organize and illustrate the relationships between the ten identified ABBs, two generic ABBs are identified, as shown in Figure 27. The first is the *control flow* ABB, which is the super ABB for the three ABBs: process decomposition ABB, service composition ABB, and process collaboration ABB. The second super ABB is the Utility Helper ABB, which is the super ABB for the three ABBs: access control ABB, configuration rule ABB, and state management module ABB. With the aid of the two super ABBs, relationships between an ABB and a super ABB implies the corresponding relationships between the ABB and all the sub-ABBs. For example, as shown in Figure 27, the one-to-one relationship between process compliance ABB and control flow ABB implies three one-to-one relationships: (1) a one-to-one relationship between process compliance ABB and process decomposition ABB, (2) a one-to-one relationship between process compliance ABB and service composition ABB, and (3) a one-to-one relationship between process compliance ABB and process collaboration controller ABB.

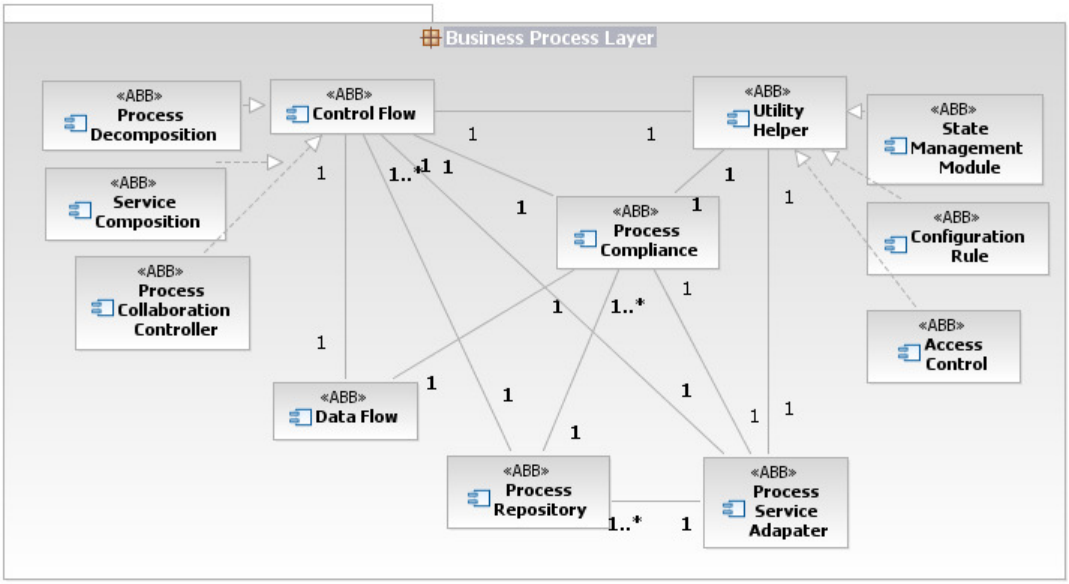

Figure 27. *Component Relationship Diagram – ABBs within the Business Process Layer.*

Certain relationships exist between the ten types of ABBs:

- Process decomposition ABB, service composition ABB, and process collaboration controller ABB realize control flow ABB.
- Access control ABB, configuration rule ABB, and state management module ABB realize utility helper ABB.

- Process compliance ABB interacts with control flow ABB, data flow ABB, utility helper ABB, process repository ABB, and process service adapter ABB. Their relationships are all one-to-one mapping relationships.
- There is a one-to-one relationship between control flow ABB and data flow ABB.
- There is a one-to-many relationship between control flow ABB and process repository ABB.
- There is a one-to-many relationship between process service adapter ABB and process repository ABB.
- There is a one-to-one mapping relationship between process service adapter ABB and utility helper ABB.
- There is a one-to-one relationship between control flow ABB and utility helper ABB.

5.8.2 Relationship Diagram across Other Layers

Figure 28 uses a UML component diagram to illustrate a static view of the relationships for the ABBs across layers. Various layers identified in the SOA-RA are represented as packages; ABBs are represented as components. Note that the relationships between ABBs within the Business Process layer are removed from Figure 28 to highlight the relationships between the ABBs across other layers.

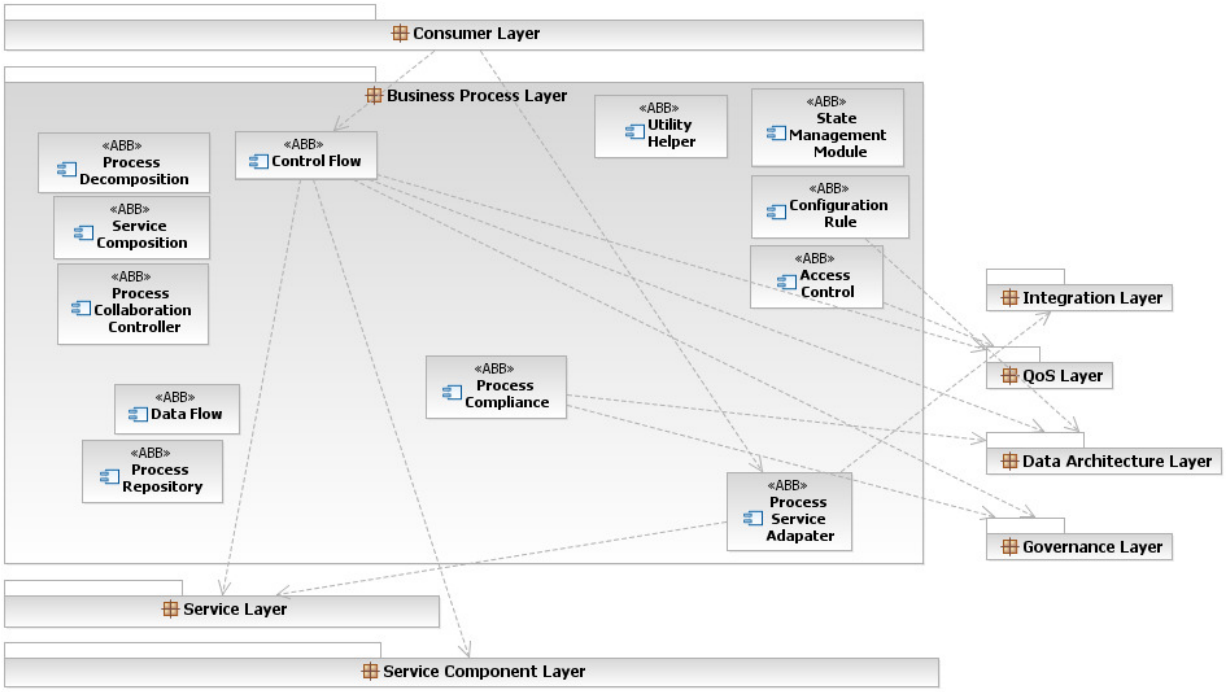

Figure 28. *Component Relationship Diagram – ABBs in the Business Process layer across layers.*

Certain relationships exist between the ABBs in the Business Process layers with other layers:

- A *control flow* ABB (process decomposition ABB, service composition ABB, and process collaboration controller ABB) needs to interact with the Consumer layer, the Service

layer, the Service Component layer, the QoS layer, the Data Architecture layer, and the Governance layer.
- A *process service adapter* ABB needs to interact with the Consumer layer, the Service layer, and the Integration layer.
- A *process compliance* ABB needs to interact with the Data Architecture layer and the Governance layer.
- An *access control* ABB needs to interact with the QoS layer.
- A *configuration rule* ABB needs to interact with the Data Architecture layer.
- A *state management controller* ABB needs to interact with the Governance layer.

5.9 Options and Design Decisions

There are some options for us to design and manage business processes in this layer. The Business Process layer may leverage the SOMA methodology to define goal-oriented business protocols to support business process choreography. It uses service-to-service collaboration patterns and business requirements to aggregate services as a coordinated process. When a customer changes business requirements or performs business transformation using SOA, the Business Process layer leverages modeling tools (e.g. WBI Molder) to model the business processes (AS IS and TO BE) and reflects the business changes in the message exchange protocols and supporting infrastructure.

From the design decision perspective, when messages become less transactional in nature (e.g. interactions between parties change from well defined messages, to more referential requests or notifications), the interactions between business process layer and integration layer will be very often. The business process will become message or content driven workflow instead of a predefined sequence of tasks. From the state management perspective, the state can be managed at global process level or at individual services or sub-processes level.

In order to enable and facilitate dynamic and automatic business process composition using existing Web services, the following six goals are recommended to be fulfilled:

- ***G1: A uniform representation of business requirements***

There is a need for a uniform representation to precisely capture comprehensive business requirements, preferences, Web services features, event-action mapping, as well as the relationships among Web services.

- ***G2: An automated mechanism to discover Web services***

There is a need for an automated mechanism to generate search scripts, which can dynamically discover appropriate Web services for a specific task from UDDI registries or other Web services registries populated with Web services records.

- ***G3: A seamless integration mechanism for Web services***

There is a need for a seamless integration mechanism, which can perform template-based and event-driven business process flow composition of existing Web services.

- ***G4: An effective service selection mechanism for business process optimization***

There is a need for an effective service selection mechanism, which can automatically construct an optimal business process using available Web services based on business goals and requirements.

- ***G5: An efficient tool to support dynamic adaptation of Web services flow***

There is a need for an efficient tool to support dynamic adaptation of Web services flow in regards to various modeling languages (e.g., BPEL, BPML, UML) for rapid integration.

- ***G6: A stepwise methodology to guide business process composition***

There is a need for a stepwise methodology to guide through Web services-based business process composition and adaptation. We recommend SOMA for performing this.

5.10 Activities

The techniques and technologies in the field of business process modeling and integration can be used to model, assemble, deploy, and manage service-oriented business process in this process layer.

For example, we can use IBM WBI Modeler to model the business processes. Then the activities defined in the business process maps to a set of business primitives (name and message exchange sequence) represented in a BPEL format. In addition, this layer needs to create customized data entities and messages for different business scenarios based on a plug-and-play framework in this layer.

The business processes in this layer can be deployed on a process-engine-based server or embedded in screen-flow-based Web applications.

In addition to the state management in the business process, runtime monitoring and tracking the status of the projects, tasks, documents or files in SOA environments are very important. It can keep a long-running process on the right track by handling business exceptions through people involvement or a dashboard-based interaction system.

6 Layer 5: Consumer

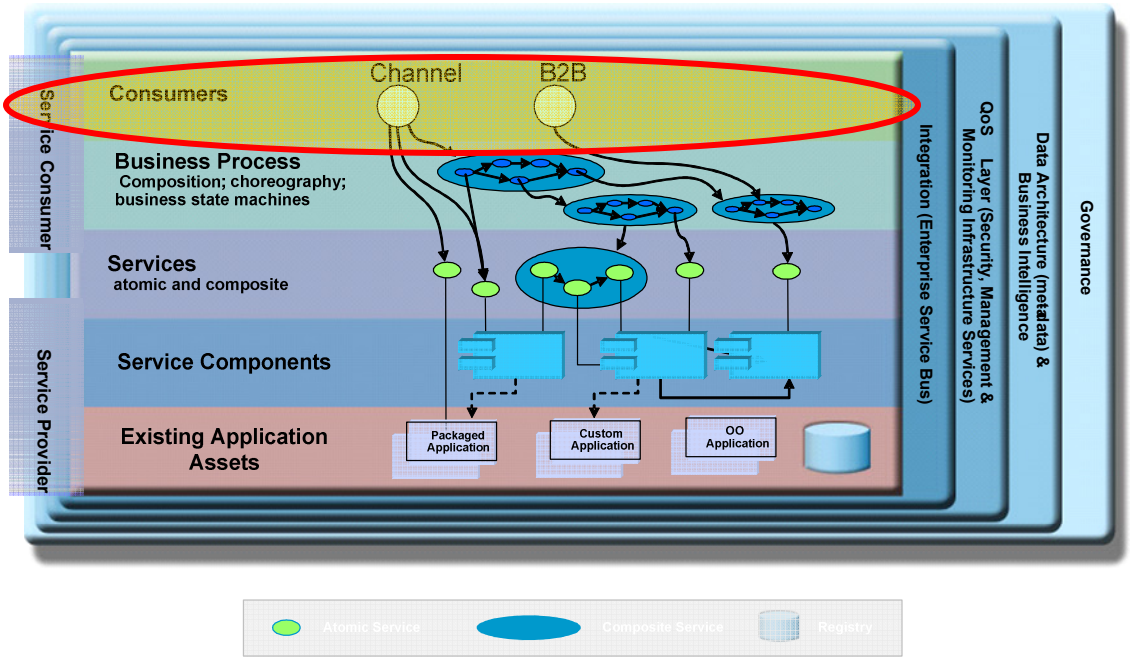

Figure 29. Consumer Layer in S3

6.1 Layer Description

The Consumer layer is the presentation layer that leverages the Business Process layer and the Service layer to quickly construct the user interfaces of business services to fulfill customer requirements. In other words, this layer is responsible for building a front-end interface for an SOA solution. The Consumer layer typically should provide caching facility to enhance presentation performance. In addition, this layer should simultaneously support various types of users, such as B2B, desktop, wireless, and PDA.

6.2 Value Proposition

The Consumer layer provides the capability to quickly create the front-end of the business processes and composite applications to respond to changes in the marketplace. It enables channel-independent access to those business processes supported by various applications and platforms.

By adopting proven front-end access patterns, portals and open standards (e.g., WSRP), the Consumer layer can decrease development and deployment cycle times through the use of many pre-built, proven and reusable front-end building blocks. It also reduces complexity and maintenance costs with those common building blocks. In this way, the Consumer layer promotes a single unified view of knowledge presentation, a single unified entry point to the

supported business processes/applications that integrates with other foundational services such as security (e.g., single sign-on) and trust, and significantly improves the usability of the business processes/applications.

More specifically, the Consumer layer allows for the plug-and-play of content sources (*e.g.*, portlets) with portals and other aggregating Web applications. It thereby standardizes the consumption of Web services in portal front-ends and the way in which content providers write Web services for portals. Scenarios that motivate WSRP/WSIA functionality include: (1) Portal servers providing portlets as presentation-oriented Web services that can be used by aggregation engines; (2) Portal servers consuming presentation-oriented Web services provided by portal or nonportal content providers and integrating them into a portal framework. The corresponding descriptions can be also applied to non-portal environments. WSRP allows content to be hosted in the environment most suitable for its execution while still being easily accessed by content aggregators. The standard enables content producers to maintain control over the code that formats the presentation of their contents. By reducing the cost for aggregators to access their content, WSRP increases the rate at which content sources may be easily integrated into pages for end-users.

6.3 Characteristics

There are three key characteristics of this layer:

- Unified, extensible front-end components
- Standards based, e.g., WSRP
- Integrated presentation and security/trust management

6.4 Key Performance Indicators

Under construction.

6.5 Architectural Building Blocks

As shown in *Figure 30*, we identify eight fundamental Architectural Building Blocks (ABBs) in the Consumer layer: (1) consumer, (2) presentation (view), (3) presentation controller, (4) consumer profile, (5) access control, (6) format transformation, (7) configuration rule, and (8) cache.

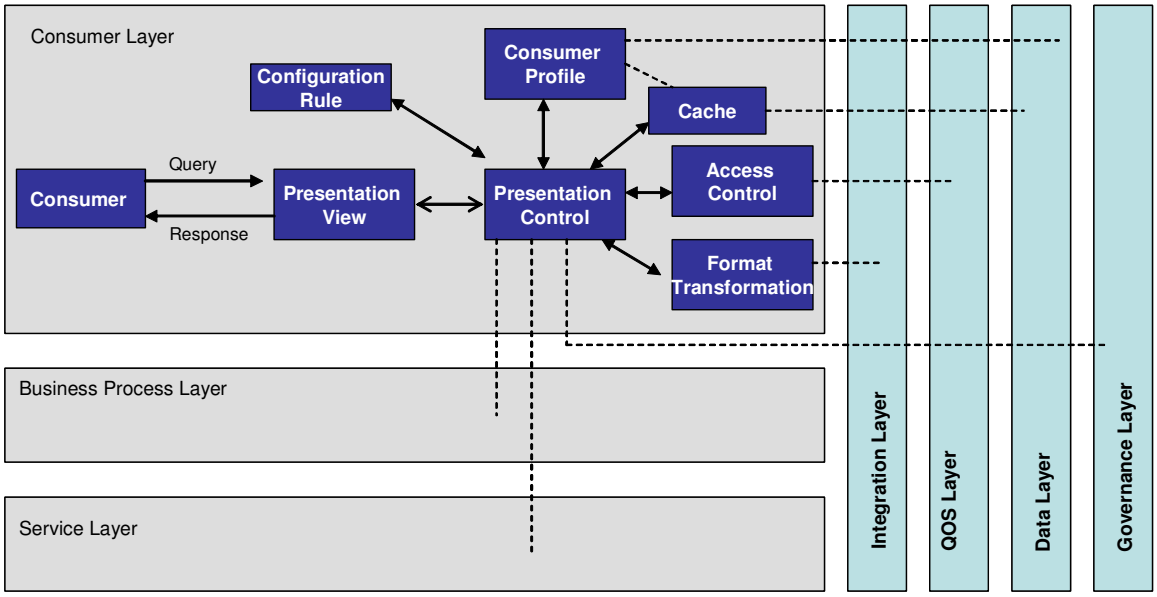

Figure 30. *Consumer module ABBs.*

6.5.1 Component Descriptions – Consumer Layer

This section describes in detail each ABB in terms of its responsibilities.

6.5.2 Consumer

A *consumer* building block represents an external user of the presentation model. It can be either a program or an individual who requests a service.

6.5.3 Presentation (View)

A *presentation (view)* building block is responsible for getting input via Query from a consumer and providing or displaying response to a consumer. In other words, it is responsible for communicating to and from the consumer.

6.5.4 Presentation Controller

A *presentation controller* building block manages the navigation logic and process for consumer interactions. For instance, it may interact with a service in the Service layer or a process in the Process layer. Meanwhile, it interacts with other ABBs, such as: customer profile to control navigation based on the consumer profile, access control to determine what content can be presented, and a format Transformation to translate to Query data formats required by the Integration module and to convert response from the Integration module to appropriate consumer response format. In addition, this ABB typically should conduct load balancing facility to enhance presentation performance.

6.5.5 Consumer Profile

A *consumer profile* building block is responsible for getting customer-specific information (enabled through the Data Architecture layer) to be used by the presentation controller for navigation and content presentation purposes.

6.5.6 Access Control

An *access control* building block provides access to authentication/authorization capabilities (enabled through the QoS layer) to be used by the presentation controller to allow/prevent what content can be presented to the consumer.

6.5.7 Format Transform

A *format transformation* building block is responsible for translation of Query content format required by the Integration layer and to convert content returned from the Integration layer to a consumer response format. It also can be used for transforming content (e.g., a message payload can be translated using XSLT to a required XML format) in the invocation messages to a service or a process. It can support multiple XSLTs and other transformation mechanisms. It should be noted that this transformation addresses only the changing of content formats for presentation, e.g., from XML to HTML or from XML to VXML. It does not handle the transformation of actual content, which is the responsibility of either the Process or the Service layer (e.g., converting to Industry-specific message format).

6.5.8 Configuration Rule

A *configuration rule* building block is responsible for hosting rules that indicate how the ABBs can be configured based on consumer request scenarios. This unit enables presentation configuration on demand. It also allows the use of only appropriate ABBs. It should be noted that this configuration capability will not be effective if the ABBs are coarse grained, as it will reduce flexibility. On the other hand, if ABBs are fine grained, they will be more flexible to be configured based on specified rules. This configuration can be handled in the following two ways. The first is through template-based configuration, where a user can select a specific template based on the corresponding service request scenario. The system will select all the rules associated with this template and configure the ABBs to support the rules. This approach requires that scenario templates be created and stored in a repository to be selected when needed. The second is through dynamic template creation, where a user selects certain characteristics and the system will determine the appropriate rules and configure using relevant ABBs at run time. For instance, user may specify the following two characteristics: - data is static (i.e., does not change during an interaction) and user should not be challenged multiple times. The first characteristic implies that a cache ABB is needed; the second one implies the necessity of using a proper security token mechanism. These characteristics will be used during the run time to configure appropriate ABBs.

6.5.9 Cache

A *cache* building block is responsible for storing consumer interaction-related data temporarily to enhance system performance. These data typically are maintained for the duration of the entire interaction. Example of such a kind of data is a customer profile. This building block will increase performance as it minimizes interactions with the Data layer.

6.6 Dependencies and Interactions (patterns)

6.6.1 Relationship Diagram within the Consumer Layer

Figure 31 uses a UML component diagram to illustrate a static view of the relationships between the ABBs within the Consumer layer. All identified ABBs are represented as components in the diagram, with "ABB" as stereo type. The Consume layer is represented as a package containing all identified ABBs.

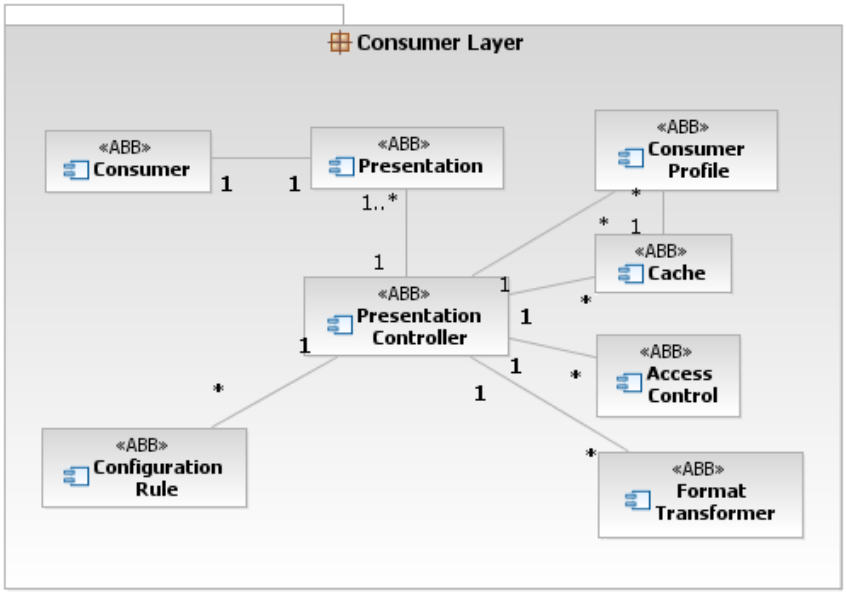

Figure 31. *Component Relationship Diagram – ABBs within the Consumer Layer.*

Certain relationships exist between the eight types of ABBs:

- There is a one-to-one relationship between a consumer and a presentation (view). Each type of consumer should have a dedicated presentation (view).
- Each presentation (view), consumer profile, access control, format transformation, configuration file, and cache needs to interact with and under control of the presentation controller. The relationship between the presentation controller and its related ABBs is a one-to-many relationship, meaning that there is only one presentation controller ABB in a Consumer layer that controls all other ABBs.
- There is a one-to-many or many-to-many relationship between a cache ABB and a consumer profile ABB. This relationship is for the purpose of single-sign-on. After a consumer signs onto the system, his/her consumer profile needs to be stored in the

cache, so that he/she can navigate through different functionalities without multiple sign-ons. In addition, when a consumer re-sign in, the system should remember his/her profile and serve accordingly.

- Identified presentation (view), consumer profile, access control, format transformation, configuration file, and cache ABBs are not recommended to interact with each other directly. In order to realize higher reusability, flexibility, and extensibility of identified ABBs, these ABBs should be loosely coupled and only interact with each other through other ABBs (i.e., presentation controller) or other layers (e.g., the Integration layer or the QoS layer).

6.6.2 7.6.2. Relationship Diagram across Other Layers

Figure 32 uses a UML component diagram to illustrate a static view of the relationships for the ABBs across layers. Various layers identified in the SOA-RA are represented as packages; ABBs are represented as components.

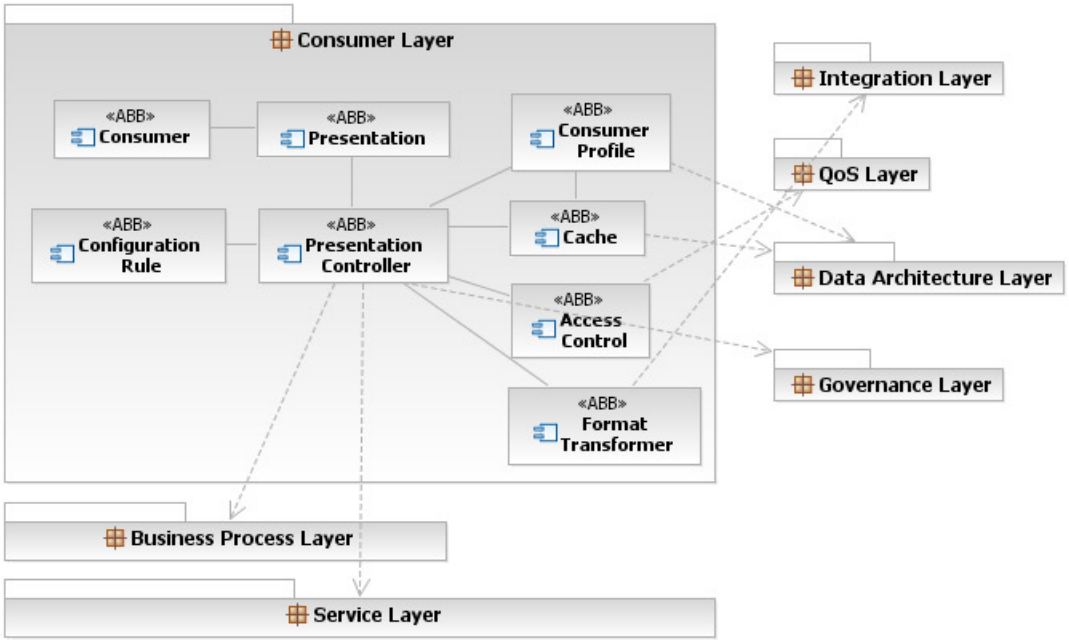

Figure 32. *Component Relationship Diagram – ABBs in the Consumer layer across layers.*

Certain relationships exist between the ABBs in the Consumer layers with other layers:

- A *presentation control* ABB needs to interact with both the Business Process layer, the Service layer, and the Governance layer.
- A *format transformation* ABB needs to interact with the Integration layer.
- An *access control* ABB needs to interact with the Quality of Services (QoS) layer.
- A *cache* ABB needs to interact with the Data Architecture layer.
- A *consumer profile* ABB needs to interact with the Data Architecture layer.

In theory, a *presentation control* ABB can directly interact with the Service Component layer or even the Existing Application Asset layer. However, in SOA-RA, we strongly recommend that they do not directly communicate with each other. Instead, they should indirectly interact with each other through other layers, such as the Integration layer or the Data Architecture layer.

6.7 Options

There are two major option points:

1. First, there is a key decision on how to integrate applications at the front-end, namely whether a portal will be used. If so, WSRP should be adopted.
2. Whether to use single-sign-on through portals is another important question. Some of the issues that need to be considered are: a) whether it is possible to share a security token after the authentication by the business processes/applications and how to make such tokens applicable across several security domains; b) what processes or services need to enforce proper authorization and how (role based, e.g).

Presentation oriented characteristics of services can surface in SOA through constraints imposed by legacy operational systems. Customers have had extensive experience trying to transform 3270 application into a set service. Legacy applications can't always be easily decoupled from service definitions e.g. legacy apps already implement extensive data validation logic. This aspect leads to constraints in service design such as attaching error indicators to every data element or adding runtime metadata to messages such as read-only indicators, lists of options etc. Because logic that drives this remains buried in the legacy, service interfaces can't always explicitly contain corresponding data models and rather are deliberately "dumbed-down" to allow embedded metadata to be included at runtime. (See section related to the Data Architecture Layer for more details).

6.8 Architectural Design Decisions

In this section, we will discuss architectural decisions that an SOA solution architect needs to make for the Consumer layer.

When designing a Consumer layer, an architect needs to make critical architectural decisions based upon the S3 model. This section describes a set of typical architectural decisions regarding the Consumer layer in a standard format, as summarized in Table 2. It should be noted that a specific application may require further architectural decisions; the descriptions here can be used as templates and examples to make further decisions.

Table 2. Typical architectural decisions for the Consumer layer.

Architectural Decisions	**ID**
Identification of ABBs	**AD-6-01**

Connections among ABBs within the Consumer layer	**AD-6-02**
Configuration of the attributes of ABBs	**AD-6-03**
Interaction patterns for ABBs with other layers	**AD-6-04**
Stateful vs. stateless ABBs	**AD-6-05**
Granularity enablement of state management	**AD-6-06**
Federated vs. individual state management	**AD-6-07**
Enablement of access control on ABBs	**AD-6-08**
Business performance management	**AD-6-09**
Exception handling management	**AD-6-10**
Consumer types	**AD-6-11**
Presentation models (MVC model, portlet technique, and Ajax technique)	**AD-6-12**
Message exchange for ABB interactions	**AD-6-13**
Message presentation transformation strategies	**AD-6-14**
Connectivity to the Business Process layer	**AD-6-15**
Lifecycle management of ABBs	**AD-6-16**

6.8.1 Architectural Decision 1: Identification of ABBs

Subject Area		**Topic**	
Design Decision	Identification of ABBs	**Id.**	AD-6-01
Issue or Problem Statement	For each type of ABB, an architect needs to decide: Whether such a type of ABB is needed in the system architecture? (e.g., Do I need a separate *access control* ABB?) How many instances of each type of ABB are needed? (e.g., Do I need to identify one or more *access control* ABB?) In other words, this is a decision on yes/no and how many.		
Assumptions	ABBs are used to construct the Consumer layer.		
Motivation	Major motivational factors for identifying ABBs according to the SOA-RA model are for enhancing flexibility, extensibility, maintainability, scalability, variability, and reusability of the Consumer layer.		

Alternatives	**1. ABB-based architecture** Identify instances of each of the eight types of ABBs. **2. Integrated architecture** Identify more integrated components that combine multiple ABBs in one unit.

Decision	If the Consumer layer is expected to be highly simple, straightforward with tight budget and time frame requirements, more integrated solution (Alternative #2) that combines multiple ABBs in one component is an alternative. Otherwise, ABB-based architecture (Alternative #1) should be adopted. For each type of ABB, the architect needs to make decisions based upon the following criteria: • Consumer: If the system is required to provide interfaces to users (either programs or individuals), the *consumer* ABB is needed. Each expected type of user requires a separate *consumer* ABB. For example, PDA or wireless phone-based users, desktop-based uses, and programs each require a dedicated *consumer* ABB. • Presentation (view): If the system is expected to receive input via Query from a consumer and provide or display response to a consumer, the *presentation (view)* ABB is needed. If multiple consumer ABB instances are identified, multiple *presentation (view)* ABB instances are needed. • Presentation controller: If the system needs to manage the navigation logic for consumer interactions, a *presentation controller* ABB is needed. In general, it is recommended that this ABB is always identified. • Consumer profile: If customers are associated with specific information to be used by the presentation controller for navigation and content presentation purposes, a *consumer profile* ABB is needed. • Access control: If Authentication/Authorization capabilities need to be provided for the presentation controller to allow or prevent what content can be presented to the consumers, the *access control* ABB is needed. • Format transformation: If there is need for translation of Query content format required by the Integration layer and to convert content returned from the Integration layer to a consumer response format, the *format transformation* ABB is needed. • Configuration rule: If rules are needed to indicate how the identified ABBs are configured based on consumer request scenarios, the *configuration rule* ABB is needed. In general, if the ABBs are coarse grained, this ABB is not really needed as it will reduce flexibility. On the other hand, if ABBs are fine grained, a dedicated *configuration rule* will enhance flexibility. • Cache: If there is a need to store consumer interaction-related data temporarily to enhance presentation performance, the *cache* ABB is needed.

Justification	Major advantages of identifying ABBs according to the SOA-RA model include: • Separating concerns between data and presentation allows to separate business logic and user interface. • Associating different concerns into different ABBs enhances flexibility, scalability, maintainability, extensibility, variability, configurability, and testability. Different ABBs can be independently developed and tested before integrating with each other. • Associating different concerns into different ABBs based on Web services standards (e.g., WSRF and BPEL) also allow run-time dynamic selection and integration of various ABBs based upon workflow, system status, and user preferences predefined. • To minimize code development efforts due to requirements changes in the Consumer layer. • Allows for both dynamic and static configuration of ABBs. • Allows for maintaining a uniform interface of various ABBs in the Consumer layer, so that they can easily communicate with each other. Moreover, uniform ABBs can easily interact with other components in the other layers.
Implications	This is a critical decision and the first decision for the architectural design of the Consumer layer of any large-scale SOA applications. Unless there are sound technical or business reasons, the ABB-based model should be adopted.
Derived requirements	
Related Decisions	• Connections among ABBs within the Consumer layer (AD-6-02) • Configuration of the attributes of ABBs (AD-6-03)

6.8.2 Architectural Decision 2: Connections among ABBs

Subject Area		**Topic**	
Design Decision	Connections among ABBs	**Id.**	AD-6-02
Issue or Problem Statement	How will we specify the connections among identified ABBs? For example, do they have some connectivity patterns? Will they link to each other? Will they be linked in sequences? What are their relationships?		
Assumptions	ABBs have already been identified.		
Motivation	We need to properly connect the identified ABBs with each other, so that they can collaborate appropriately.		

Alternatives	More connection patterns may be configured if needed.
Decision	Figure 31 has depicted a recommended connectivity diagram among various ABBs identified: • A consumer and a presentation (view) need to interact with each other. • Each presentation (view), consumer profile, access control, format transformation, configuration file, and cache needs to interact with and under control of the presentation controller. • A cache ABB needs to interact with corresponding consumer profile ABB. • Identified presentation (view), consumer profile, access control, format transformation, configuration file, and cache ABBs are not recommended to interact with each other directly.
Justification	The recommended loosely coupled connection pattern supports various presentation scenarios that are stochastic, dynamic, fluctuant, location-based, and application-specific.
Implications	Unless there are sound technical or business reasons, the connectivity model depicted by Figure 31 should be adopted.
Derived requirements	
Related Decisions	• Identification of ABBs (AD-6-01) • Configuration of the attributes of ABBs (AD-6-03)

6.8.3 Architectural Decision 3: Configuration of the attributes of ABBs

Subject Area		**Topic**	
Design Decision	Configuration of the attributes of ABBs.	**Id.**	AD-6-03
Issue or Problem Statement	How can we configure and decide internal structures of each identified ABB?		
Assumptions	ABBs have already been identified.		
Motivation	The architect needs to decide internal structures of each identified ABB.		
Alternatives			
Decision	The architect needs to decide attributes, interfaces, and states of each ABB, according to the WSRF-based ABB definitions described in detail in ARC 101 – SOA-RA – Architecture Overview – Vx.x.doc).		
Justification			
Implications			

Derived requirements	• Identification of ABBs (AD-6-01) • Connections among ABBs within the Consumer layer (AD-6-02)
Related Decisions	

6.8.4 Architectural Decision 4: Interaction patterns for ABBs with other layers

Subject Area		**Topic**	
Design Decision	Interaction patterns for ABBs with other layers	**Id.**	AD-6-04
Issue or Problem Statement	Will the identified ABBs connect with other layers? Which ones will connect to other layers?		
Assumptions	ABBs have already been identified.		
Motivation	ABBs in the Consumer layer need to connect to the ABBs in other layers. We should identify and specify their standard interaction patterns.		
Alternatives	**1. Stochastic interaction patterns** Identified ABBs build random interaction patterns with other layers based upon specific business logic in the development process. **2. Predefined interaction patterns** Figure 31 has depicted a recommended interaction pattern for various ABBs to connect to other layers: • A *presentation control* ABB needs to interact with both the Business Process layer, the Service layer, and the Governance layer. • A *format transformation* ABB needs to interact with the Integration layer. • An *access control* ABB needs to interact with the Quality of Services (QoS) layer. • A *cache* ABB needs to interact with the Data layer. • A *consumer profile* ABB needs to interact with the data layer.		
Decision			
Justification	The recommended interaction pattern supports stochastic, fluctuant presentation scenarios. Meanwhile, the recommended predefined interaction pattern enables a loosely coupled integration relationships between the Consumer layer and the other layers in the SOA-RA.		
Implications	Unless there are sound technical or business reasons, the recommended predefined interaction patterns depicted by Figure 31 should be adopted.		
Derived requirements			

Related Decisions	• Identification of ABBs (AD-6-01) • Connections among ABBs within the Consumer layer (AD-6-02) • Configuration of the attributes of ABBs (AD-6-03)

6.8.5 Architectural Decision 5: Stateful vs. stateless ABBs

Subject Area		**Topic**	
Design Decision	Stateful vs. stateless ABBs	**Id.**	AD-5-05
Issue or Problem Statement	Based upon the assumptions below, an architect needs to decide whether state information needs to be captured and tracked (stateful vs. stateless) in ABBs?		
Assumptions	Due to the world-wide competition, an Internet service needs to serve various types of consumers and provide personalized services. In addition, instead of providing simple, isolated transactions, such a modern service needs to maintain persistent information (e.g., user historical data) to better serve consumers. Furthermore, many such services are comprehensive enough to require a series of related interactions between a consumer and the service; thus, conversational information needs to be kept and tracked during a transaction. ABBs have already been identified, and their relationships, interaction patterns have been specified.		
Motivation	The motivations for this problem is five-fold: • The Consumer layer should maintain a stateful session for a specific consumer in the whole period of a process • A consumer may expect that the system remembers his/her preferences and historical data (e.g., previous activities) to better serve him/her in the future. • A consumer may expect to input his/her preferences only once, while the system can remember them whenever he/she signs in and provides him/her personalized services accordingly. • The Consumer layer needs to interact with other layers, so that stateful information about their interactions and communications is necessary to keep persistent. • The Consumer layer should track the status of running instances of ABBs based upon some predefined requirements (e.g., bandwidth) to check their availabilities. This will enable management of the lifecycles of the created ABB instances.		

Alternatives	**1. Stateless Consumer layer** If the Consumer layer (1) responds to requests from a consumer without regard to any previous information (including requests, preferences, historical information, etc.) from the same consumer, or (2) does not capture and track interactions among ABBs, it is considered as a stateless Consumer layer. This option is easy to implement as no additional mechanism is needed. **2. Stateful Consumer layer** If the Consumer layer (1) remembers consumer information (e.g., personal information and preferences) and keeps track of consumer activities (e.g., historical information), and (2) capture and track interactions among ABBs, it is considered as a stateful Consumer layer. This option requires additional mechanisms (including code and resources).
Decision	• If the system intends to (1) provide simple read-only services (e.g., catalog service) with no user information (e.g., preferences and historical activities) needed to remember, or (2) provide one-time interactions (i.e., the system does not remember a user, and the user always logs in the system as a visitor or new user), a stateless alternative (#1) is probably enough. • If the system provides comprehensive business services requiring multi-step interactions or if the system provides personalized services for multiple entries, a stateful alternative (#2) is needed.
Justification	Business services typically possess a level of sophistication to provide their consumers personalized and individualized services. This in many cases means that the system needs to remember user input preferences and historical activities.
Implications	Stateful Consumer layer does require more coding and additional processing resources. Therefore, it typically has a definite impact on the performance of the system (e.g., cost, configurability and re-configurability of ABBs). Furthermore, it takes longer time to develop the mechanism and the application. Moreover, it may affect the scalability of the system. Therefore, it should be used when it is necessary.
Derived requirements	
Related Decisions	• Granularity enablement of state management (AD-5-06) • Federated vs. individual state management (AD-5-07)

6.8.6 Architectural Decision 6: Granularity enablement of state management

Subject Area		**Topic**	
Design	Granularity enablement of state	**Id.**	AD-5-06

Decision	management		
Issue or Problem Statement	If we decide to construct a stateful Consumer layer, to which granularity we need to capture and track state information? E.g., Do we need to log all interactions and activities in the Consumer layer? What kind of consumer preferences will be captured and stored in the layer?		
Assumptions	It has been decided that a stateful Consumer layer is required.		
Motivation	Different granularities of state management imply different performance impacts and governance levels.		
Alternatives	**1. Predefined state management** **1.1. Coarse grained** High-level state information about interactions of ABBs are captured and tracked. Coarse grained consumer preferences are configurable. **1.2. Fine grained state management** Fine-grained state information about interactions of ABBs and consumer historical activities are captured and tracked. Consumer preferences are captured and stored to provide personalized services to consumers. In addition, consumer preferences are reconfigurable. **2. Reconfigurable state management** Administrators can dynamically adjust the granularity levels of state management, based upon business requirements, including cost and presentation performance. **3. Hybrid state management** Take a hybrid granularity level of state management as well as its configurability, based upon business requirements including cost, development time and efforts, resources, and presentation performance.		
Decision	For a specific application, the architect should take into consideration of all related business requirements and system requirements and decide an application-specific hybrid state management mechanism.		

Justification	Coarse grained state management (Alternative #1.1) is cheaper to implement and takes less resources. It also affects less on presentation performance. Fine grained state management (Alternative #1.2), on the other hand, provides comprehensive state information and allows various levels of control and management over the Consumer layer, as well as various levels of presentation services to customers. However, its tradeoff is that not only it is expensive to implement and maintain, but also it requires significant amount of resources at run time thus it may affect presentation performance. Reconfigurable state management (Alternative #2) offers re-configurability, flexibility, and extensibility to both the Consumer layer and customers. However, it requires significant development efforts. Hybrid state management (Alternative #3) takes into consideration of various system requirements (e.g., flexibility, extensibility, configurability, re-configurability, maintainability, performance) and business requirements (e.g., cost, development time and efforts, resources) and leads to an application-specific solution.
Implications	The architect should gather all related business requirements and system requirements before making a decision on the granularity of state management.
Derived requirements	
Related Decisions	• Stateful vs. stateless ABBs (AD-5-5) • Federated vs. individual state management (AD-5-07)

6.8.7 Architectural Decision 7: Federated vs. individual state management

Subject Area		**Topic**	
Design Decision	Federated vs. individual state management	**AD ID**	AD-5-07
Issue or Problem Statement	Will the state information be associated with individual ABBs or will we use a federated state management mechanism? Who will have access to logged states?		
Assumptions	ABBs have been identified; stateful Consumer layer strategy has been adopted; granularity of state management has been decided.		
Motivation	We need to decide whether each ABB needs to capture and store state information, or a centralized state management unit is needed.		
Alternatives	**1. Federated state management** The Consumer layer establishes a centralized solution, with a dedicated state management control unit and a centralized state repository. All state information is		

	captured by the control unit and stored into the centralized state repository. Any state information needed is retrieved through the control unit from the centralized repository. Typically, this centralized state management control unit and associated state repository can be part of the *presentation control* ABB, either integrated with the ABB or becoming a self-controlled unit. **2. Individual state management** Each ABB needs to implement a state management control unit and include a state repository. In other words, each ABB encapsulates a state management facility. **3. Hybrid solution** Each ABB encapsulates certain level of state management facility, while the Consumer layer maintains a centralized state management control unit associated with a dedicated state repository to manage the interaction activities among the ABB instances within the Control layer, as well as to the other layers.
Decision	An application-specific hybrid solution (Alternative #3) is recommended.
Justification	A centralized state management solution (Alternative #1) is probably easier to implement and management. However, it has to undergo changes when there are any changes of ABBs controlled within the Consumer layer. In addition, its reusability is limited. Moreover, the centralized state management control unit and its state repository may become a performance bottleneck under certain circumstances. An individual state management solution (Alternative #2) integrates state management into each ABB. State management (including capturing and retrieval) thus can be distributed to individual ABBs; presentation performance is affected less. However, the development of each ABB thus requires more efforts. In addition, how to share state information among different types of ABBs needs to be solved. Typically, a common schema of how state information can be captured and stored by individual ABBs needs to be defined. The hybrid solution (Alternative #3) requires the architect to balance between the Alternative #1 and Alternative #2, taking into consideration of various system requirements (e.g., flexibility, extensibility, configurability, re-configurability, maintainability, performance) and business requirements (e.g., cost, development time and efforts, resources).
Implications	The architect should gather all related business requirements and system requirements before making a decision on the state management implementation solution. Furthermore, the granularity level of state management decided earlier (in AD-5-05) also influences the decision. For example, if it has been decided that a very fine grained state management is required, more individually controlled state management may be more appropriate, since otherwise the centralized solution may cause a significant bottleneck during peak interaction time.
Derived	

requirements	
Related Decisions	• Stateful vs. stateless ABBs (AD-5-5) • Granularity enablement of state management (AD-5-06)

6.8.8 Architectural Decision 8: Enablement of access control on ABBs

Subject Area		**Topic**	
Design Decision	Enablement of access control on ABBs	**Id.**	AD-5-08
Issue or Problem Statement	Who, When, How much? • Who can access each ABB? • When can an ABB be accessed? • What kind of information (how much content, partial or full) can be shared by an ABB? • Who can change an ABB?		
Assumptions	ABBs have been identified.		
Motivation	On one side, one may want to have access to another ABB. On the other side, one ABB may want to protect some information from accessing.		
Alternatives	**1. Associate each ABB with an access control ABB instance** Each ABB is associated an access control ABB instance, which decides which ABB can access it, what kind of information can be shared by whom, etc. **2. One access control ABB for all ABBs** Only one access control ABB is established to specify the access privileges and policies for all ABBs. **3. A hybrid approach** One access control ABB is established to specify overall access privileges and policies. If a specific ABB requires finer grained access control, it can be associated with a dedicated access control ABB.		
Decision	If control privileges are comprehensive and fine grained, each ABB can be associated with a dedicated access control ABB. Otherwise, one shared access control ABB can be used. An ABB can be defined as either public, private, or partially public with configurable expose levels, or even configurable expose levels to different users (i.e., privileges).		
Justification			
Implications	Any-level of granularity can be realized on access control, based upon business requirements.		
Derived requirements			

Related Decisions	• Identification of ABBs (AD-5-01)

6.8.9 Architectural Decision 9: Business performance management

Subject Area		**Topic**	
Design Decision	Business performance management	**Id.**	AD-5-09
Issue or Problem Statement	To which level we need to conduct business performance monitoring and management? What kinds of information need to be monitored? Do we need to track all activities, including query data from consumers and responses from back-end systems?		
Assumptions	ABBs have been identified and configured.		
Motivation	We should manage business performance of the Consumer layer.		
Alternatives	Various levels of business performance management can be adopted, such as: • Status of ABB instances, • Alignment of ABB patterns with business goals and requirements, • Alignment with business Key Performance Indicators (KPIs), such as cost and time, • Alignment with business processes, services, and people, • Impact analysis and dependency analysis		
Decision	The architect needs to make decisions on what levels of business performance management are needed, i.e., which types of above business performance management are needed.		
Justification	Different presentation scenarios may require different levels of business performance management. All above alternatives should be considered and decided.		
Implications	The architect needs to examine the above alternatives and make decisions.		
Derived requirements			
Related Decisions			

6.8.10 Architectural Decision 10: Exception handling management

Subject Area		**Topic**	
Design Decision	Exception handling management	**Id.**	AD-5-10

Issue or Problem Statement	• Which ABB instance creates the exception? • What kind of exception is it? • When does the exception occur? • How to address and handle the exception? • What are the implications and impacts of the exception?
Assumptions	
Motivation	They system should have a powerful exception handling mechanism. When exceptions occur in the Consumer layer, they should be caught and blocked, by the exception handling mechanism, from propagating to other layers or consumers. From a user's perspective, if an exception happens, he/she should only be notified that the system encounters some issues without seeing system-internal information (e.g., which function meets what exception). In a word, the fundamental rule is to reduce and minimize cascading effect of exceptions.
Alternatives	
Decision	A comprehensive exception handling mechanism and system needs to be decided and constructed early in the stage.
Justification	
Implications	
Derived requirements	
Related Decisions	

6.8.11 Architectural Decision 11: *Consumer types*

Subject Area		**Topic**	
Design Decision	Consumer types	**Id.**	AD-5-11
Issue or Problem Statement	Consumer type selection decisions, in general, for B2B type interactions, we can assume a consumer is a program, for the Web-based services portal, we can assume that a consumer is an individual user.		
Assumptions			
Motivation	We need to first decide the type of consumers of the system, so that we can decide what kind of presentation information we need to generate and prepare.		
Alternatives			

Decision	In general, regarding B2B type interactions, we can assume a consumer is a program; regarding Web-based Internet services portal, we can assume that a consumer is an individual user.
Justification	In a B2B type interaction, a consumer is normally a business process that intends to request the system for services. Responses are consumed by the business process to conduct its predefined business goals. Therefore, if the system intends to be used by a B2B process, we can assume that a consumer is a program. If the system intends to be a Web-based services portal, the users are usually individual users who access the portal through a GUI.
Implications	If a consumer is a program, we normally need to produce machine-understandable (e.g., XML-based) presentation views. Otherwise, if a consumer is an individual user, we need to consider in building GUI for the system.
Derived requirements	
Related Decisions	• Identification of ABBs (AD-5-01)

6.8.12 Architectural Decision 12: Presentation models (MVC model, portlet technique, and Ajax technique)

Subject Area		**Topic**	
Design Decision	Presentation models (MVC model, portlet technique, and Ajax technique)	**Id.**	AD-5-12
Issue or Problem Statement	The architect needs to decide whether to adopt some presentation models and techniques, such as Model-View-Controller (MVC) model, portlet technique, and Ajax technique.		
Assumptions	Presentation (view) ABBs have been identified.		
Motivation	There exist several popular models and techniques dedicated for presentations. Adopting appropriate techniques can enhance flexibility and extensibility of the Consumer layer without developing everything scratch.		

Alternatives	**1. MVC model** MVC model (a.k.a. pattern) stands for Model-View-Controller, which intends to guide the establishment of a reusable, modularized presentation framework. The MVC model decouples domain (i.e., model) objects from its presentations (i.e., views) to enable reusability of domain objects. It also decouple business logic (data) from its presentation (views). **2. Portlet technique** In short, a portlet represents an independent window on a portal. Each portlet directly connects to the back-end systems to represent an interface to serve specific business requirements. For example, a portal can have many portlets, such as one for personalized email browser, one for company-wide news, one for task processing, etc. Portlets can communicate with each other through message passing. In short, the portlet technique provides a modularized layout framework to define a presentation interface. **3. Ajax technique** Ajax technique is a non-intrusive content refreshing architecture for Web-based presentations. In general, the content of a Web page can be refreshed often. The Ajax technique intends to offer users smooth experiences when the content updates. Embedding synchronous Java scripts and XML tags, its main idea is to send to the server an HTTP request that includes XML contents, so the server side parses the HTTP request content and retrieves information from the back-end systems. Once XML content is returned, it will be extracted and reflected in the Web page without refreshing (reloading) the whole current page.
Decision	Always try to adopt and apply the MVC pattern to enhance reusability and flexibility of the Consumer layer. Try to adopt the portlet technique to organize various business logics in a modularized manner in one presentation interface. Try to adopt the Ajax technique to enable a light-weight non-intrusive content refreshing to offer smooth browse experiences.
Justification	These are known and proven models and techniques that facilitate the design and development of presentation services.
Implications	Try to use the known techniques.
Derived requirements	
Related Decisions	• Identification of ABBs (AD-5-01)

6.8.13 Architectural Decision 13: Message exchange for ABB interactions

Subject Area		**Topic**	Connectivity
Design Decision	Message exchange for ABB interactions	**AD ID**	AD-5-13
Issue or Problem Statement	Different types of ABBs within the Consumer layer or across different layers may adopt different internal schemas for data descriptions. Thus, in order to enable communications and interactions between different types of ABBs, proper message exchange mechanisms are needed.		
Assumptions	The interaction patterns between ABBs within the Consumer layer and across with other layers have been identified and decided.		
Motivation	• Flexible and extensible mechanisms are needed for message exchange between ABBs. • Further enhancements on performance are needed regarding message communications.		
Alternatives	**1. Fixed schema** A fixed schema is adopted and followed by all ABBs across all layers for data presentations. **2. End-to-end message exchange** A dedicated message exchange method is implemented between two types of ABBs (either within the Consumer layer or across various layers in SOA-RA) if necessary. **3. Message exchange adapter** A generic message exchange adapter mechanism intends to enable and facilitate message communications between two ABBs (either within the Consumer layer or across various layers in SOA-RA), by translating between two message schemas. An instance of the adapter will be created as a plug-in for a pair of ABBs requiring interactions. It is the adapter's responsibility to realize smooth message exchange; the two ABBs are not aware of it.		
Decision	A message exchange adapter (Alternative #3) mechanism is recommended.		
Justification	The fixed schema strategy (Alternative #1) is easy to realize and requires only one-time implementation at the beginning. However, it is not easy, sometimes may be impractical, to define a fixed schema suitable to all ABBs in all nine layers in SOA-RA. Furthermore, this alternative is inflexible. When one type of ABB requires changes in its data presentation schema, the fixed schema needs to be changed accordingly. Therefore, its maintainability is also limited. The end-to-end message exchange strategy (Alternative #2) builds transformation mechanism when necessary. It undergoes limitation of low reusability. The message exchange adapter strategy (Alternative #3) provides a highly reusable and flexible technique to enable and facilitate message communications and interactions between any pair of ABBs. In addition, message transformation		

	implementations are hidden inside of adapters; thus, ABBs do not need to consider or carry any code regarding message transformation. As a result, ABBs become more reusable.
Implications	For a pair of ABBs that need message communication, a message exchange adapter needs to be extended from the generic message exchange adapter..
Derived requirements	
Related Decisions	• Connectivity to the Business Process layer (AD-5-15)

6.8.14 Architectural Decision 14: Presentation transformation

Subject Area		**Topic**	Connectivity
Design Decision	Presentation transformation	**AD ID**	AD-5-14
Issue or Problem Statement	• Proper presentation translation strategies are needed for pre-processing and post-processing. Input data from consumers may need to be transformed into internal queries; response data from other layers may need to be transformed into output data for presentation. • Input queries may need to be aggregated before sending to another layer for further processing; results may need to be aggregated before returning to consumers.		
Assumptions	The interaction patterns between ABBs within the Consumer layer and across with other layers have been identified and decided.		
Motivation	• A flexible and extensible strategy is needed for input pre-processing and output post-processing. • Further performance enhancements are needed regarding message communications.		
Alternatives	**1. Propagate without transformation** Consumer input is propagated directly to other layers without pre-processing; responses from other layers are propagated to the consumers without post-processing. In other words, the Consumer layer merely forward consumer input and system responses without processing. **2. Presentation pre- and post-processing** Consumer input data are translated into system-internal queries before passing to other layers. Responses from other layers are translated into consumer-understandable presentation formats before returning to consumers. In general, two alternative strategies are available to realize presentation pre-processing and post-processing: end-to-end translation and translation adapter.		

	2.1 End-to-end translation A dedicated presentation translation method is implemented for a specific type of consumers. **2.2 Translation adapter** A generic presentation translation adapter mechanism intends to enable and facilitate pre-processing and post-processing for any consumers. An instance of the adapter will be created as a plug-in for a specific type of consumer. **3. Message aggregation** Aggregate queries (if there are multiple) before sending to other layers for further processing, and aggregate results ((if there are multiple) before returning to consumers.
Decision	A presentation translation adapter (Alternative #2.2) mechanism is recommended. An adapter should also include ability of message aggregation (Alternative #3).
Justification	Propagation without processing (Alternative #1) is easy to realize, since the Consumer layer acts only as an intermediary for information passing. However, this alternative requires other layers and consumers conduct more work. Since consumer input is not translated, every other layer receiving the propagated consumer input must interpret and translate the information by itself; since responses from other layers are not translated, consumers must implement a translation mechanism for each layer. This strategy in general is neither flexible nor extensible. The end-to-end presentation translation strategy (Alternative #2.1) builds a translation mechanism for each specific type of consumer. It limitation is that the reusability of the consumer-specific translation mechanisms are low. The presentation translation adapter strategy (Alternative #2.2) provides a highly reusable and flexible technique to enable and facilitate message communications and interactions for any types of consumers. In addition, presentation translation implementations are hidden inside of adapters; thus, both consumers and other layers in SOA-RA do not need to consider or carry any code regarding presentation translation. Automatically aggregating messages enhances message communication performance, by reducing the number of messages sent. Integrating automatic message aggregation facility inside of a presentation translation adapter further enhances presentation performance, since message translation and message aggregation are two things that always need to be handled.
Implications	For a type of consumer, a presentation translation adapter needs to be derived from the generic translation adapter. The automatic message aggregation facility can be implemented as part of the generic presentation translation adapter. Therefore any extended adapters automatically inherit the facility.
Derived	

requirements	
Related Decisions	

6.8.15 Architectural Decision 15: Connectivity to the Business Process layer

Subject Area		**Topic**	Connectivity
Design Decision	Connectivity to the Business Process layer	**AD ID**	AD-5-15
Issue of Problem Statement	How to enable the connectivity between the Consumer layer and the Business Process layer?		
Assumptions	Interactions with the Business Process layer should be a mix of synchronous and asynchronous communications.		
Motivation	We need to establish a connectivity pattern between the Consumer layer and the Business Process layer to enable consistent and manageable connectivity.		
Alternatives	**1. Direct connectivity** ABBs in the Consumer layer directly communicates with the ABBs in the Business Process layer through message passing. **2. Through the Integration layer** The Consumer layer does not directly communicate with the Business Process layer. Instead, all communications between ABBs from the two layers are centralized at a common intermediary - the Integration layer. The Integration layer receives all messages, and routes them to the proper ABBs abiding by some predefined rules and policies.		
Decision	We recommend that all communications between the Consumer layer and the Business Process layer go through the Integration layer (Alternative #2). As a matter of fact, we recommend any cross-layer message communications go through the Integration layer.		
Justification	Direct connectivity (Alternative #1) limits the reusability of both the Consumer layer and the Business Process layer. When a Consumer layer communicates with another Business Process layer, their connectivity has to be re-developed and rebuilt. Communication through the Integration layer provides a centralized strategy for management of cross-layer communications. Any communication methods can be implemented in the Integration layer once and reused by different ABB communications.		
Implications	All ABBs in the Consumer layer need to be enabled for working with the Integration layer.		
Derived requirements			

Related Decisions	• Message exchange for ABB interactions (AD-1-13)

6.8.16 Architectural Decision 16: Lifecycle management of ABBs

Subject Area		**Topic**	
Design Decision	Lifecycle management of ABBs	**AD ID**	AD-5-16
Issue of Problem Statement	How to manage the lifecycle of an ABB instance?		
Assumptions	ABBs have been identified and configured.		
Motivation	We need to manage the lifecycle of an ABB instance, its status, health, etc.		
Alternatives	**1. No lifecycle management of ABBs** The simplest way is to have no lifecycle management of ABBs. **2. Lifecycle management of ABBs** An ABB may have its lifecycle, from the time it is created until the time it is destroyed.		
Decision	We recommend to establish lifecycles for ABBs and manage their status during their lifecycles.		
Justification	Managing lifecycles of ABBs can help to monitor their status, health, and availability. So that corresponding methods can be adopted accordingly. For example, if an ABB is not responding, a new ABB may need to be created and replace the previous one.		
Implications	We need to define a lifecycle for an ABB.		
Derived requirements			
Related Decisions	• Identification of ABBs (AD-5-01)		

6.9 Activities

Under construction <includes layer specific activities related to model, assemble, deploy and manage (life-cycle activities related to this layer)>.

7 Layer 6: Integration

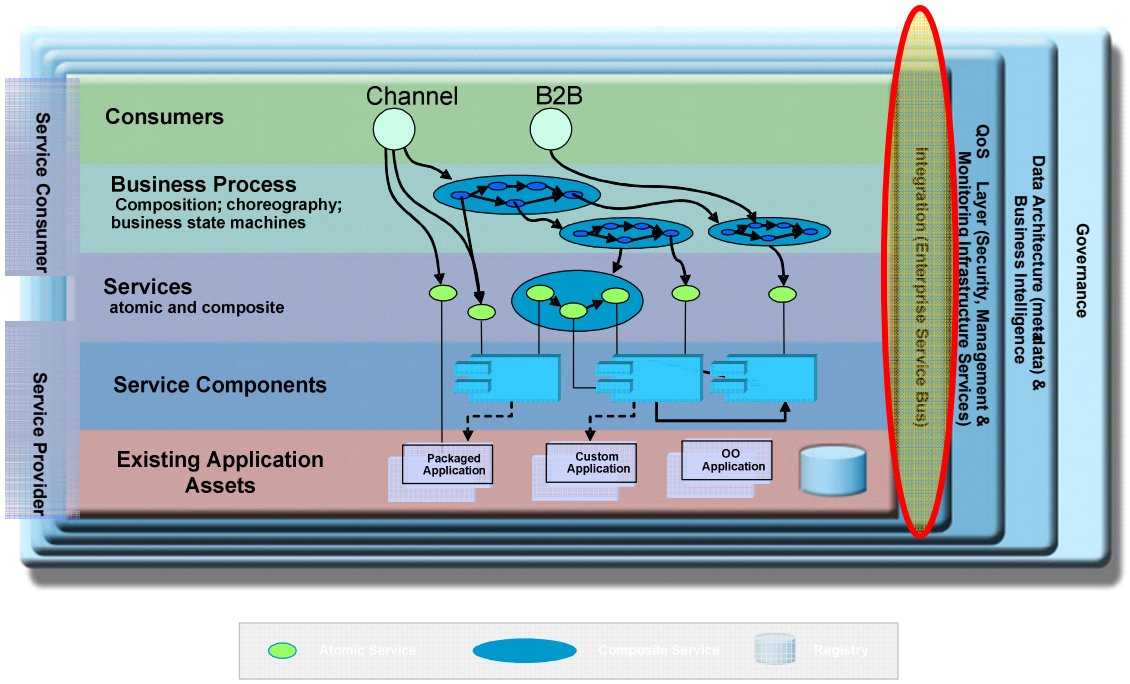

Figure 33. Integration Layer in S3

7.1 Layer Description

The Integration layer is a key enabler for an SOA as it provides the capability to mediate, route, and transform service requests between service requestors and service providers. Enterprise Service Bus (ESB) is one example that fulfills the responsibility of this layer.

This layer enables the integration of services through the introduction of a reliable set of capabilities. These can start with modest point-to-point capabilities for tightly coupled endpoint integration and cover the spectrum to a set of much more intelligent routing, protocol mediation, and other transformation mechanisms often described as an Enterprise Service Bus [5]. WSDL specifies a binding, which implies location where a service is provided. An ESB, on the other hand, provides a location-independent mechanism for integration.

The integration that occurs here is primarily the integration of layers 2 thru 4. This is the place where the Platform 9 ¾ issues are dealt with and they exist for all scales of SOA. For example, this is where binding (late or otherwise) of services occurs for process execution (the **Service Discovery & Execution** box on this chart should be in the Integration layer). This is the SOA infrastructure (I'm characterizing this as a separate concept from the service implementation infrastructure). I like that this chart puts whitespace between the five horizontal layers and the integration layer because I'm starting to think of the horizontal

layers as the SOA (the main value proposition) and the vertical layers as the SOA Infrastructure.

This allows a service to be exposed consistently across multiple customer facing channels such as Web, IVR, Seibel client (used by Customer Service Rep), etc. The transformation of response to HTML (for Web), Voice XML (for IVR), XML string (for Siebel client) can be done via XSLT functionality supported through ESB transformation capability in the Integration layer.

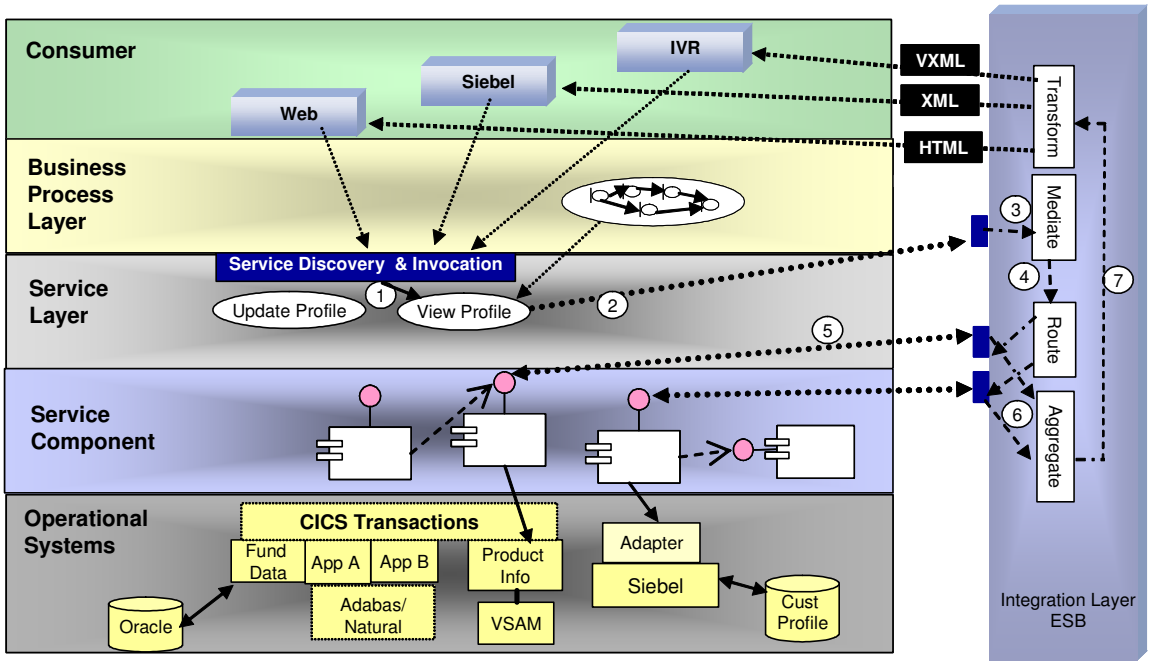

Figure 34. Interactions with the Integration Layer

7.2 Value Proposition

Integration layer

- provides a *level of indirection* between the consumer of functionality and its provider. A service consumer interacts with the service provider via the Integration Layer Hence, each service specification is only exposed via the ESB, never directly:
- consumers and providers are *decoupled*, this decoupling allows *integration* of disparate systems into new solutions.

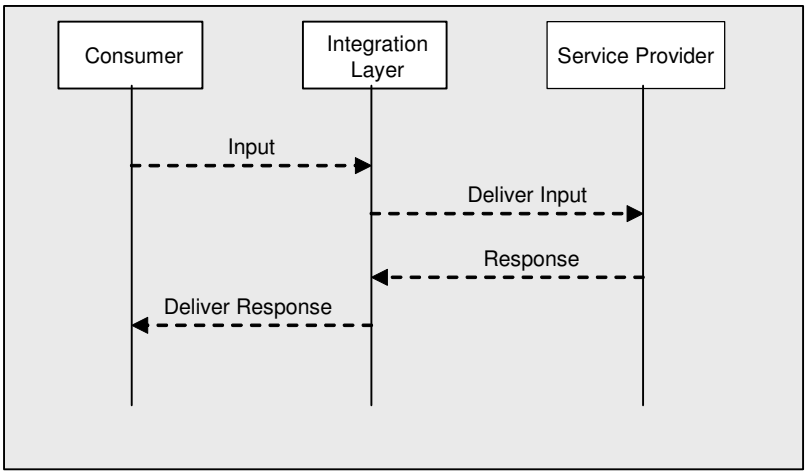

Figure 35. Integration Layer As an agent between Service Consumer and Service Provider

7.3 Characteristics

(i) Communication/Messaging

(ii)

It supports means of communication that allow consumers and providers to interact with it. The interaction happens in the form of message exchanges. It provides for

- Support for multiple transport protocols
- Support for all common messaging patterns
- Consistent naming and addressing
- Support for version management

A consumer must be able to send a message to the Integration Layer over one or more transport protocols and must also be able to receive messages from it. Similarly, a service provider must be able to receive messages from the integration layer and send response messages back to it.

(ii) Service Interactions

The pieces of software functionality that interact via the Interaction layer are services that have a defined interface, separate from its implementation. They can be invoked, across a network, independent of platform or programming language. The integration layer can interact with services by honouring any given service interface and being able to interpret its specific bindings.

(iii) Integration

The Integration layer also serves as an access point for functionality that has not been enabled to act as a service (i.e. it doesn't support a standard service. To the consumer,

functionality is offered as a service by adapting service messages to whatever native protocol and format is supported by the legacy application.

This level of integration includes the ability to transform data from one format to another. Messages coming from service consumers might have to be transformed into the specific data format that a legacy system may require. The following picture shows different styles of integrating legacy applications:

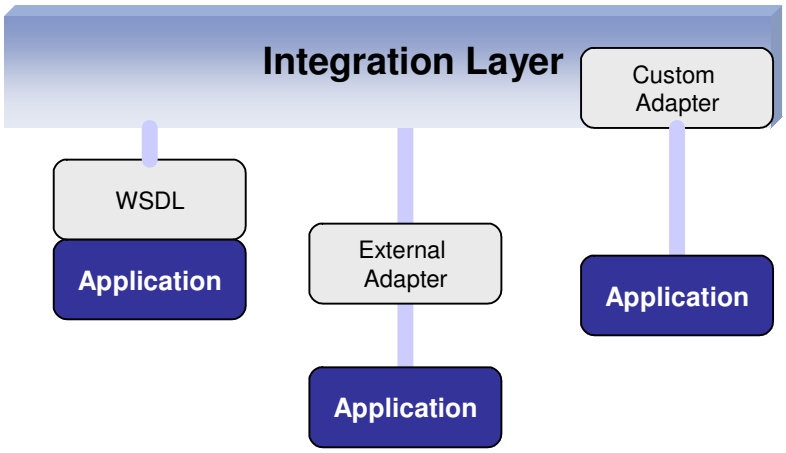

Figure 36. Legacy Application through Integration Layer

(iv) Message Processing/Mediation

The Integration layer supports the following mediation capabilities:

- Protocol conversion which means the extracting of content of a message from one protocol (e.g. SOAP over HTTP) and inserting the message into another transport (e.g. SOAP over JMS).
- Message transformation from application specific format to industry specific format. This includes the transformation of the message payload from one data model to another.
- Perform routing based on content of service messages.
- Appropriately correlate request messages and response messages

7.4 Key Performance Indicators

Under construction.

7.5 Architectural Building Blocks

As shown in Figure 37, we identify thirteen fundamental Architectural Building Blocks (ABBs) in the Integration layer: (1) service mediator, (2) message router, (3) data aggregator, (4) configuration rule, (5) state manager, (6) event manager, (7) message transformer, (8) logger, (9) auditor, (10) access controller, (11) data transformer, (12) protocol transformer,

and (13) exception handler. As shown in Figure 37, the service mediator ABB and the message router ABB form the centralized management unit of the Integration layer, while other ABBs are considered common utility ABBs that are used to support the service mediator ABB and the message router ABB.

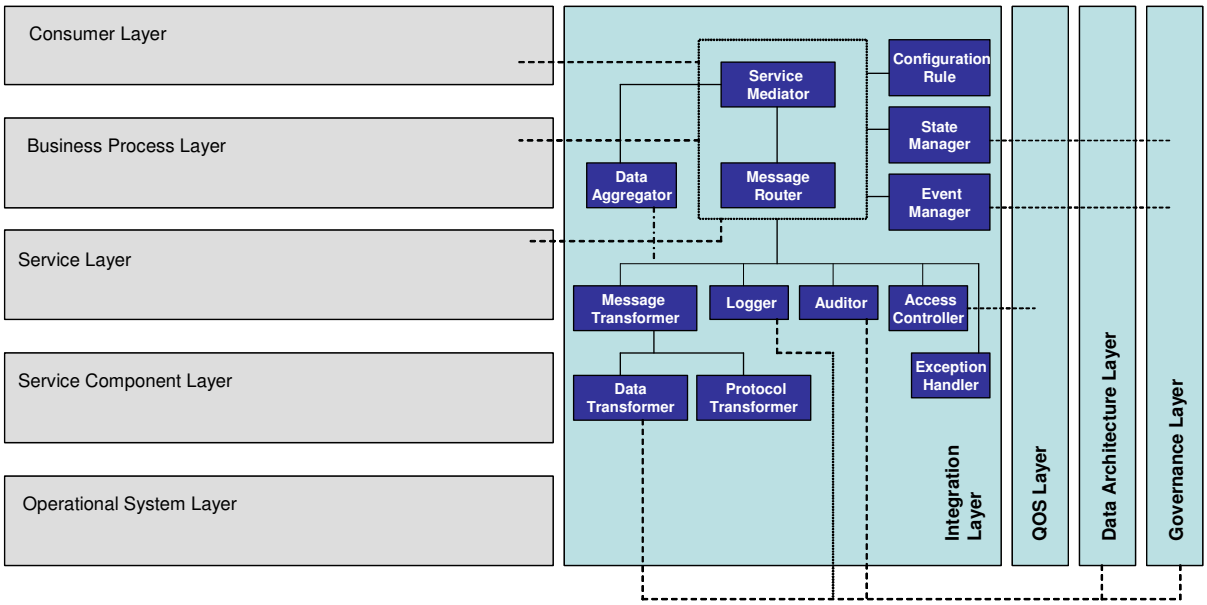

Figure 37. *Integration layer ABBs.*

7.5.1 Component Descriptions – Integration Layer

This section describes in detail each ABB in terms of its responsibilities.

7.5.2 Service Mediator

A *service mediator* building block is the center of the Integration layer. It has two major responsibilities: service request handling and service response handling. In other words, a service mediator ABB acts as a Web Services Invocation Framework (WSIF). Receiving service invocation requests, a service mediator ABB acts on behave of corresponding service requestors to invoke services provided by various service providers. Before a service request is routed to invoke a service, the service mediator ABB may perform message transformation. Once the result is returned from the service provider, the service mediator ABB may also perform message transformation before sending the response back. A service mediator ABB will utilize all other ABBs (including the message router ABB and the utility ABBs) identified in the Integration layer. For example, it may use the message router ABB to receive and disseminate messages. It may also use the access control ABB to ensure authentication and authorization. Service invocation history may be logged using the logger ABB; activity tracking information may be captured using the audit ABB. If any exceptions happen during a service invocation or message routing, the service mediator may use the exception handler ABB to deal with the exceptions. Once the results are received from

various service providers, the service mediator may trigger the service aggregator ABB to aggregate the information. If transformation is needed, it may trigger the message transformer ABB and data transformer ABB.

7.5.3 Message Router

A *message router* building block is responsible for routing messages. In certain circumstances it may be used to route messages without the involvement of service mediator ABB to realize straight message pass-through. A message router ABB may use any of the common service ABBs within the Service layer, including message transformer ABB, auditor ABB, logger ABB, exception handler ABB, and access controller ABB.

7.5.4 Data Aggregator

A *data aggregator* building block is responsible for aggregating information (including messages and data) from different services and service providers under control of the service mediator ABB. Information arriving at data aggregator ABB has already been performed transformation from message transformer ABB and data transformer ABB.

7.5.5 Configuration Rule

A *configuration rule* building block stores rules for configuring ABBs within the Integration layer in accordance with specific scenarios. This unit enables dynamic configuration of ABBs on demand. It also allows the use of only appropriate ABBs. It should be noted that this configuration capability will not be effective if the ABBs are coarse grained, as it will reduce flexibility. On the other hand, if ABBs are fine grained, they will be more flexible to be configured based on specified rules. This configuration can be handled in the following two ways. The first is through template-based configuration, where a user can select a specific template based on the corresponding service request scenario. The system will select all the rules associated with this template and configure the ABBs to support the rules. This approach requires that scenario templates be created and stored in a repository to be selected when needed. The second is through dynamic template creation, where a user selects certain characteristics and the system will determine the appropriate rules and configure using relevant ABBs at run time. For instance, user may require that the system adopt industry messaging standard and satisfy some service level agreements (SLAs). Based on these requirements the system during run-time will select the appropriate data transformation, protocol conversion, and service providers that meet the SLAs.

7.5.6 State Manager

A *state manager* building block is responsible for maintaining state information of the service instances that have been invoked by the service mediator ABB. This ABB is necessary since it is common that a service mediator ABB invokes multiple services simultaneously, and a service may have multiple running instances at the same time. For example, a UPS shipping service provider may support hundreds of shipping services concurrently. At any time, it is necessary to monitor and track the status of a specific shipping service.

7.5.7 Event Manager

An *event manager* building block is responsible for managing the events happened in the Integration layer. An event can be generated from an incoming request, from state changes triggered by the state manager ABB, from the completion of an activity in the data Transformer ABB, from the completion of an exception handling by the exception handler ABB, or from other ABBS in the Integration layer.

7.5.8 Message Transformer

A *message transformer* building block is responsible for message information transformation. It is a super ABB that has two sub ABBs: data transformer ABB and protocol transformer ABB, which handle data and protocol transformation respectively.

7.5.9 Logger

A *logger* building block is responsible for capturing and recording message routing and service invocation history.

7.5.10 Auditor

An *auditor* building block is responsible for tracking and monitoring the message routing and service invocation activities.

7.5.11 Access Controller

An *access controller* building block is responsible for providing authentication/authorization facility for service invocation and message routing.

7.5.12 Data Transformer

A *data transformer* ABB is responsible for transforming data formats, e.g., from proprietary to industry standard format and vice versa.

7.5.13 Protocol Transformer

A *protocol transformer* ABB is responsible for transforming protocol formats, e.g., from SOAP/HTTP to SOAP/MQ or SOAP/JMS. It is a direct sub ABB of the data transformer ABB.

7.5.14 Exception Handler

An *exception handler* building block is responsible for handling exceptions raised in the process of service invocation and message passing. Note that this ABB does not handle any message-specific errors that are normally covered by message exception handling capabilities (e.g. SOAP exception handling facility).

7.6 Dependencies and Interactions (patterns)

This section describes the interactions or services Integration layer provides to other layers of the SOA Solution stack.

Table 3. Dependence and Interactions Overview for Integration Layer

Consumer	Business Process	Services	Service Components	Operational Systems	Integration	Quality of Service	Data Architecture	Governance
Consumer can access /invoke services	Processes implement their activities through service invocation. Integration layer supports the invocation of these services. It also does what is called service mapping which is a combination of protocol conversion, message transformation and dynamic routing.	It will serve as an access point for functionality that is not intrinsically enabled to act as a service (i.e. it does not support a standard interface in WSDL, it cannot be invoked over a standard protocol like SOAA/HTTP, etc).	It will provide (1) message mediation - message flow and transformation (2) Protocol switching (3) Routing (4) Monitoring (5) utility services such as auditing, logging	It will help virtualizes functionality in legacy applications. It will adapt service messages to whatever native protocol and format is supported by existing applications. In summary, it will provide support for integration of non-service enabled applications	N/A	It will provide (1) support for assured message delivery (2) support for transactions (3) support for service level management	It will provide data transformation (schema mapping)	Roles (identified by Governance Model) that will be required for Integration layer are – Adapter Developer, Solution Architect, Integration Developer and Solution administrator.

7.6.1 Diagram within the Integration Layer

Figure 38 uses a UML component diagram to illustrate a static view of the relationships between the ABBs within the Integration layer. All identified ABBs are represented as components in the diagram, with "ABB" as stereo type. The Integration layer is represented as a package containing all identified ABBs. In order to better organize and illustrate the relationships between the ten identified ABBs, one generic ABB is identified, as shown in Figure 38: integration controller ABB, which is the super ABB for the two ABBs: service mediator ABB and message router ABB. Note that message transformer ABB is the super ABB for data transformer ABB and protocol transformer ABB. Relationships between an ABB and a super ABB implies the corresponding relationships between the ABB and all the sub ABBs. For example, as shown in Figure 38, the one-to-one relationship between integration controller ABB and data aggregator ABB implies two one-to-one relationships: (1) a one-to-one relationship between data aggregator ABB and service mediator ABB and (2) a one-to-one relationship between data aggregator ABB and message router ABB.

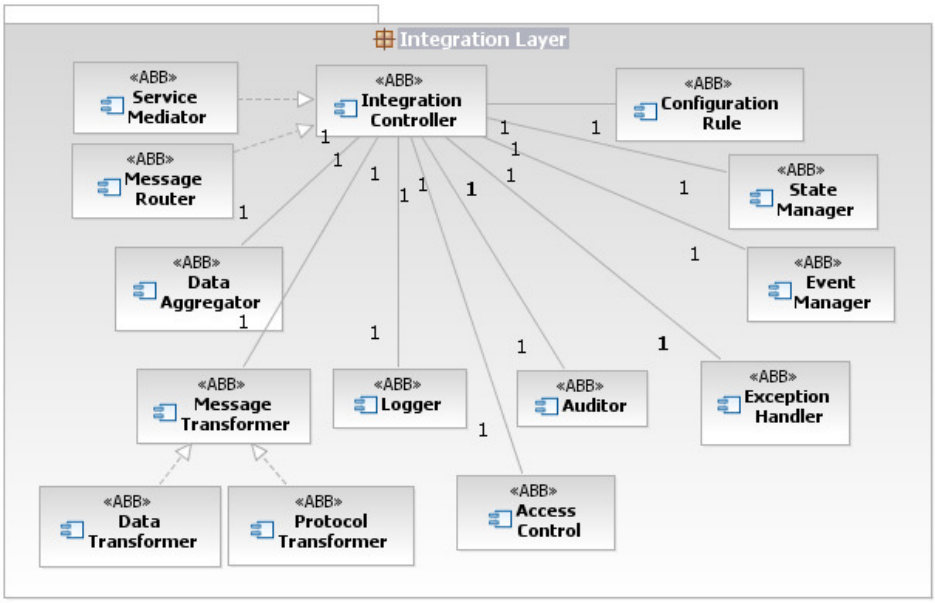

Figure 38. *Component Relationship Diagram – ABBs within the Integration Layer.*

Certain relationships exist between the identified ABBs:

- There is a one-to-one relationship between integration controller ABB with all other ABBs: data aggregator ABB, message transformer (including data transformer ABB and protocol transformer ABB), logger ABB, access control ABB, auditor ABB, exception handler ABB, event manager ABB, state manager ABB, and configuration rule ABB.

7.6.2 Relationship Diagram across Other Layers

Figure 39 uses a UML component diagram to illustrate a static view of the relationships for the ABBs across layers. Various layers identified in the SOA-RA are represented as packages; ABBs are represented as components.

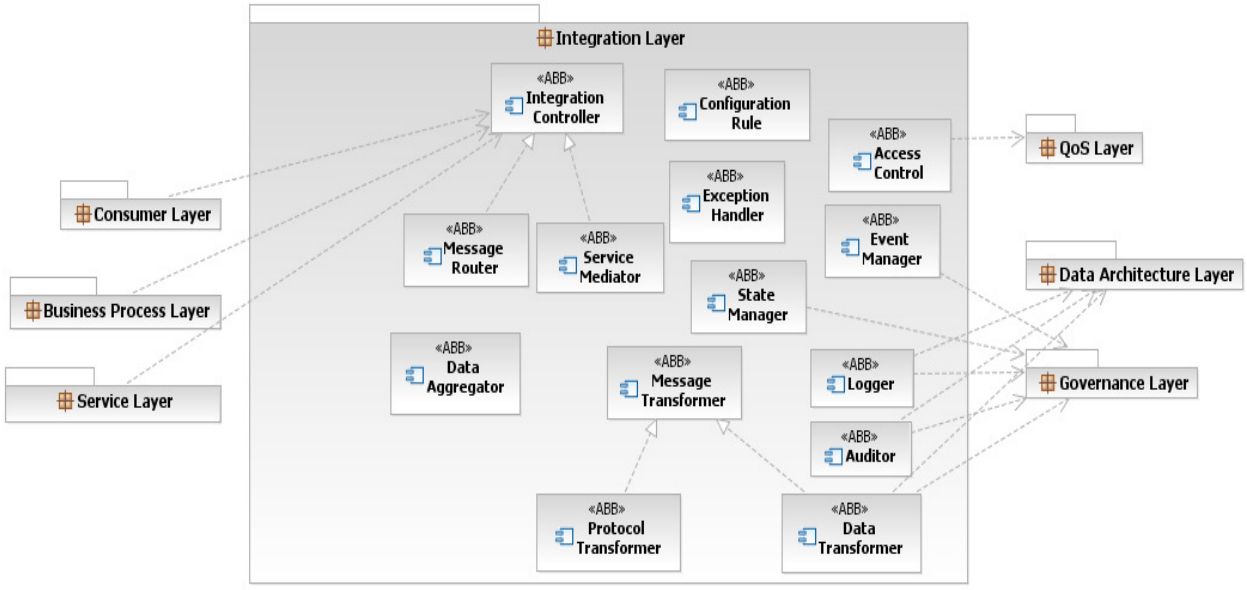

Figure 39. *Component Relationship Diagram – ABBs in the Consumer layer across layers.*

Certain relationships exist between the ABBs in the Integration layers with other layers:

- An *integration controller* ABB (including service mediator ABB and message router ABB) needs to interact with both the Consumer layer, the Business Process layer, and the Service layer.
- An *access control* ABB needs to interact with the Quality of Services (QoS) layer.
- An *event manager* ABB needs to interact with the Governance layer.
- A *state manager* ABB needs to interact with the Governance layer.
- A *logger* ABB needs to interact with the Data Architecture layer and the Governance layer.
- An *auditor* ABB needs to interact with the Data Architecture layer and the Governance layer.
- A *data transformer* ABB needs to interact with the Data Architecture layer and the Governance layer.

7.7 Options and Design Decisions

The integration layer should provide connectivity to existing applications such as CICS, IMS, SAP, etc. that do not directly support service-style interactions. To achieve this, integration technologies that can be used are such as JDBC, FTP or EDI or adapters such as J2EE Connector Architecture (JCA) resource adapters of application specific adapters such as SAP Adapter, etc.

For the purpose of our discussion, we will consider adapters. The following are the key types of functions the adapters perform:

Technology Adapters: This type of adapters allows handling of service requestors and service providers that use technologies such as CORBA, COM, JDBC, JMS and EJB

Application Adapters: This type of adapter facilitate integration with package solutions e.g Siebel, Peoplesoft and SAP.

Legacy Adapters: This type of adapters facilitate exposing existing applications as services. This achieved through use of technologies such as CICS Transaction Server, IMS Transaction manger, etc.

Some of common questions regarding adapters location are – (1) is it part of integration layer (2) is it part of existing application (3) somewhere in the middle - neither part of integration layer or application or (4) on the boundary of integration layer as shown in figure below.

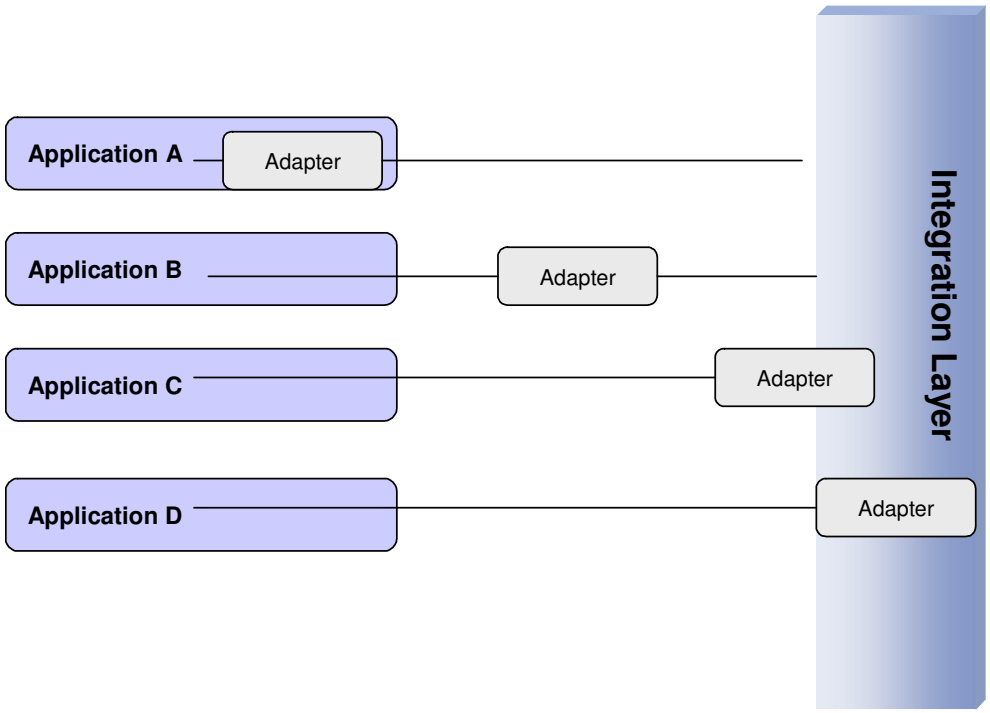

Figure 40. Adaptors for Application Integration

To answer these questions we need to analyze the situation. In general, IT artifacts such as adapters, APIs have some configuration that determines their behavior. If this configuration is managed by the Infrastructure management infrastructure, then the adapter is within the infrastructure.

In some instances, an adapter can be either outside or boundary of the integration layer depending on whether it is managed or partly managed by the integration infrastructures. An example of a partly managed adapter could be an integration layer that is built on J2EE application server with J2C Adapter to CICS Transaction Server using CICS Transaction Gateway in Server or Client Mode. CICS Transaction Gateway runs as a separate process with its own configuration and management, but the J2C end is application server control.

On the other hand, if we build the Integration layer using product such as WebSphere Business Integration Message Broker, linking to the J2C adaptor that is running in Application Server, the J2C adaptor is outside of the Integration layer.

The last placement option is the least interesting from the Integration layer perspective, but nevertheless a common scenario. In this case, the adapter configuration is managed in the Application A environment, and is thus invisible to the Integration layer and its management environment. Thus, from the Integration layer perspective there is no adapter at all, since the Application A component is "wrapped" by the adapter which interacts with the Integration layer using one of its natural connectivity mechanisms. An example of this sort of adaptation is the SOAP for CICS wrapper to allow web service-style interactions with CICS transactions.

In some cases, an ESB cannot effectively handle transformations to presentation oriented message formats. Additional metadata may be needed that may be not contained in particular message or may need special client logic to cooperate with legacy logic that is not explicitly reflected in service contracts.

7.8 Activities

In this layer one could model mediation patterns and communication interfaces e.g JMS, etc.

8 Layer 7: Quality of Service.

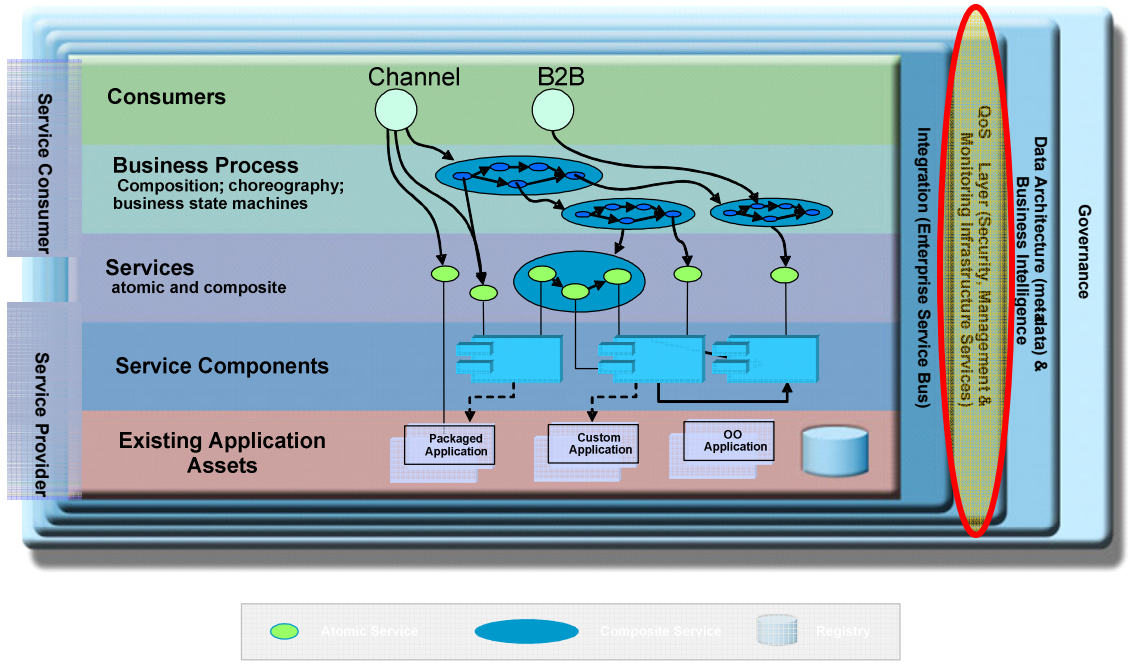

Figure 41. Quality of Services Layer in S3

8.1 Layer Description

The Quality of Services (QoS) layer provides solution-level QoS management in various aspects, such as availability, reliability, security, and safety. Note that this layer does not focus on service-level QoS control; instead, this layer concentrates on providing a mechanism to support, track, monitor, and manage solution-level QoS control.

Inherent in SOA are characteristics that exacerbate existing QoS concerns in computer systems: increased virtualization / loose coupling, widespread use of XML, the composition of federated services, heterogeneous computing infrastructures, decentralized SLAs, the need to aggregate IT QoS metrics to produce business metrics etc. are the nature of SOA. These characteristics create complications for QoS that clearly require attention within any SOA solution.

This layer serves as an observer of the other layers and can emit signals/events when a non-compliance condition is detected or (preferably) when a non-compliance condition is *anticipated*.

8.2 Value Proposition

Layer 7 establishes NFR related issues as a primary feature/concern of SOA and provides a focal point for dealing with them in any given solution. It provides the means ensure that an SOA meets its requirements with respect to: reliability, availability, manageability, scalability and security. Finally, it enhances the business value of SOA by enabling

businesses to monitor the business processes contained in the SOA with respect to the business KPIs that they influence.

8.3 Characteristics

QoS concerns are not localized in a particular area of the architecture; therefore the QoS Layer has some interaction with each of the SOA Architecture layers.

Maintain and monitor business continuity: ensure availability, reliability and failover subsequent to disaster recovery (see architectural decisions below)

Table 4: QoS Characteristics by Layer

5 Consumer	4 Business Process	3 Services	2 Service Components	1 Operational Systems	6 Integration	7 Quality of Service	8 Data Architecture	9 Governance
SLAs, business KPI monitoring	Security, conventional IT QoS concerns[5], business KPIs, business events	Service management, policy description	Security, conventional IT QoS concerns, IT events, SLAs	Security, conventional IT QoS concerns, IT events, SLAs	policy enforcement	Security, Policy Descriptions; SOA management descriptions; monitoring;	Security, disaster recovery, conventional IT QoS concerns	Service life-cycle policy, security architecture

Monitoring

The QoS layer has the capability to monitor, analyze and manage the salient QoS factors related to each layer in the SOA. In most of the layers (those that are operational) conventional IT QoS concerns are present, i.e. the need to maintain and monitor business continuity: ensure availability, reliability and failover subsequent to disaster recovery is just as present in SOA as it is in non-SOA architectures and the QoS Layer should provide the means to address this need.

Business KPI Monitoring

At Layers 3 through 5, concerns related to business performance metrics enter the picture. Failed/successful transactions, computing resource consumption levels, network outages etc. are not meaningful metrics in the context of business processes. On the other hand, their significance in business terms is meaningful, e.g. product orders received, improved/diminished customer satisfaction, average shipping delay etc. In other words, at a higher level, these IT events influence the key performance indicators (KPIs) that drive business process design, the same KPIs that businesses use to gauge their effectiveness. The business process layer should be able to measure these KPIs such that monitoring capabilities in the QoS layer can be used to analyze business performance.

Security

The QoS layer is responsible for the enforcement of security in an SOA. The case for appropriate levels of security in IT systems is well understood. The problem of achieving an appropriate level of security is made more complex in SOA due to the federated nature of many SOAs and the potential for a single service to participate in several unrelated business

[5] the various "*abilities*" that are typical in IT including: manageability, reliability, scalability, availability, traceability and performance

processes. The security architecture employed within an SOA should consider these additional dimensions.

8.4 Key Performance Indicators

Service components are a managed, governed set of enterprise assets that are funded at the Enterprise or the Business Unit level. In any case, as Enterprise-scale assets they are responsible for ensuring conformance to service level agreements through architectural best practices. This layer typically uses container–based technologies such as application servers to implement the components, has high-availability, load-balancing, etc.

- reflect a service description
- funded, governed, monitored, managed software resource
- abides by the Enterprise Component pattern

8.5 Architectural Building Blocks

As shown in , we identify a set of sixteen fundamental Architectural Building Blocks (ABBs) in the QoS layer: (1) s-QoS manager, (2) representation, (3) capability manager, (4) s-QoS context, (5) s-QoS enablement, (6) s-level instance manager, (7) life cycle manager, (8) delivery time, (9) execution cost, (10) delivery environment, (11) relationship, (12) s-QoS content, (13) reliability, (14) availability, (15) security, and (16) safety. As shown in , the s-QoS manager is the centralized controller of the QoS layer coordinating other ABBs. It has six first-level child ABBs: representation ABB, s-level instance manager ABB, life cycle manager, s-QoS enablement ABB, capability manager ABB, and QoS context ABB. The enablement ABB is associated with another ABB: s-QoS content ABB. The capability manager ABB in turn contains four sub ABBs: reliability ABB, availability ABB, security ABB, and safety ABB. The s-QoS context ABB in turn contains four sub ABBs: delivery time ABB, execution cost ABB, delivery environment ABB, and relationship ABB.

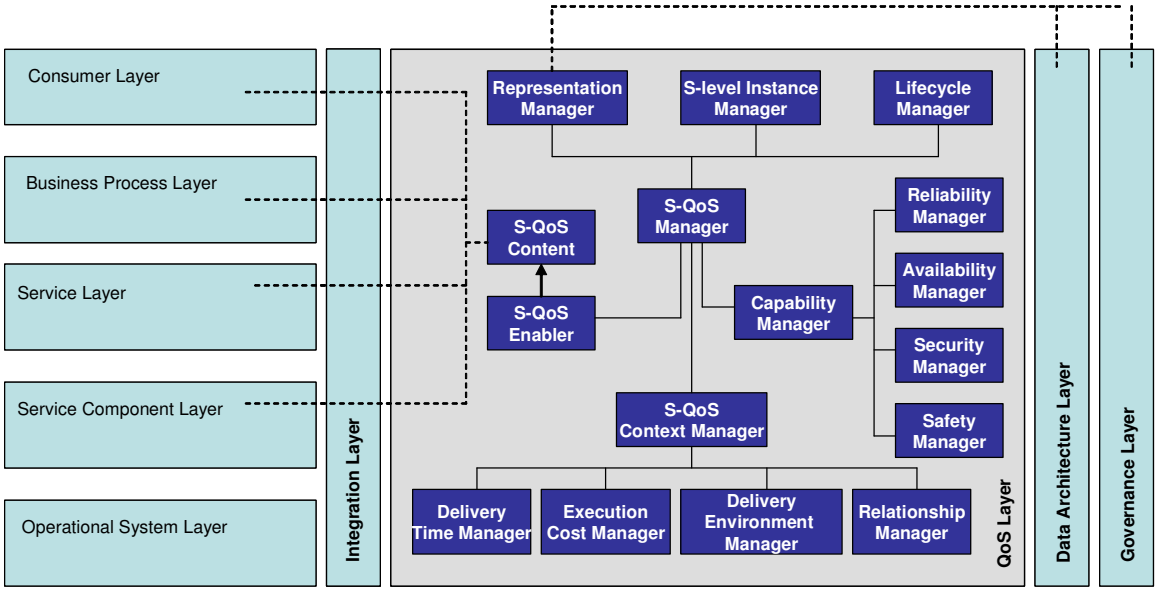

Figure 42. *QoS layer ABBs.*

8.5.1 Component Descriptions – QoS Layer

This section describes in detail each ABB in terms of its responsibilities.

8.5.2 S-QoS Manager

An s-*QoS manager* building block is the center of the QoS layer. It coordinates all other ABBs. It is called an s-QoS manager instead of a QoS manager, because it refers to solution-level QoS management instead of service-level QoS management. As shown in , the s-QoS manager ABB is a six-element tuple <representation manager ABB, s-level instance manager ABB, lifecycle manager ABB, s-QoS enabler ABB, capability manager ABB, s-QoS context manager ABB>.

8.5.3 Representation Manager

A *representation manager* building block is responsible for providing a unified description interface for QoS requirements. Web Services Resource Framework (WSRF) and WS-Policy are two possible candidates.

8.5.4 Capability Manager

A *capability manager* building block is responsible for describing the detailed solution-level QoS requirements in terms of traditional software-related attributes. As shown in , four QoS capabilities are captured and handled as default sub-ABBs: reliability manager, availability manager, security manager, and safety manager. It should be noted that more software-related attributes can be plugged into the QoS layer as sub-ABBs for the capability manager ABB.

8.5.5 S-QoS Context Manager

An *s-QoS context manager* building block keeps contextual information for solution-level QoS management. It contains four categories of information managers: delivery time manager, execution cost manager, delivery environment manager, and relationship manager. This ABB implies that the solution-level QoS measurement and assessment are already based on surrounding contexts.

8.5.6 S-QoS Enabler

An *s-QoS enabler* building block is responsible for adapt solution-level QoS requirements into layer-specific QoS requirements. For example, a solution-level QoS requirement may imply different QoS requirement to the Consumer layer and the Business Process layer.

8.5.7 S-Level Instance Manager

An *s-level instance manager* building block is responsible for managing running instances of various layers under control of solution-level QoS requirements.

8.5.8 Lifecycle Manager

A *lifecycle manager* building block is responsible for manage solution-level QoS requirements during the period of a solution's lifecycle, from the time when the solution is delivered to the time when the solution is terminated or discarded.

8.5.9 Delivery Time Manager

A *delivery time manager* building block is responsible for recording, tracking, and monitoring the time needed to deliver a specific solution.

8.5.10 Execution Cost Manager

An *execution cost manager* building block is responsible for recording, tracking, and monitoring the cost needed to execute a specific solution.

8.5.11 Delivery Environment Manager

A *delivery environment manager* building block is responsible for defining the required running environment needed to deliver a specific solution.

8.5.12 Relationship Manager

A *relationship manager* ABB is responsible for defining SOA-oriented business relationships regarding a solution. The relationship should be considered as a layered structure. Four layers should be defined: business entity level, business service level, Web service level, and operation level. Business entity-level relationship refers to coupling relationships between business entities, their goals, objectives, and so on. Business service-level relationship refers to interaction relationships between business services. Web service-level relationship refers to interaction relationships between Web services. Operation-level relationship refers to supportive or temporal relationships between operations provided by services. These relationships are important supplement of functionalities for solution-level quality control.

8.5.13 S-QoS Content

An *s-QoS content* ABB is responsible for storing layer-specific <name, value> pair of QoS attributes and required values.

8.5.14 Reliability Manager

A *reliability manager* building block is responsible for handling the reliability feature of a solution. It refers to how many percents that a solution can be successfully executed without failure during a certain period of time.

8.5.15 Availability Manager

An *availability manager* building block is responsible for handling the availability feature of a solution. Since a solution here refers to an SOA-oriented business solution, it implies a network-based service accessible remotely. Due to unpredictable network features, a solution is considered with high availability if its delay is always below some predefined threshold.

8.5.16 Security Manager

A *security manager* building block is responsible for handling the security feature of a solution. Since a solution here refers to an SOA-oriented business solution, it implies a network-based service accessible remotely. Due to unpredictable network features, a solution is considered with high security if ensures authentication and authorization based upon proper roles.

8.5.17 Safety Manager

A *safety manager* building block is responsible for handling the safety feature of a solution. A solution is considered safe if it is protected against predefined types of failure, damage, error, accidents, and harm.

8.6 Dependencies and Interactions (patterns)

8.6.1 Relationship Diagram within the QoS Layer

Figure 43 uses a UML component diagram to illustrate a static view of the relationships between the ABBs within the QoS layer. All identified ABBs are represented as components in the diagram, with "ABB" as stereo type. The QoS layer is represented as a package containing all identified ABBs.

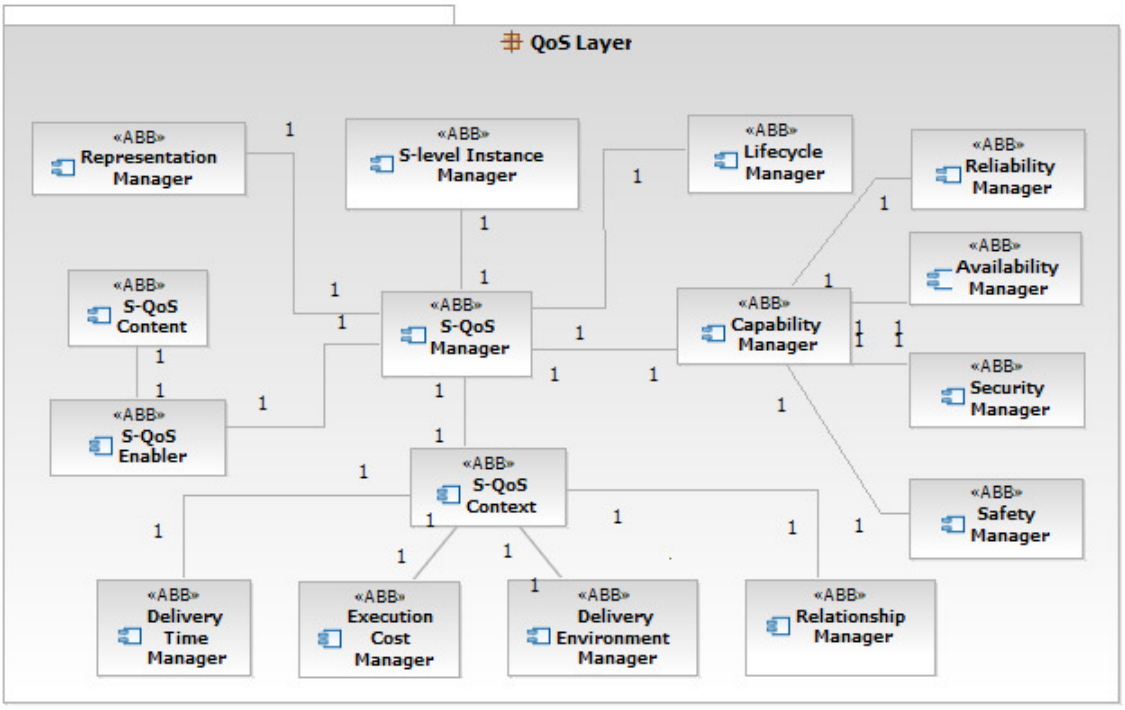

Figure 43. *Component Relationship Diagram – ABBs within the QoS Layer.*

Certain relationships exist between the identified ABBs:

- There is a one-to-one relationship between s-QoS manager ABB with a list of other ABBs: representation manager ABB, s-level instance manager ABB, lifecycle manager ABB, s-QoS enabler ABB, capability manager ABB, and s-QoS context manager ABB.
- There is a one-to-one relationship between s-QoS enabler ABB and s-QoS content ABB.
- There is a one-to-one relationship between capability manager ABB and a list of ABBs: reliability manager ABB, availability manager ABB, security manager ABB, and safety manager ABB.
- There is a one-to-one relationship between s-QoS context manager ABB and a list of ABBs: delivery time manager ABB, execution cost manager ABB, delivery environment manager ABB, and relationship manager ABB.
-

8.6.2 Relationship Diagram across Other Layers

Figure 44 uses a UML component diagram to illustrate a static view of the relationships for the ABBs across layers. Various layers identified in the SOA-RA are represented as packages; ABBs are represented as components.

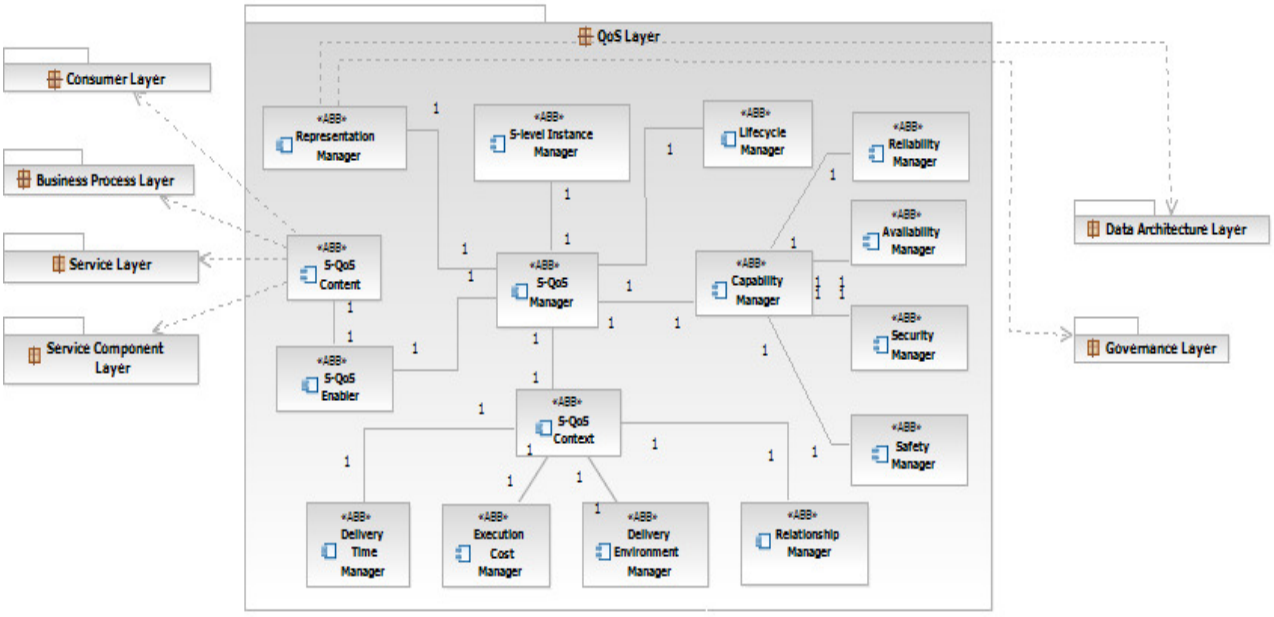

Figure 44. *Component Relationship Diagram – ABBs in the QoS layer across layers.*

Certain relationships exist between the ABBs in the QoS layers with other layers:

- An *s-QoS content* ABB interacts with the Consumer layer, the Business Process layer, the Service layer, and the Service Component layer.
- A *representation manager* ABB interacts with the Data Architecture layer and the Governance layer.
- It should be noted that the s-QoS manager ABB interacts with all other layers: the Consumer layer, the Business Process layer, the Service layer, the Service Component layer, the Existing Application Asset layer, the Integration layer, the Data Architecture layer, and the Governance layer. In order to simplify the diagram, these connections are not shown.

8.7 Options and Design Decisions

<options for instantiating ABBs, the architectural and design decisions relating to this layer>

8.8 Activities

<includes layer specific activities related to model, assemble, deploy and manage (life-cycle activities related to this layer)>

9 Layer 8: Data Architecture and Business Intelligence.

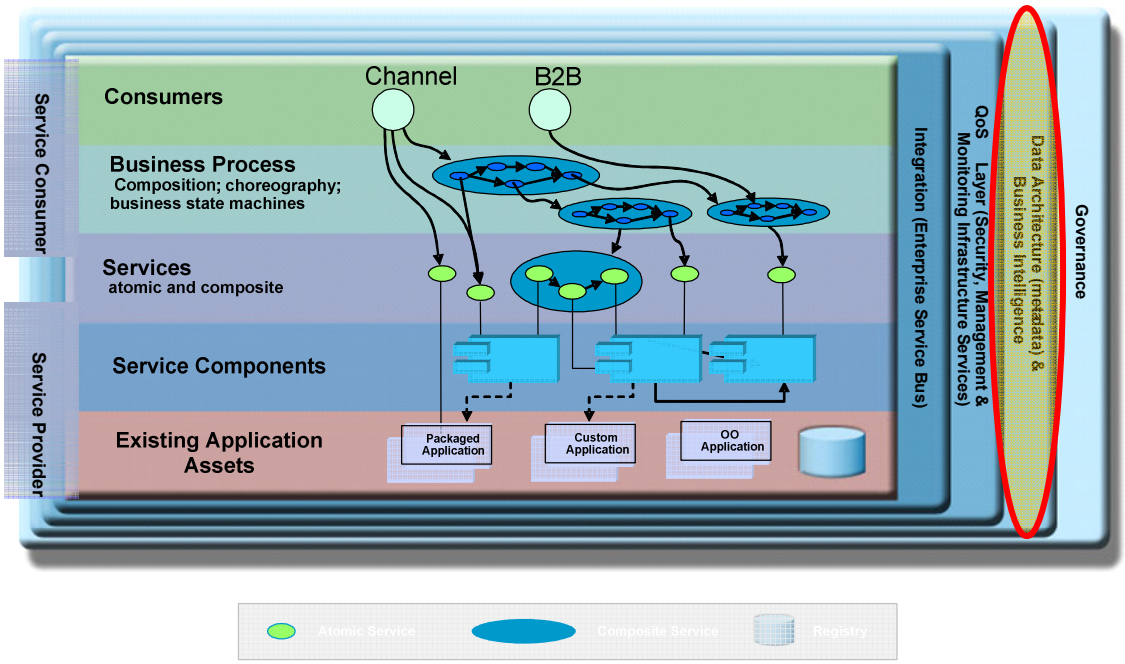

Figure 45. Data Architecture Layer in S3

9.1 Layer Description

The Data Architecture layer provides a unified representation and enablement framework that integrates with domain-specific data architecture to facilitate value chain integration. Typical domain-specific data architecture examples are: the Shared Information and Data model (SID) in eTOM of telecom industry, as well as the RosettaNet Technical Dictionary and RosettaNet Business Dictionary defined by RosettaNet for electronics industry.

This layer ensures the inclusion of key considerations pertaining to data architecture and information models that can also be used as the basis for the creation of business intelligence through data marts and data warehouses. This includes meta-data content that is stored in this layer.

Scope of definition of data; enterprise level data? Services will expose it in meaningful ways.

Note: each of the layers 1,6,7,8 have "registries" shown in them. For layer 1, this signifies the existing services that can be found in the Existing Application Asset layer. SOA will be built incrementally, project by project, in most cases. After one project is complete, another project can leverage the existing assets provided by the previous project. One of these assets is the actual services (e.g., Web Services) that were created in a prior project and are available for re-composition or realization of new (composite) services. Other registries or repositories or databases exist in other layers. For instance, the data architecture layer will have its databases within its tier. The quality of service layer will have (e.g.) ACL's or LDAP residing in registries for its operational/non-functional concerns.

This layer is responsible to manage access to data stored in any type of data storage. The data storage may be a reference data store, operational data store, data warehouse and data mart. The data can be stored in any types of data storage including relational databases.

Data services can be divided into 2 categories:

1. Core data operation services: This category includes Create, Read, Update, and Delete (CRUD) operations as well as data transformation services.
2. Data management services: This category includes services such as logging, subscription, and distribution services.

The *core data services* are encapsulated with the data content into *information component*. The information component may also include a data integration component. The data integration component may be implemented using WebSphere Information Integrator. The data integration component may include the metadata component. The metadata can also be implemented using data federation technology.

The data management services are included into the Enterprise Service Bus (ESB) along with other services.

9.2 Value Proposition

Previous attempts to make data sharing easier and more effective have centered around creating centralized data stores. However, those attempts have failed to provide the expected results because they delegated data operation services to data exploiters.

An SOA based approach to the data architecture focuses on data operations as well as data storage. Separation of data operations from applications and encapsulating them with data into information component or placing them into ESB make data sharing easier and feasible. This approach makes the data architecture to be application agnostic.

An SOA based architecture will bring its most value in an enterprise environment when data sharing is critical. In addition to IT cost saving, the SOA based data architecture will make it easier to integrate data across an enterprise (horizontal sharing of data) and transform data into actionable information and knowledge.

9.3 Characteristics

The data architecture layer for the enterprise is a managed, governed enterprise asset that is funded at the enterprise or the business unit level. If it is funded at the business level, it must fit into an enterprise level architecture framework. The data management services including logging that are placed in the ESB must be shared at the enterprise level.

The overall architecture should not require any knowledge of data structure to access the data. A search engine with capabilities like Google may be needed to get the retrieved data more useful to the user.

9.4 Key Performance Indicators (KPIs)

The KPIs for this layer are Latency, Availability, Scalability, Security, and data quality.

9.5 Architecture Building Blocks

As shown in Figure 46, we identify a set of seven fundamental Architectural Building Blocks (ABBs) in the Data Architecture layer: (1) data services gateway, (2) data aggregator, (3) data mining manager, (4) access control manager, (5) traceability enabler, (6) data representation manager, and (7) data sources manager. As shown in Figure 46, the data aggregator is the centralized controller of the Data Architecture layer coordinating other ABBs. The data services gateway ABB acts as the gateway of the Data Architecture layer. Other layers typically go through the data services gateway to obtain data-related services from the Data Architecture layer.

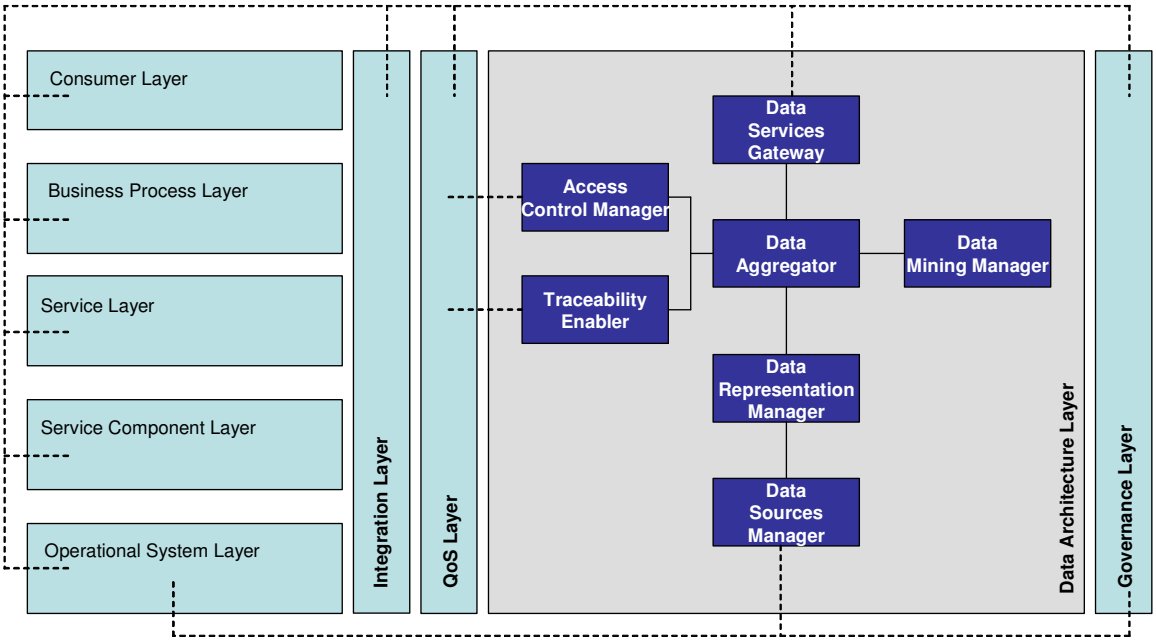

Figure 46. *Data Architecture layer ABBs.*

9.5.1 Component Descriptions – Data Architecture Layer

This section describes in detail each ABB in terms of its responsibilities.

9.5.2 Data Services Gateway

A data services gateway building block acts as the gateway (front desk) of the Data Architecture layer. It mainly has three responsibilities. First is to expose data as services. Second is to add/remove/manipulate data entries in different services or service components. Third is to disable some data from outside access.

9.5.3 Data Aggregator

A *data aggregator* building block is the federated data manager of the Data Architecture layer. Its major responsibilities are three-fold: first is to dispatch requests to other ABBs; second is to handle data transformation (including transformation of data types and contents); third is to aggregate data from multiple data sources.

9.5.4 Data Mining Manager

A *data mining manager* building block is responsible for analyzing data access history and providing optimization algorithms and business intelligence for data optimization.

9.5.5 Access Control Manager

An *access control manager* building block is responsible for handling access privileges of various participants to data. It typically involves authorization and authentication functionalities for registered participants. The access control manager ABB normally defines access control attributes. For example, the access control manager ABB handles who can see or access which portion of a data source (documents) and who can change it.

9.5.6 Traceability Enabler

A *traceability enabler* building block is responsible for monitoring and managing data usages using a log-like facility. Typical traceability log includes: who has accessed the data, when, and what part of the data has been accessed.

9.5.7 Data Representation Manager

A *data representation manager* building block is responsible for handling representing data from various data sources in a unified data format. In other words, the data representation manager ABB intends to hide various data sources and present data in uniform formats to other ABBs for data handling. It should be noted that the data representation manager ABB may link to various data sources and handle relationships between the data sources.

9.5.8 Data Sources Manager

A *data sources manager* building block represents the actual data repositories in various types, such as a DB2 database in the Existing Application Asset layer, or paypal representation, or an excel file. It should be noted that the data sources manager ABB in the Data Architecture layer refers to high-level links associated with metadata to real data sources in the Existing Application Asset layer. For example, instead of containing (e.g., attaching) a huge document, the data sources manager ABB here typically contains an on-demand link to the original document, together with some metadata describing the document (e.g., goals, purposes, and short descriptions) that help users decide whether he/she needs to access the original document (e.g., a CEO may decide not to download a detailed design document while a project architect may decide to download and review.) In addition, it should be noted that data sources manager ABB here typically represent industry-specific data structure; therefore, transformation may be needed for further processing.

9.6 Dependencies and Interactions (patterns)

9.6.1 Relationship Diagram within the Data Architecture Layer

Figure 47 uses a UML component diagram to illustrate a static view of the relationships between the ABBs within the Data Architecture layer. All identified ABBs are represented as components in the diagram, with "ABB" as stereo type. The Data Architecture layer is represented as a package containing all identified ABBs.

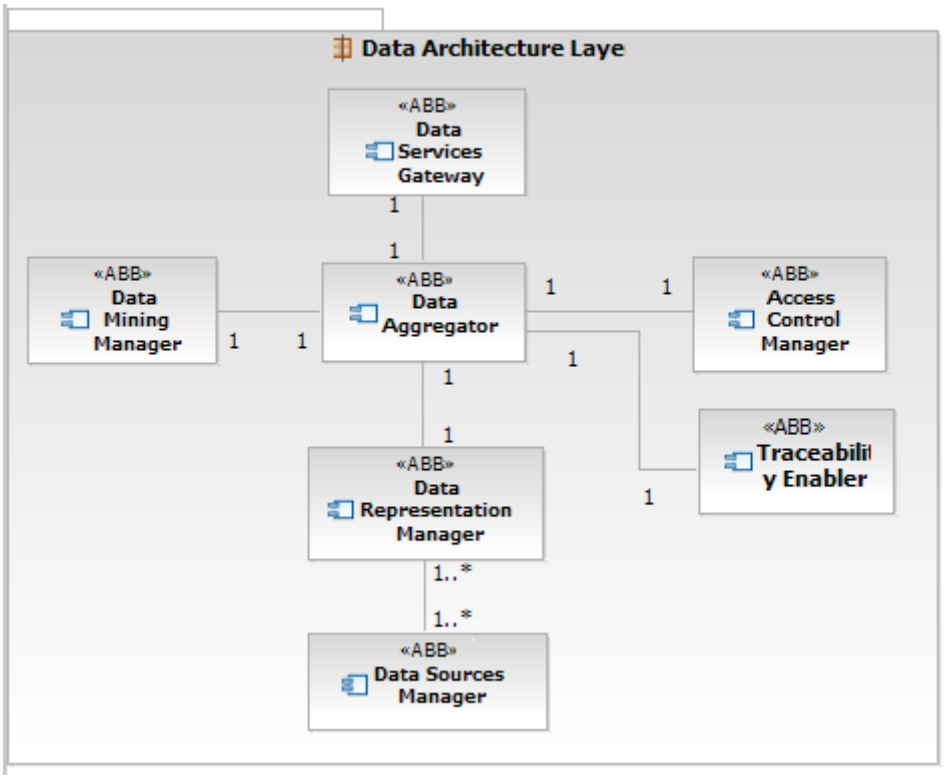

Figure 47. *Component Relationship Diagram – ABBs within the Data Architecture Layer.*

Certain relationships exist between the identified ABBs:

- There is a one-to-one relationship between data aggregator ABB with a list of other ABBs: data services gateway ABB, access control manager ABB, traceability enabler ABB, data mining manager ABB, and data representation manager ABB.
- There is a many-to-many relationship between data representation manager ABB and data sources manager ABB.

9.6.2 Relationship Diagram across Other Layers

Figure 48 uses a UML component diagram to illustrate a static view of the relationships for the ABBs across layers. Various layers identified in the SOA-RA are represented as packages; ABBs are represented as components.

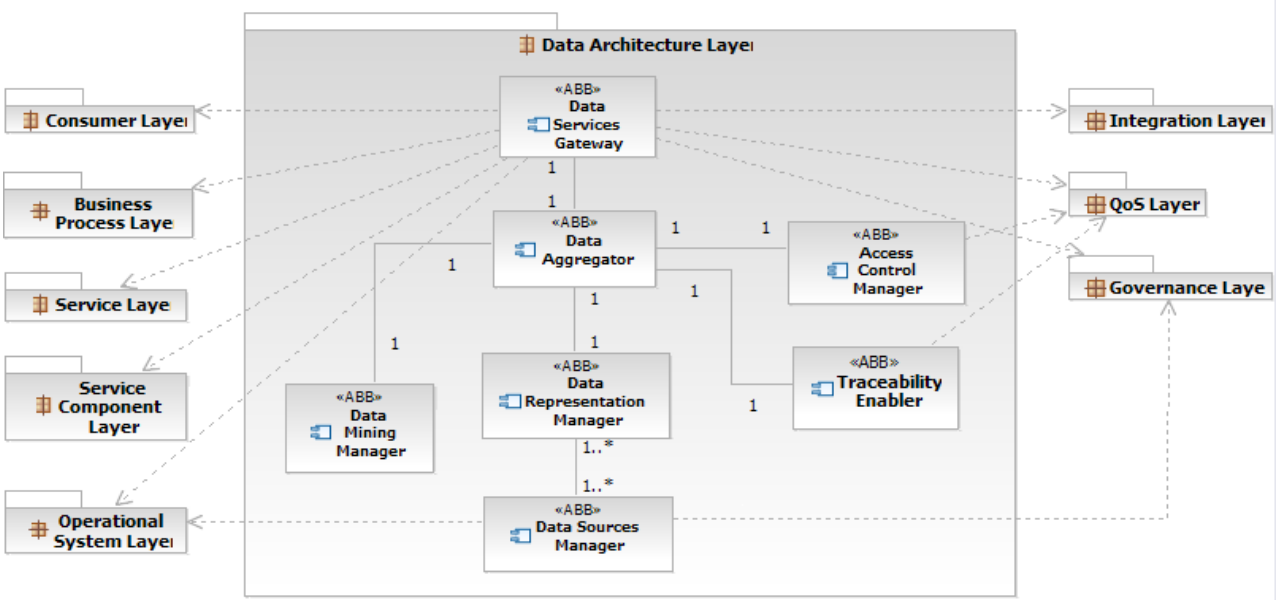

Figure 48. *Component Relationship Diagram – ABBs in the Data Architecture layer across layers.*

Certain relationships exist between the ABBs in the Data Architecture layers with other layers:

- A *data services gateway* ABB interacts with the Consumer layer, the Business Process layer, the Service layer, the Service Component layer, the Existing Application Asset layer, the Integration layer, the QoS layer, and the Governance layer.
- An *access control manager* ABB interacts with the QoS layer.
- A *traceability enabler* ABB interacts with the QoS layer.
- A Data Sources manager ABB interacts with the Existing Application Asset layer and the Governance layer.

9.7 Options and Design Decisions

This layer has three major design decision points:

- How many specific data models are required?
- How many common data models are required?
- How many transformations are required between the data models?

9.8 Activities

<includes layer specific activities related to model, assemble, deploy and manage (lifecycle activities related to this layer)>

10 Layer 9: Governance

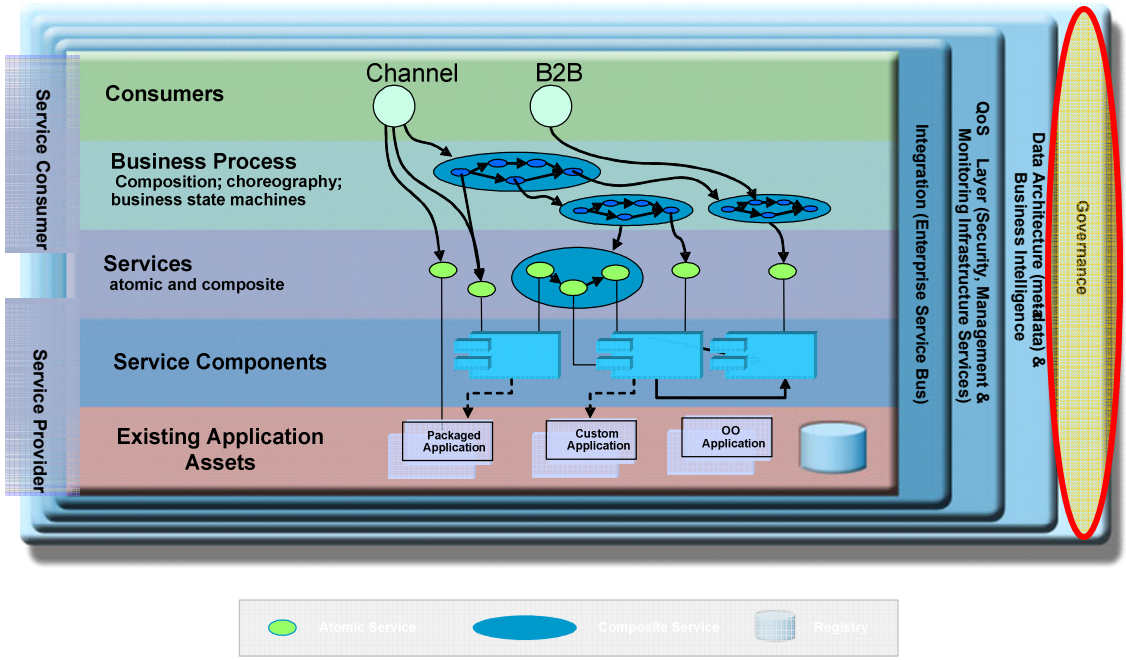

Figure 49. Governance Layer in S3

10.1 Layer Description

The Governance layer provides design guidance to ensure the proper design of the SOA solution architecture. Typically, the layer helps to establish best practices or their references (e.g., Center of Excellence of SOA/Web services), to establish principles of how to define SOA solution in each layer, and to establish principles of how to monitor in running system and how to handle exceptions at runtime.

This layer supports policy-time and run-time governance. Policy-time governance defines the reference architecture, policies about governance, compliance points, definitions of what conformance is, what projects warrant exceptions to compliance and how to maintain the relevance of the governance model over time.

In terms of runtime governance, this layer supports the notion of a service registry that is used to store the various versions of a service over time.
Governance describes policies and gives authority to management to monitor and execute on the policies as well as make decisions based on those policies.

10.2 Value Proposition

The value of this layer is to ensure that the mechanisms are in place to define, monitor and implement governance from en enterprise architecture and a solution architecture view.

10.3 Characteristics

This layer features the following characteristics:

Defines policies, compliance and exceptions characteristics
Monitor, the health of services should be monitored
Version, the ability to define versions of a service and invoke them from a registry
Report , report on compliance, exceptions, service health, versions

10.4 Key Performance Indicators

Four KIPs exist for this layer:

- Usage metrics of a service
- Downtime and failure statistics on a service or service set
- Policy violations
- Number of services compliant and number applied for exceptions

10.5 Architectural Building Blocks

As shown in Figure 50, we identify a set of six fundamental Architectural Building Blocks (ABBs) in the Governance layer: (1) governance services gateway, (2) governance enablement module, (3) governance definition module, (4) governance planning module, (5) governance measurement module, and (6) governance practice repository. As shown in , the governance enablement module ABB is the centralized controller of the QoS layer coordinating other ABBs. In addition, the governance services gateway ABB is the gateway between all the ABBs in the governance layer and all other layers: the Consumer layer, the Business Process layer, the Service layer, the Service Component layer, the Existing Application Asset layer, the Integration layer, the QoS layer, and the Data Architecture layer. Furthermore, the governance practice layer provides solution-level life cycle best practice to other ABBs: governance enablement module ABB, governance definition module ABB, governance planning module ABB, and governance measurement module ABB.

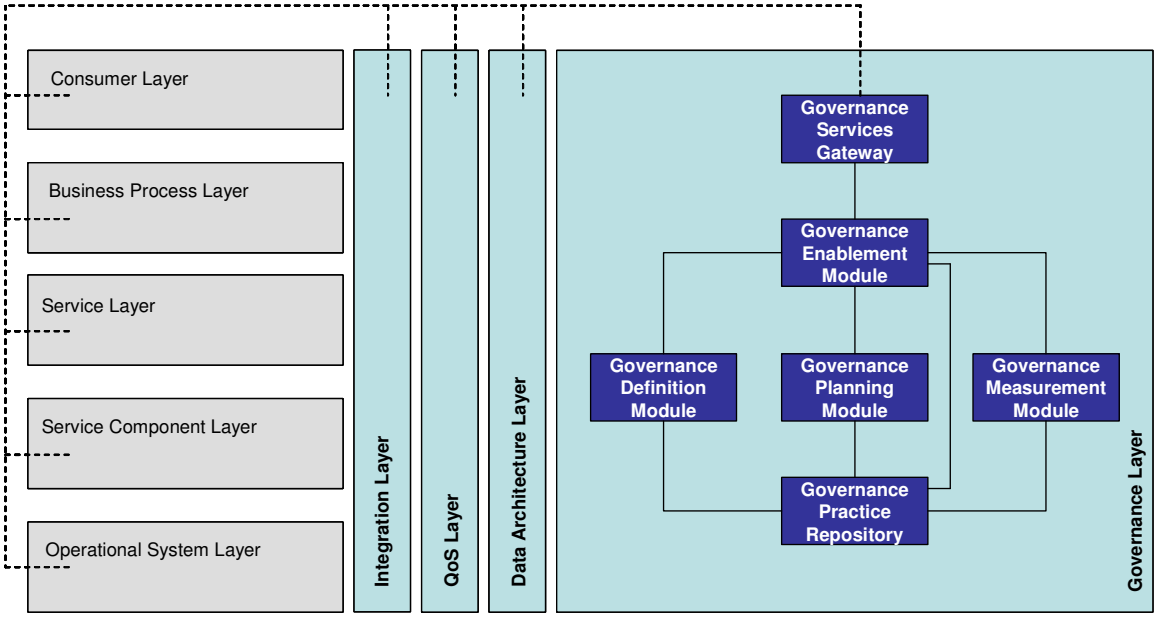

Figure 50. *Governance layer ABBs.*

10.5.1 Component Descriptions – Governance Layer

This section describes in detail each ABB in terms of its responsibilities.

10.5.2 Governance Services Gateway

A *governance services gateway* building block is the gateway of all the ABBs in the Governance layer to the other layers. In other words, it provides a focal point for processing both outgoing and incoming requests for governance management to and from the other eight layers. As shown in , the governance services gateway ABB connects to the other eight layers: the Consumer layer, the Business Process layer, the Service layer, the Service Component layer, the Existing Application Asset layer, the Integration layer, the QoS layer, and the Data Architecture layer. All the eight layers need to connect to the governance services gateway ABB in order to interact with the other ABBs in the Governance layer; all the other ABBs need to go through the governance services gateway ABB to interact with the other eight layers.

10.5.3 Governance Enablement Module

A *governance enablement module* building block is responsible for enabling and realizing solution-level governance control. In detail, this ABB fulfills the following six typical functions. First is to deploy the governance mechanisms predefined from the governance definition module ABB. Second is to deploy the governance IT infrastructure predefined from the governance definition module ABB. Third is to educate related participants the best practices (e.g., behaviors and experiences) pre-stored in the governance practice repository ABB. Fourth is to deploy into the system the governance policies predefined from the governance definition module ABB according to the deployment plans predefined from the governance planning module ABB. In addition, it should be noted that this ABB is the centralized control unit of the Governance layer.

10.5.4 Governance Definition Module

A *governance definition module* building block is responsible for defining a solution-specific SOA-based governance control model and strategy. In detail, this ABB fulfills the following six typical functions. First is to define solution-specific governance processes. If the

enterprise has some existing governance processes, this ABB is to refine them according to SOA concepts and requirements. Second is to design solution-specific governance policies (i.e., rules and protocols) and associated enforcement mechanisms. Third is to identify key success factors, usually in metrics. Key Performance Indicators (KPIs) are typically identified in this ABB. Fourth is to identify corresponding governance owners and funding models. Since the major goal of the governance layer is to ensure the success of the entire solution, a dedicated business unit may need to be established to ensure the responsibility with necessary funding. Fifth is to charter or refine SOA Center of Excellence. Related SOA best practices need to be accumulated to guide the specific solution. Sixth is to design an IT-level governance infrastructure, typically being a layered model-drive architecture.

10.5.5 Governance Planning Module

A *governance planning module* building block is responsible for analyzing, arranging, and scheduling solution-level monitoring and management. In detail, this ABB fulfills the following five typical functions. First is to elicit, document, and validate business strategy for the specific SOA-based solution. Second is to evaluate and assess current and available IT and SOA capabilities. Third is to define or refine specific SOA vision and strategy accordingly. Fourth is to review current governance capabilities and arrangements, and define or refine specific SOA-based governance monitoring and management tasks. Fifth is to layout a solution-specific governance plan.

10.5.6 Governance Measurement Module

A *governance measurement module* building block is responsible for monitoring and managing solution-level system status according to predefined governance policies and plans. In detail, this AGG fulfills the following three typical functions. First is to monitor system status to ensure its compliance with predefined policies. Second is to monitor system status to ensure its compliance with predefined governance arrangements. Third is to monitor system status to ensure it compliance with predefined IT effectiveness metrics (e.g., KPIs).

10.5.7 Governance Practice Repository

A *governance practice repository* building block is responsible for storing best practices for SOA solutions from various perspectives, including organizational aspect, development aspect, and run-time operational aspect, and life cycle management aspect. In other words, all ABBs have corresponding best practices. The *governance practice repository* ABB thus provides guidance to all ABBs.

10.6 Dependencies and Interactions (patterns)

Table 5. Dependence and Interactions Overview for Governance Layer

Consumer	Business Process	Services	Service Components	Operational Systems	Integration	Quality of Service	Data Architecture	Governance
Conformance of valid channel types and technologies and protocols	Traceability of Services to Business processes and Business Goal Service Model	Versions of Service	Valid Service Component Types and Architecture	Valid Package selection criteria; operational SLA's	Access Rights for Consumer Types	operational SLA's	Data Models Meta-Models Analytical Models	Policies governing Services

10.6.1 Relationship Diagram within the Governance Layer

Figure 51 uses a UML component diagram to illustrate a static view of the relationships between the ABBs within the Governance layer. All identified ABBs are represented as components in the diagram, with "ABB" as stereo type. The QoS layer is represented as a package containing all identified ABBs.

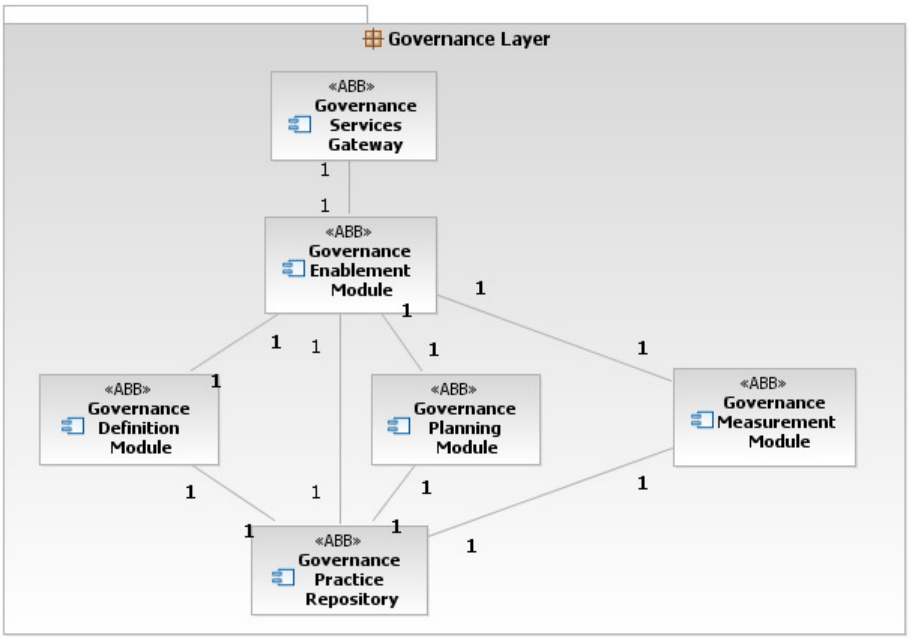

Figure 51. *Component Relationship Diagram – ABBs within the Governance Layer.*

Certain relationships exist between the identified ABBs:

- There is a one-to-one relationship between governance enablement module ABB with all other ABBs: governance services gateway ABB, governance definition module ABB, governance planning module ABB, governance measurement module ABB, and governance practice repository ABB.
- There is a one-to-one relationship between governance practice repository ABB with four ABBs: governance enablement module ABB, governance definition module ABB, governance planning module ABB, and governance measurement module ABB.

10.6.2 Relationship Diagram across Other Layers

Figure 52 uses a UML component diagram to illustrate a static view of the relationships for the ABBs across layers. Various layers identified in the SOA-RA are represented as packages; ABBs are represented as components.

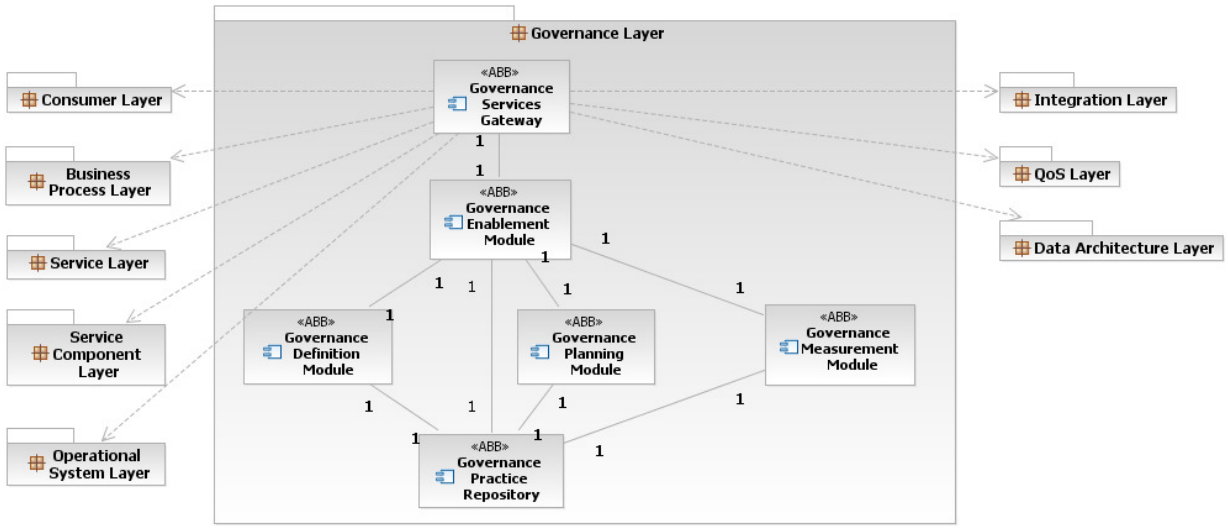

Figure 52. *Component Relationship Diagram – ABBs in the Governance layer across layers.*

Certain relationships exist between the ABBs in the Governance layers with other layers:

- A *governance services gateway* ABB interacts with the Consumer layer, the Business Process layer, the Service layer, the Service Component layer, the Existing Application Asset layer, the Integration layer, the QoS layer, and the Data Architecture layer.

10.7 Options and Design Decisions

Four design decision points exist:

- The Use of a standard Service Registry, Repository versus roll-your-own
- Collaboration technologies for Communication and Vitality
- Automation of Service Life-cycle and Tracking
- Automation of Compliance and exception handling processes

10.8 Activities

Governance has four main processes:

1. Compliance. Conformance or exception processes and policies
2. Vitality. Ensure the continued relevance of key governance model constructs, such as reference models, life-cycle (activities, workproducts), etc.
3. Communication. Disseminate and educate the fundamental constructs of life-cycle, governance models, etc.
4. Life-cycle. Describe the activities, roles and workproducts as it relates to the modeling of services through out the life-cycle.

At the heart of these processes is the Service Model, the unifying concept that binds these elements to together and makes them relevant.

11 Normative Guidance

Includes layer-level for selected design processes for

11.1 Architectural Principle Guidelines

This section customizes and extends the "Architectural Principles" section from e-business Reference Architecture [7] in the domain of SOA, and provides high-level architectural principle guidelines when architects design SOA solutions.

Business drivers are the fundamental driving force behind all architectural principles, policies, and decisions. Based on the business goals and business environments that exist in modern enterprises, the following key business drivers have been identified and widely recognized:

- Time to market
- Technology as a driver for business
- Flexibility
- Ease of use and maintenance
- Buy vs. build decisions
- Reuse existing infrastructure
- Maximize language and platform neutrality
- Use proven technologies

11.1.1 Architectural Principle – Time to market

Category	General Architecture	**Topic**	Time to market
Principle/Policy	Architectural decisions should be made with an eye towards accelerating the time to market of solutions that are based on this architecture.		
Explanation	Architectural designs using SOA-RA intend to support SOA-oriented applications that need to be launched in short time frames. Typically, these applications are expected to release new or modified functionalities every 8 to 12 weeks. Architectural decisions should be geared towards enabling these aggressive time frames.		
Relevant Requirements			
Motivation	The pace of evolution of business models and technologies in the marketplace forces a modern enterprise to innovate and respond to market pressures and customer expectations to stay ahead of its		

	competition.
Implications	• Certain architectural features and functions may have to be deferred to meet an established time line. • There is a need for rigorous application development processes and an application development methodology to support rapid and concurrent application development. • There is a need for reuse of assets to speed up the process of developing and deploying applications.

11.1.2 Architectural Principle – Technology as a driver for business

Category	General Architecture	**Topic**	Technology as a driver
Principle/Policy	Architectural decisions should be driven by their importance and relevance to the ability to meet the goals and objectives of the business.		
Explanation	Architectural decisions should enable the business's abilities to meet its commitments to the marketplace and not because the technology is cool or the latest that is available for use.		
Relevant Requirements			
Motivation	The need to enhance the business' ability to respond to market pressures and be able to set the pace in the industry. This means that technology decisions should be made to enable this goal as opposed to slowing it down.		
Implications	Decisions regarding architectural purity may have to be deferred or benched to meet the business goals and priorities.		

11.1.3 Architectural Principle – Flexibility

Category	General Architecture	**Topic**	Flexibility
Principle/Policy	The architecture should be extensible to include new products and technologies.		
Explanation	The architecture should be able to easily adapt to changes in underlying technologies, support newer versions of products, and enable new business requirements.		
Relevant Requirements			
Motivation	The pace of technology evolution in IT field demands a flexible architecture that can easily incorporate new products available.		
Implications	The application architecture should be layered and should encapsulate the product-specific details of the implementation into separate components.		

11.1.4 Architectural Principle – Ease of use and maintenance

Category	General Architecture	**Topic**	Ease of use and maintenance
Principle/Policy	The architecture should promote ease of use and maintenance.		
Explanation	The architecture should simplify the technology underpinnings of the architecture and infrastructure, the process of developing applications that are based on this architecture, and the tasks that are required to maintain and update the architecture on an ongoing basis.		
Relevant Requirements			
Motivation	The current infrastructure, while functional, is both complex and tedious to maintain and upgrade. By simplifying the architecture and eliminating some of the redundant and competing service components, we can make the architecture both flexible and extensible. One of the unwanted side effects of more sophisticated and intelligent software and hardware technologies is that they are more complicated and difficult to use.		
Implications	The number of middleware options need to be limited to reduce the total cost of ownership of the overall solution which includes: development, integration and testing costs, hardware and software costs, support and training costs, and ongoing maintenance costs.		

11.1.5 Architectural Principle – Buy vs. build

Category	General Architecture	**Topic**	Buy vs. build
Principle/Policy	Buy vs. build Infrastructure components of the application.		
Explanation	Components and services that are typically available from outside vendors should not be rebuilt. Examples of these components are: connectivity, load balancing, monitoring, and logging.		
Relevant Requirements			
Motivation	Custom developed middleware and infrastructure components are time-consuming to develop and maintain. Moreover, these components require a great deal of testing and documentation in order to be understood by a team of developers. Finally, developing these custom components takes away valuable time from the task of building business components and logic that can provide an enterprise with an edge over its competition.		
Implications	Eliminate custom middleware within the architecture over time by replacing them with products and technologies that are available from outside vendors.		

11.1.6 Architectural Principle – Reuse existing infrastructure

Category	General Architecture	**Topic**	Reuse existing infrastructure
Principle/Policy	Reuse existing services and components.		
Explanation	In order to meet time commitments to the business, it is necessary to reuse as much as possible of the current infrastructure, with the aim of gradually replacing custom developed components with off-the-shelf components in future releases of the application.		
Relevant Requirements			
Motivation	Reuse is required to launch the new or enhanced enterprise applications in a specified time frame.		
Implications	The application should adopt a phased implementation strategy, with the first set of releases focusing on new business functions by reusing existing back-end systems, system databases, and infrastructures. Some of the architectural enhancements should be defer to future releases.		

11.1.7 Architectural Principle – Maximize language and platform neutrality

Category	General Architecture	**Topic**	Maximize language and platform neutrality
Principle/Policy	Maximize language and platform neutrality.		
Explanation	Platform and language neutrality is essential to create a flexible and extensible architecture that can be morphed dynamically to suit ever-changing business and IT requirements.		
Relevant Requirements			
Motivation	It is difficult to predict the volumes and workloads of Web-based applications. Keeping the architecture platform neutral provides the ability to dynamically scale up or scale down the infrastructure that supports the application.		
Implications	Open standards should be considered for building the application as opposed to more closed technologies that limit choices to specific platforms.		

11.1.8 Architectural Principle – Use proven technologies

Category	General Architecture	**Topic**	Use proven technologies
Principle/Policy	The architecture should be based on proven technologies.		
Explanation	The architecture should be based on technologies and products that are proven and that can handle the operational requirements (e.g., scalability and transaction volumes) of the application.		
Relevant			

Requirements	
Motivation	Even though there are many promising technologies that seem to be on the horizon, it is unwise to bet the business by basing the architecture on these new technologies. By staying with proven technologies that are open and flexible, current requirements can be met and position the enterprise to use these new technologies as they become widely accepted by the marketplace.
Implications	Avoid emerging areas.

11.2 Other Aspects of Principle Guidelines

In addition to the above key architectural principles as normative guidance, following is an additional set of general architectural principles to use as guidelines for design decisions:

- Complexity hiding - wrapping and encapsulation
- Simplicity and conciseness
- Capability to interoperate with existing systems
- Component computing - delivery of functionality as simple, reusable components
- Component isolation - components should be loosely coupled to other components
- Design for "mass-production" in development
- Design of "separation of concerns" in development - there should be distinct roles played in the development process (e.g., Hypertext Markup Language (HTML) developer, Component Developer, and Component Assembler)
- Processing should be placed as close to the data as possible
- Design for distinct layering of presentation/presentation logic, business logic and data/application services
- Design for scalability enabled by distribution and scaling at each distinct level
- Design for reliability

11.3 Layer-Specific Normative Guidance

This section will provide on-going layer-specific normative guidance.

11.3.1 Services Discovery in the Services Layer

Federated services discovery from multiple services registries

11.3.2 Business Process Re-engineering in the Business Process Layer

Top-down

11.3.3 Services Composition in the Business Process Layer

Bottom-up

12 Modeling End-to-End SOA Solutions Using S3

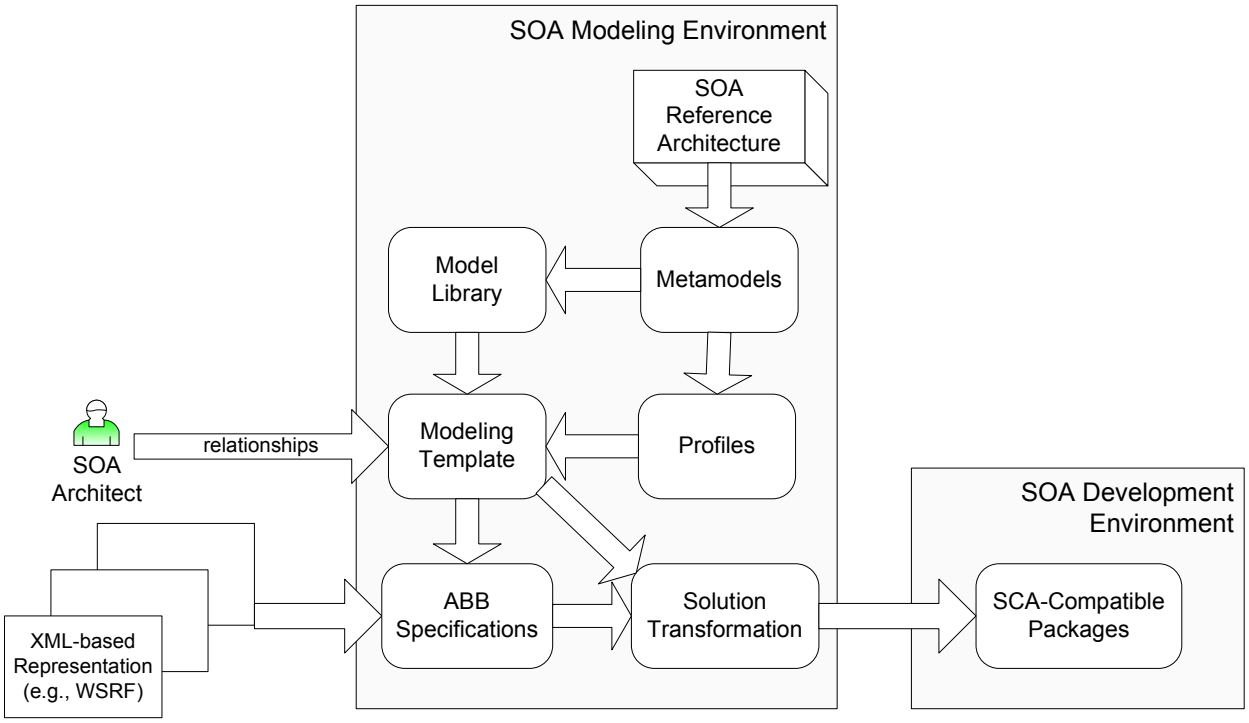

Figure 53. *End-to-End SOA solution modeling.*

Figure 54 shows an end-to-end SOA solution design and modeling framework. As shown in Figure 54, the SOA modeling environment roots in an SOA reference architecture, which possesses a set of metamodels organized in layers. The metamodels can generate model libraries that in turn generate modeling templates; the metamodels also generate profiles to guide and regulate user interactions with modeling templates.

In order to facilitate reusability and interoperability, an SOA solution comprising ABBs is represented in XML-based formats, for example, in WSRF to favor stateful ABBs. Since WSRF does not support relationships modeling between components, an SOA architect needs to interact with the SOA modeling environment to customize generated modeling template for the relationships between identified ABBs.

Customized SOA models comprising modeling templates and specifications are then executed model-to-solution transformation. Resulting SCA-compatible code packages, which can be imported into SOA development platforms, such as IBM WebSphere Integration Developer (WID).

We can now summarize how to model an end-to-end SOA solution using the S3 model. Our methodology comprises a six-phase procedure, as shown in Figure 54: (1) create a default S3 patterns; (2) identify and specify services using techniques such as SOMA; (3) export identified and specified services to the default S3 model; (4) customize each layer; (5) attach additional diagrams if necessary, and (6) export customized S3 model into an SCA-compatible solution skeleton. As shown in Figure 54, this procedure may go through many rounds of iterations between phases. Since each step is supported by automatic model and code generation, the iterations will not create too many overheads and potential information loss.

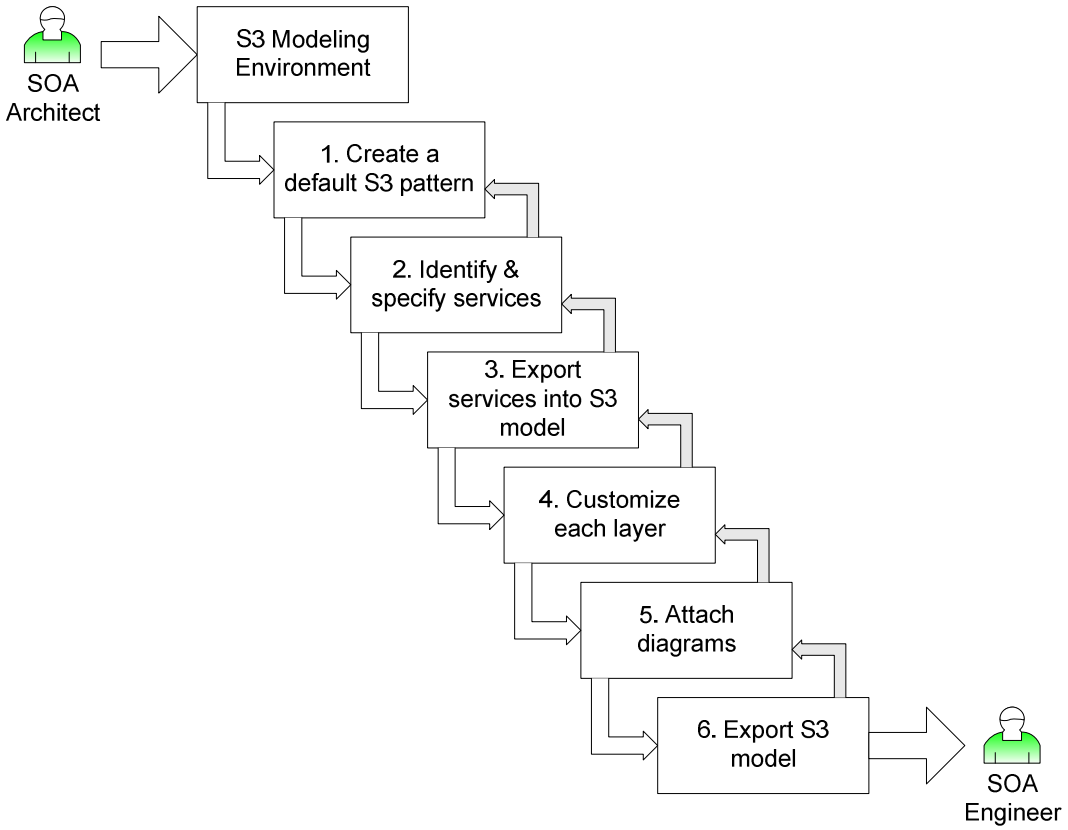

Figure 54. *Methodology of modeling end-to-end SOA solutions using S3.*

First, we create a UML project with a complete nine-layer S3 model comprising instances of all ABBs together with their relationships with each other. Second, we go through each layer and customize the default pattern using one of the aforementioned approaches, by either extending default ABBs with sub-ABBs or replacing default ABBs with customized ABBs.

Third, we attach additional diagrams to some ABBs if it is necessary. In the Business Process layer, existing services are to be organized in a synergistic manner using business logic. The ABBs with their relationships of the Business Process layer is shown in Figure 55. An instance of the control flow ABB is responsible for carrying a specific

business logic scenario, which can be used to guide process composition, decomposition, and collaboration.

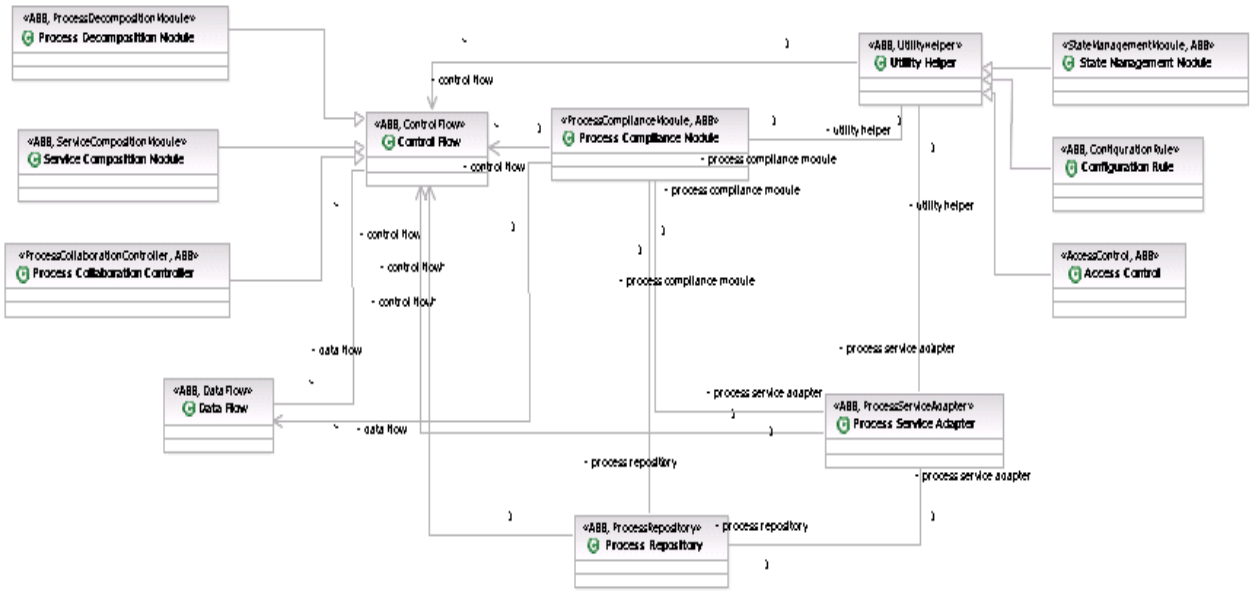

Figure 55. *ABBs identified for the Business Process layer.*

Take the travel booking example again. The business logic to compose services in the example can be represented by a UML activity diagram as shown in Figure 56. It can be seen that the activity diagram illustrates the business logic described in Figure 56. A travel booking process starts from invoking the credit card checking sub-process. Then three sub-processes are invoked in parallel: flight reservation, car reservation, and hotel reservation. Afterwards all results are aggregated and finish the process.

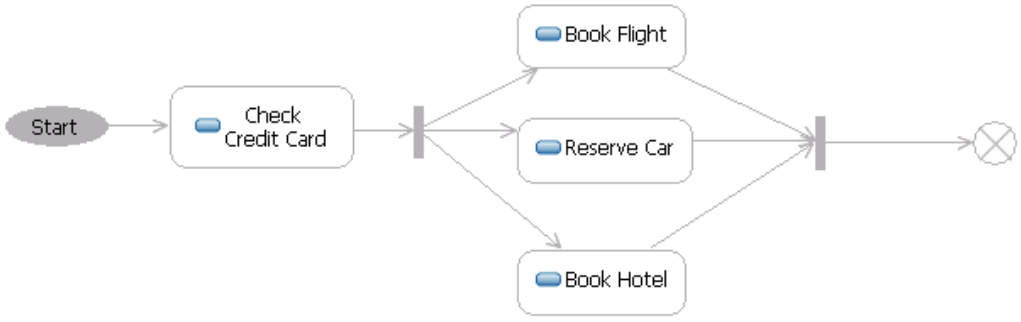

Figure 56. *An activity diagram for travel booking example.*

It is natural to associate the activity diagram with the control flow ABB. This activity diagram can be used to generate process algorithms for later development, for example, a Business Process Execution Language (BPEL) file.

13 The SOA Infrastructure: "Platform 9 ¾ "[3]

This layer is the underlying fabric and connection between layers that enables the interaction of the conceptual layers of the SOA.

13.1 Layer Description

A common occurrence with any architectural model is that a misunderstanding of the architecture's features/motivations can lead to failed attempts to realize the architecture.[6] This architecture is no different in that respect as it requires a very clear understanding of the role of each layer, the assumptions that each layer makes about the SOA infrastructure supporting it and the details that are ignored in each layer to focus it's architectural role.

Layers 1 through 5 of the SOA model are 'architectural abstractions' for salient features of the SOA value proposition. The features they support are:

- Channel independent access to business processes (layer 5)
- Business process alignment of IT (layer 4)
- Relaxed coupling of business and IT via implementation independent, business aligned service abstractions (layer 3)
- IT flexibility via Service aligned implementation façades (layer 2)
- Protection of existing IT investment (layer 1)

The value proposition associated with each of these layers provides the core concept around which to establish its scope of concern within an SOA and conversely the set of things that it is not concerned with but depends on the environment to provide. Any of these dependencies that are not within the scope of another layer constitute the requirements for the SOA Infrastructure (Platform 9 ¾). It is important therefore to understand the scope of each layer in order to drive out the nature of the SOA Infrastructure.

[6] This is often due to the "theory vs. reality" issue associated with architecture. By necessity, architectural models must be digestible and that often comes at the cost of 'glossing over' the details of realization. Realistically, those details cannot be ignored during implementation.

Table 6 SOA Value Propositions and the Features that support them.

Layer Value Proposition	Key features (to achieve value proposition) (italicized items are not necessary for small SOAs)	Assumptions about Environment (italicized items are not necessary for small SOAs)
Channel independent access to business processes	• Ability to consume a service definition and invoke a service in an implementation independent way • *Invocation mechanism is immune to service relocation side-effects*	• Service definitions are well-defined and available • Service invocation is uniform regardless of implementation • *Service implementations can be located using a reference to a given service definition*
Business process alignment of IT	• Ability to capture/model business process definitions independently of service implementation concerns • Ability to consume a service definition and invoke a service in an implementation independent way • *Immune to service relocation side-effects*	• Service definitions are well-defined and available • Service invocation is uniform regardless of implementation • *Service implementations can be located using a reference to a given service definition*
Relaxed coupling of business and IT	• Ability to capture the definition of a service in an implementation independent way. This definition should capture both functional and non-functional service traits. • *Ability to manage the storage of service definitions.* • *Ability to record service definitions in a centralized registry*	• *A service registry is available to record services and service implementations*
IT flexibility via Service aligned implementation façades	• Ability to create an implementation technology-specific service skeleton that conforms to an implementation-independent service definition • Ability to invoke operational systems in platform specific ways to provide service implementation. • Provide a platform independent (uniform) way to invoke a service component • Provide a selection of QoS features to suit the requirements	• *A service registry is available to capture the mapping between services and their implementations.*

	of a given service	
Protection of existing IT investment	• Capability to refactor existing applications etc. according to requirements of Service Components. • Capability to represent master data in a way consistent with the SOA • Data service enablement. Protection of existing IT investments including data and applications. Capability to render master data in a format appropriate for an SOA.	• Data needs to be encapsulated by the service. I.e. the data schema should not need to be exposed / changed in order to obtain access to the data. • Existing applications should perform the specified business functions without changes to the application itself. (we tend to build wrappers around existing applications to leverage investments in new ways)

Fundamentally, these layers separate concerns such that each is isolated from changes below the layers they consume and from changes in the layers that consume them. This isolation is a key factor in achieving the SOA value proposition. For example, one would not want to see a service implementation occur in the Business Process layer as that would fail to achieve the value proposition supported by level 3 and below, i.e. the opportunity to freely exchange one service implementation with another is lost because the implementation is coupled to the business process model[7].

For each of the inter-layer boundaries there is a set of requirements that the layer must address to meet its value proposition goals. Likewise, each layer has requirements of its consumers that make it possible for them to be consumers. These requirements, taken together, form the framework for realizing an SOA that can achieve the value propositions outlined above.

For this architecture, it is important to understand the assumptions made about each layer because those assumptions establish the scope of concerns for each layer.

There are 5 layers because there are 5 aspects of SOA that are managed/prone to change and hence require a level of abstraction to provide protection from update side-effects.

In effect, these layers establish a set of contracts that layout in very explicit terms the details for each abstraction within a given SOA. The abstractions then provide guarantees that insulate each side of a given layer from changes that might occur on the other side of the abstraction.

[7] This should not be confused with the possibility that business process technology might be used to *implement* a service, e.g. BPEL. The choice of implementation technology is a layer 4/5 decision. There is nothing wrong with building a service using BPEL as long as it conforms to a service abstraction/description contained in layer 3. Architecturally, one would say that such a service has been realized using business process technology but it is important to stress that it is not implemented in the business process layer.

For example, the consumer/service layer contract establishes a set of abstractions that permit service consumers to be built without concern for the implementation details of a given service; the service layer/component layer contract establishes a service definition conformant entry point for a service implementation in an SOA providing the service implementers the assurance that they have met the interface requirements of a service.
Understanding the nature of the contract associated with each pair of adjacent layers is an important step toward understanding the factors that are of primary concern for SOA architects and tooling designers when dealing with each layer.

Consumer/service contract requirements

- Open standards based – Rationale: promotes broader reusability of services/consumers
- Synergy with business process – Rationale: each service/service operation must integrate naturally into a business process
- Operations are self contained, non-stateful – Rationale: each operation makes no assumptions about the before and after states of the business process.
- Captures NFRs on which the business process depends – Rationale: NFRs such as ACID are fundamental dependencies for business processes

Service/component contract requirements

- Open standards based – Rationale: promotes broader reusability of services/consumers
- Free of implementation tech idiosyncrasies – Rationale: permits unencumbered selection of implementation tech
- Freely mappable to *generated* implementation artifacts. (e.g. JAX-RPC, .Net) – Rationale: permits unencumbered selection of implementation technology and simplifies the creation of this layer through tooling
- NFRs on which the business process depends – Rationale: NFRs such as ACID are fundamental dependencies for business processes

13.2 Value Proposition

Describe the underlying enabling mechanisms that allow invocation of services, business processes, defines the mechanisms by which the integration layer functions to integrate service consumers and service providers.

13.3 Characteristics

- SOA Infrastructure enables/allows layers to interact; it is the "layer in between the layers"
- Layers rely upon the SOA infrastructure to carry out their responsibilities
- Binding
  - Algorithms, version management, load balancing, etc.

- MQ to directly support SOAP in v 6.0
  - SOAP MQ (.NET)
  - Business: I need a reliable connection:
    - It solutions: MQ

13.4 Key Performance Indicators

<Generic KPIs that typically play in this layer>

13.5 Architectural Building Blocks

<major elements of the layer, including software and software>

13.6 Dependencies and Interactions (patterns)

< interactions among ABBs within and across the layers>

13.7 Options and Design Decisions

<options for instantiating ABBs, the architectural and design decisions relating to this layer>

13.8 Activities

<includes layer specific activities related to model, assemble, deploy and manage (life-cycle activities related to this layer)>

14 Layer Interactions

<insert table that maps layers to layers>
e.g the service layer maps to the Existing Application Asset layer via the service realization matrix.

14.1 Component/system contract

Note that for every pair of layers in this reference model, we have a layer of enabling, runtime, underlying technology that realizes the mechanisms that connect two layers. For example, the underlying invocation technology for the services described in the service layer (layer 3) must provide the runtime binding/activation support necessary to translate a service request into the execution of a software component in the service component layer (layer 2).
For each pair of adjacent layers, several runtime options may exist. The set of selection criteria used to choose one runtime option over another (or to perhaps support several simultaneously) is derived from the role of the layers in question. In the case of the service definition/service component layer boundary, these criteria include:
Service virtualization: provision for the relocation of the service implementation in a way that is transparent to the consumer.
Open standards compliance: the use of binding and invocation techniques that is defined with open standards and therefore likely to facilitate the replacement of one service implementation with another without any side-effect on the consumer.
Quality of Service: provision for requirements such as transactional integrity, security, reliability etc.
Layer Contracts : Each layer has a contract with its adjacent layer.

15 Frequently Asked Questions

15.1 What is the Relation of the SOA Reference Model to Existing Practices?

It is important to recognize that SOA systems and solutions are designed and implemented by building upon and leveraging *existing* techniques and technologies. These existing technologies and components have an associated set of best practices that may not specifically be related to SOA. For example, writing robust J2EE applications and components are an important part of building SOA solutions. In this paper, we will be primarily focusing on the areas that are critical success factors in building service-oriented architectures.

To introduce a controversial subject, some of the critical success factors in the design and implementation of an SOA are:

1. ***The proposed SOA meets client's functional and non-functional requirements***
2. ***The SOA defines a set of business-aligned IT services that supports an organization's business processes and goals***
3. ***The SOA provides a basis for a flexible integration architecture that can evolve with the business needs***
4. ***The SOA eliminates redundancy and provides single point of access to redundant back-end functionality in a way that does not compromise quality of service***
5. ***Consumers are able to use the identified services***
6. ***The SOA increases the agility and flexibility of the business and the ability of IT to meet business needs***
7. ***Consumers are actively engaged in using the identified services of the SOA***
8. ***Services decouple interface from potential multiple back-end implementation options***
9. ***Allocation of Components to Layers:*** The logical abstraction of a layer helps manage complexity and insulates change. It allows the centering of a responsibility for the execution of a software engineering "concern" to an abstraction that is a logical container, where solution components can be allocated and ABBs of a solution can be allotted. Components of a solution are allocated to layers and this allocation is in itself an architectural decision.
10. ***Allocation of Services to Components:*** A service will be implemented by a service component. That allocation of responsibility is another key architectural decision that will affect the success of the SOA in terms of functional and non-functional characteristics.
11. ***Identification of Services:*** Choosing which services to include in your SOA along with the right level of granularity for consumers is another key architectural decision that should follow an analysis process called Service Identification.
12. ***Specification of Services:*** Design the internals of the service portfolio, including operations, input and output messages and exceptions. Map data to the message content required by services.
13. ***Realization of Services:*** Source the implementers of the service from service components or from ISV packages or from existing back-end legacy systems in a way that supports the desired functionality in the allotted timeframes that the business requires them.

14. ***Monitoring and Management of Services:*** Monitor the health of services and manage their operations, trap and correlate events that reflect Key Performance Indicators that translate into business or IT value.
15. ***Governance of Services:*** Ensure compliance, communication, vitality and life-cycle through the institutions of SOA governance by providing checkpoints, reference models and architectures, methods and variation-oriented analysis and design to harvest commonality and externalize variations.
16. ***Use Externalized Policy Descriptions for altering runtime non-functional requirements and characteristics of services.***

15.2 Where, when, how would you use the S3?

The SOA Solution Stack's Reference Model can be used as:

- ***Learning aid:*** In this usage the reference architecture provides a tutorial of the essential ABBs for a domain of interest like a portal, self-service (e.g., internet banking), collaboration or network centric application, all of which apply SOA principles. That is, it defines the standard building blocks that the org has agreed upon as a set of standard products
- ***Architecture Definition Accelerator:*** In this usage the reference architecture accelerates the inception and elaboration phases of an SOA solution by shortening the time it takes to define and build the various assets described in the SOA reference architecture. The collection of work products that comprise the reference architecture shortens the architecture activities for projects
- ***Scoping Accelerator:*** In this usage the SOA reference architecture is used to scope the size of a project that will leverage the reference architecture. Using a workshop the gaps can be identified for the solution and a preliminary scope and sizing can be done for both the new software and hardware elements required to realize the solution. It also sets direction for web based or network centric development projects that apply SOA principles.
- ***Governance Accelerator:*** In this usage scenario the reference architecture facilitates technology selections (e.g., vendor RFI or RFP) as the reference architecture provides a set of standards and guidelines that can be reused on future projects. So in this case we use the reference architecture is used in the actual RFI or RFP to help make product selection choices. The reference architecture also provides both the product standards and design standards necessary for solutions that apply SOA.
- ***Governance and Management Accelerator:*** In this usage scenario the reference architecture is continuously improved based on actual project experiences. The Architecture Working Group or Enterprise Architecture Team or SOA Center of Competence uses the reference architecture to determine architectural compliance, to drive sharing and reuse, and to codify best practices from project implementations based on review cycles resulting in a continuously improved reference architecture.

15.3 How does the S3 fit within your current engagement models?

The usage of the S3 includes influencing and enabling the following workproducts and activities by providing:

- A categorization and checklist for key Architectural Decisions
- A blueprint for building SOA solutions, including creating building blocks for SOA which performs business processes. Architecture definition accelerator; establish what it looks like and filling in the blanks
- A mitigation factor in risk management; by ensuring a structure is in place based that is based on best practices, reference architectures, and allowing a gap analysis to be conducted with the current architecture that is being put in place. This helps to identify where are we deviating and thus identify the risks.
- Educational Aid in working with customers in helping them understand the elements of SOA
- As a reference architecture, we need to customize this architectural diagram for a particular client and particular solution area to meet their specific needs and maintain relevance to the project and problems at hand, which tend to vary with clients.
- An outlet in which SOMA activities can be seen to help create each one of the layers in the architecture, and how SOMA helps populate the layers, ABBs.

15.4 The Value Propositions of S3 and SOA

The usage of the S3 is a key enabler for the achievement of the value propositions of an SOA. Informally, the S3 answers an architect's questions such as :

"What are the layers I need to look at, building blocks I need to consider, architectural decisions I need to make when choosing to use a set of ABB's within a given layer."

These value propositions of SOA are as follows:

1. Business Agility: Ability to delivery business-aligned applications faster 1.

2. Reduce Cost: Through providing the opportunity to consolidate redundant application functionality and decouple functionality from obsolete and increasingly costly applications while leveraging existing investments

2. Increase Flexibility: Structure IT Applications based on services in such as way as to facilitate the rapid restructuring and reconfiguration of the business processes and applications that consume them.

3. Increase Revenue: Provide the opportunity to enter into new markets and leverage existing business capabilities in new and innovative ways using a set of loosely coupled IT services. Increase market share by offering new and better business services.

5. Consolidation: Integrating across silo-ed applications and organizations.

15.5 Is that not addressed by SOMA, for example? Where does SOMA end and S3 begin?

SOMA has a realization activity in which tasks are conducted including:

1. use the SOA Solution Stack as a architectural blueprint
2. Identify key scenario types based on how which layers will be use din which scenarios.
3.

"Who would be using these principles and guidelines?"

The S3 applies to various types of practitioners such as Enterprise Architects, Solution Architects, etc.

The S3 reference model is an abstract, logical design of an SOA. Thus it answers the question of "What is an SOA?" Architects can use it as a checklist of layers, ABBs and their relations in each layer, the options available, decisions that need to be made at each layer. The layers provide a starting point for the separation of concerns needed to build an SOA.

Clients and Industry have been presented with this architectural model during the past year (Sept 2004-2005). Publications have used this as a basis for further work independent of IBM. IT has become a de facto, unwritten reference.

This CommunityPaper documents the contributions of experts around the world from various lines of business within IBM who are members of the SOA & Web Services Community starting on September 14, 2004[8]. Please fell free to contact the community for suggestions and contributions.

15.6 Where is a home for infrastructure services?

The answer is the SOA Infrastructure Layer, we have dubbed Layer "9 ¾ ". The infrastructure services are a set of services that provide infrastructure functionality – such as Discovery (shown). But Discovery is probably not the only such service. Other example include Security service – to log in and obtain security tokens. Particular SOA may define specific sets of such services e.g. DataRecord, Authorization, whatever. These services are not part of the ESB and they are likely to be used by different layers to call other services.

15.7 Is a Service Component different from a Service Implementation?

No. If I am building brand new functionality as an atomic service that has fully self-contained implementation and does not depend on any legacy – then "component" is merely a unit of service realization that exists solely to provide partitioning of service implementations that are aligned with implementation platform, ownership, change rate, etc. In this case service and component are related as interface and implementation and there is no contract between these that we are concerned with at SOA level. If, on other hand I am building a service on top of something that already exists, then how we distinguish between "component" and legacy system. Say I have CICS app in front of a database and custom library of a Java code that sits in front of CICS and has been used before SOA to

[8] The 1st SOA Solution Stack Summit was held during December 2004 and forms the major source of input into this CommunityPaper. Results and presentations are available in the Community Knowledge café.

communicate to CICS app. What is a component in this case? It can be that Java library or it can be CICS app depending on what stays and what gets redone.
Or all we are saying is – component any kind of "internal" service that does not get thru rigor of service definition.

16 Fundamental Concepts

16.1 Perspectives of an SOA

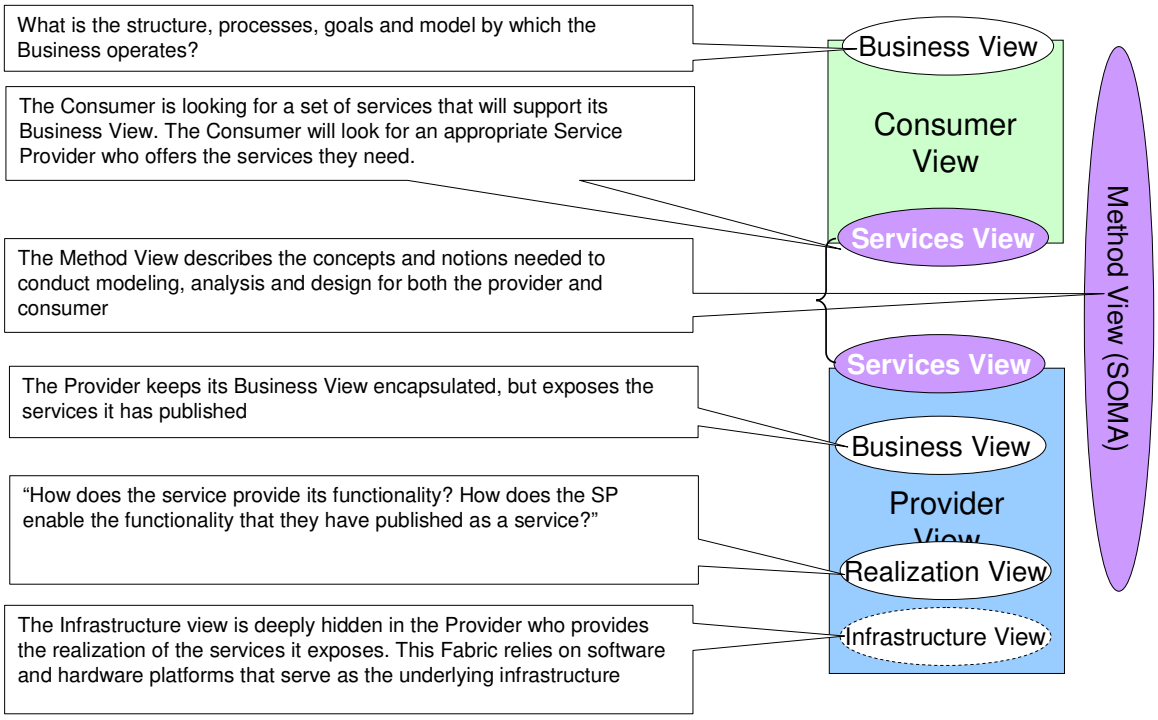

Figure 57. Method View

One of the key realizations of an SOA is the treatment of the worlds of software and business from three distinct and equally viable perspectives: that of the Service provider, Consumer and Broker.

Each provides a set of views converging in the Services View that holds that the main structuring element of an SOA is a set of business aligned IT services that collectively help fulfill an organization's business processes and goals.

16.2 The SOA Eco-system

To participate in an SOA or Services Eco-system a company would need to have a standard reference model such as that depicted by the S3, in order to facilitate the integration and collaboration of architectures across companies. Thus standardization would benefit companies at the architectural level just as it has benefited them at the level of data interchange via XML and XML Schema.

We would thus like to expand the perspective of SOA to include the creation of an SOA Eco-system:

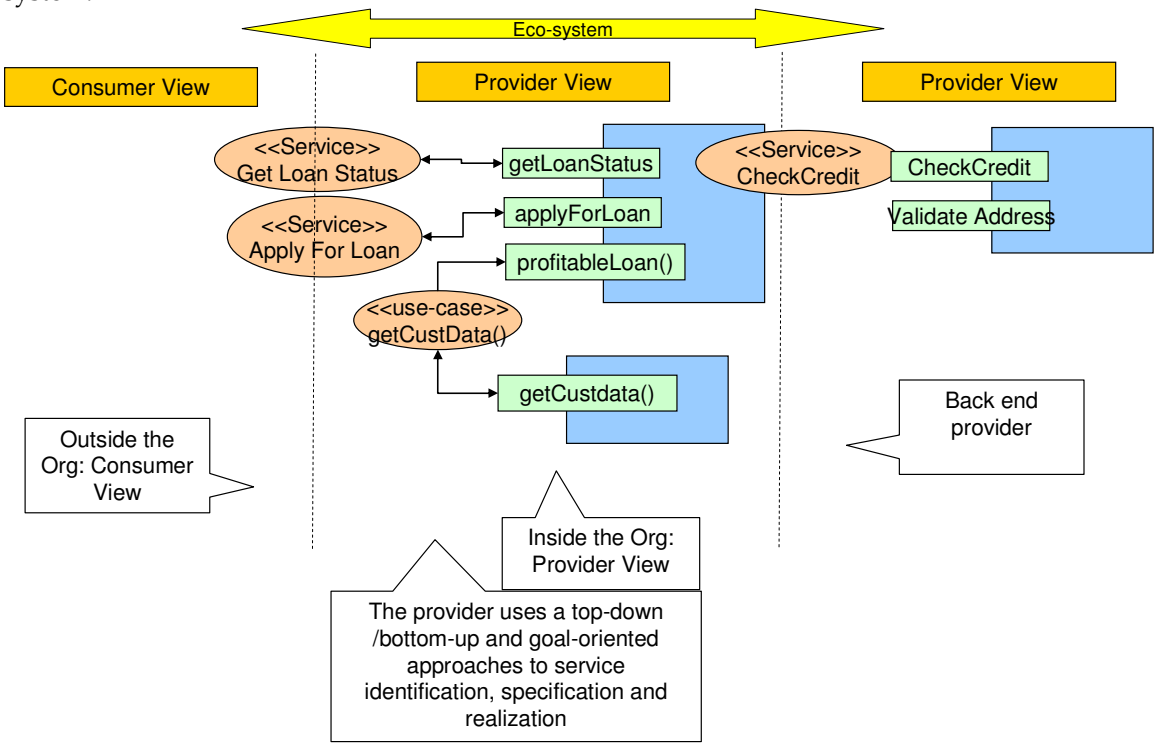

Figure 58. An Example SOA Ecosystem

The views of the consumer, provider and broker help us define how to participate in the SOA Eco-system. <1 stack view vs. multiple stack view… need to resolve this representation>

16.3 "Fractal" Usage Context

One other key notion we would like to discuss is that of fractal or recursive usage of SOA. A recurring observation in the context of SOA projects has been the applicability of SOA within multiple areas of increasing scope: a single project, a line of business, a few lines of business sharing services, enterprise scale, supply-chain (value-net) and a larger SOA Eco-system. In each case the principles of SOA tend to be applied in a similar manner. This self-similarity of application of SOA concepts is termed "fractal" usage.

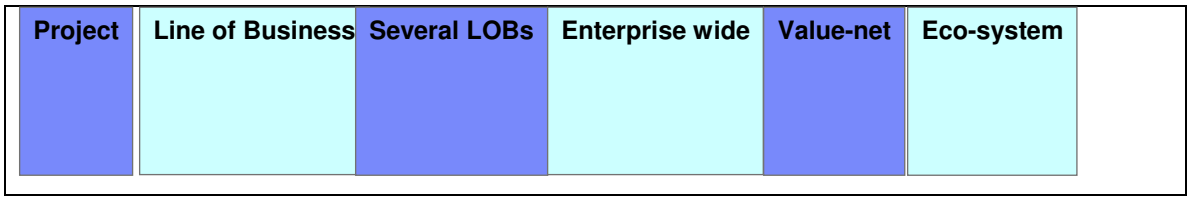

Figure 59. Fractal Nature of SOA Applicability

When we apply SOA, as defined in the above reference model, to a given level of granularity of an SOA Eco-system, we will typically find the need to create the same layers for each level of granularity. Thus, Enterprise Architecture might take the above reference model as a blueprint that will be customized or instantiated for each line of business or each product line (depending on how the organization is structured).

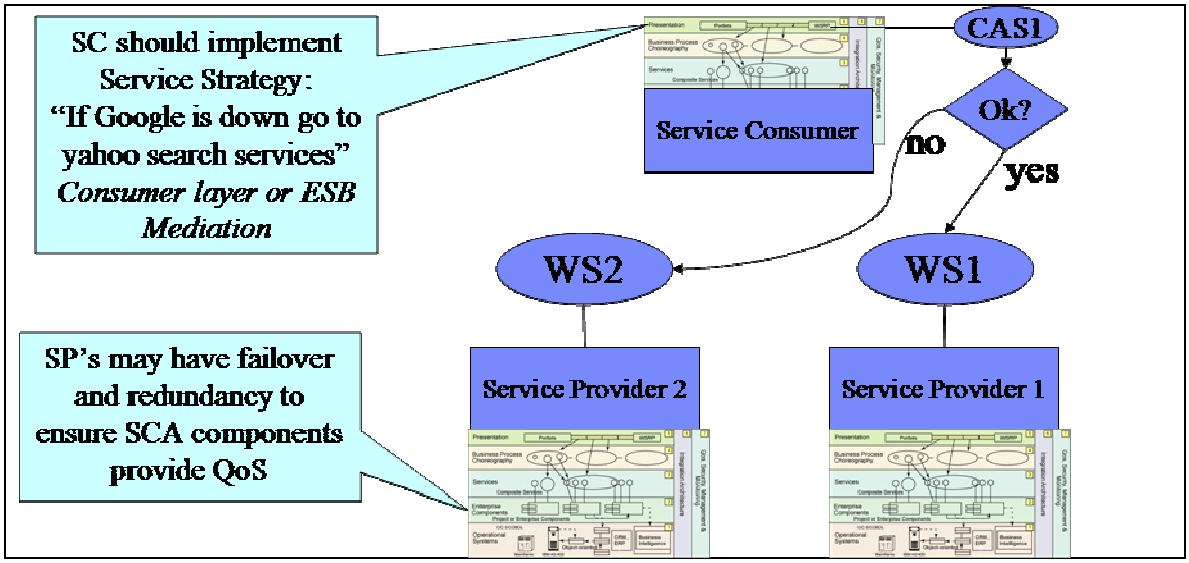

Figure 60. The reference model is defined for each organization participating in the SOA eco-system

17 Overview of the SOA Reference Architecture Work Products

The SOA Reference Architecture provides a collection of descriptive models and work products to support the development of SOA solutions

- BUS 309 – Process Definition
- BUS 312 – Process Identification
- BUS 315 – Business Environment
- APP 011 – System Context
- APP 110 – Logical Data Model
- APP 130 – Use Case Model
- APP 150 – User Profiles
- ARC 100 – Architectural Decisions
- ARC 101 – Architecture Overview Diagram
- ARC 108 – Component Model
- ARC 113 – Operational Model
- ARC 119 – Non-Functional Requirements
- ARC 319 – Performance Model
- OPS 316 - Security Architecture
- SOA 101 - Service Model

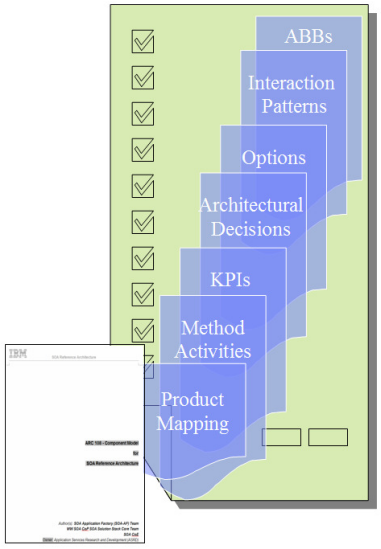

Figure 61. Enterprise View Diagram

the Enterprise View of the Architecture Overview Diagram depicts the high-level business layer and components of an SOA solution within the business contexts.
This view is mainly for high-level business people, such as C-level people.

This Enterprise View Diagram extends the one from the e-business Reference Architecture [ebRA] and applies to the SOA field to support SOA solutions.

As shown in Figure 62, the Enterprise View of the Architecture Overview Diagram depicts the high-level business layer and components of an SOA solution within the business contexts. This view is mainly for high-level business people, such as C-level people. This Enterprise View Diagram extends the one from the e-business Reference Architecture and applies to the SOA field to support SOA solutions.

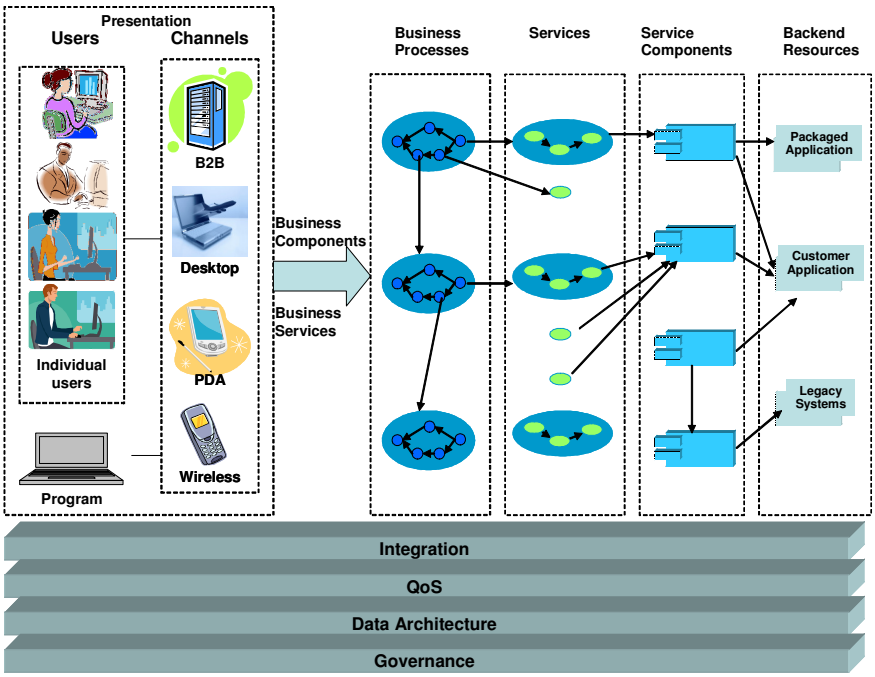

Figure 62. *Meta-model of the SOA Reference Architecture (SOA-RA).*

17.1.1 Presentation

A presentation element of an SOA solution provides the link between solution users and the solution system. In more detail, it allows users to access the system functions through various channels. In other words, the presentation element includes two major parts: users and channels.

As shown in Figure 62, users of an SOA solution include both individual users (e.g., users, administrators, and enterprise representatives) and programs for Business-to-Business (B2B, a.k.a., machine-to-machine) interactions. In other words, an SOA solution typically supports two types of users, individual users who access the solution manually or machine programs that access the solution through an automatic manner.

As shown in Figure 62, an SOA solution typically supports a variety of communication channels, including: (1) personal computer (desktop or laptop) equipped with an Internet

browser over the Internet, (2) Personal Digital Assistant (PDA), (3) wireless cell phones, and (4) servers for B2B communications. Other channels may be added if necessary.

17.1.2 Business Process

A business process element refers to workflows representing an enterprise-specific business-related procedure. As shown in Figure 62, a business service can contain multiple business processes. Similar to business services, business processes are technology-independent. As shown in Figure 62, existing business components and business services can be optional input. As shown in Figure 62, a business process may dependent on other business processes and services.

17.1.3 Service

A service element refers to reusable Web services that can be configured to construct business processes. These Web services have published interfaces in *ad hoc* standard languages (e.g., WSDL) and can be universally accessed through the Internet. Web services are hosted by their corresponding service providers. Comparing with business services and business processes, Web services are technology-dependent services and aim for generic usages. As shown in Figure 62, a business process can contain multiple Web services.

As shown in Figure 62, a service can be either an individual service or a composite service. An individual service refers to an atomic service that does not dependent on other services. A composite service refers to a service comprising multiple other services.

17.1.4 Service Component

A service component element refers to more low-level software components with clearly defined interfaces and can be used to build Web services. These service components can reside either in the Internet or Intranet. For example, a service component can be a wrapper over a legacy system. As shown in Figure 62, a Web service can contain multiple service components.

17.1.5 Backend Resource

A backend resource element refers to existing applications that can be used to build new business logic, including packaged applications, customer applications, and legacy systems. These backend resources can be wrapped into service components for higher-level constructions. Without the backed resources, new SOA solutions do not have to be built from scratch, saving significant man power and time to market. As shown in Figure 62, a service component can contain multiple backend resources.

17.1.6 Integration/QoS/Data Architecture/Governance

An integration element refers to the system facility that enables and formalizes component integration, routing, and interactions in a uniform manner. A Quality of Service (QoS)

element refers to the system facility that monitors, tracks, manages, and ensures solution-level quality of service requirements, including: reliability, availability, security, safety, maintainability, adaptability, and fault tolerance. A data architecture element refers to the system facility that enables solution-level smooth data structure and data interactions in a uniform manner. A governance element refers to the system facility that supervises and guides solution-level best practice and support.

18 References

[1] Six Approaches to SOA. Arsanjani, Toward a Pattern Language for SOA and SOI, IBM developerWorks.
[2] Service-oriented Integration-- SOI. Integrate applications through service interfaces (descriptions) rather than directly using traditional EAI methods. Provide integration using a bus rather than point to point or hub and spokes. Wrapper legacy to integrate it into the SOA eco-system. This can be done using the Patterns for SOA and SOI, such as Virtual Provider or Remote Service Strategy.
[3] Platform 9 ¾ is a tongue-in-cheek allusion to the Harry Potter novels where a platform between platforms was deemed to exist. The Infrastructure required of an SOA at runtime must provide communication, invocation and quality of service between adjacent layers in an SOA.
[4] Service – Integration Maturity Model. Arsanjani, Holley, PLTE2005,
http://community.usca.ibm.com/soawiki/Edit.jsp?page=PLTE2005ServiceIntegrationMaturityModel
[5] The Service Integration Maturity Model. IBM develoeprworks.
[6] Service Composition, Ali Arsanjani, developerWorks, Dec 2 , 2005
[7] e-business Reference Architecture V2.3, ARC 100 - ebRA-ArchitecturalDecisions - V2-3.pdf.

19 Appendix A: Composition and Functionality

This has been externalized in its own document. Please see [6].

20 Appendix B: Use-cases

We have recognized 6 types of use-cases for approaching SOA, and in this context, building SOA Solutions, e.g., via the SOA Reference Model.

Table 7. The Six approaches to SOA

Approach	Description (typical project owner characterization)	Qualifications
1. Business process driven	My business processes need to tap into resources, and each activity requires the invocation of IT functionality; I want that functionality to be available in a flexible, replaceable way.	Top-down
2. Tool-based MDA	I want to define a model (business model) and then let my tools generate the detail for me.	Top-down
3. Wrap legacy	I have existing systems I have been investing heavily on, but they are not resilient. I want new functionality added quickly, but these systems are partitioned. They are silos where functions are locked into them.	Bottom-up
4. Componentize legacy	Decompose the monolithic legacy systems into modules using compiler-based tools.	Bottom-up
5. Data-driven	Provide access to information using services without having to expose schemas or implementation decisions on the provider side. Encapsulate basic functions such as View, Edit Delete Add, Search .	Data-focused
6. Message-driven Integration	"Just want to have these systems integrate, communicate, over standard, non-proprietary protocols."	Service-Oriented Integration of Applications and Systems

20.1 Top-down Business driven

Strategic Business Goals, Models and Architecture are used to drive the need for underlying IT services that can be represented in an SOA.

These services are discovered and verified through the SOMA method as described by process decomposition, goal service modeling and variation-oriented analysis as described by SOMA [3].

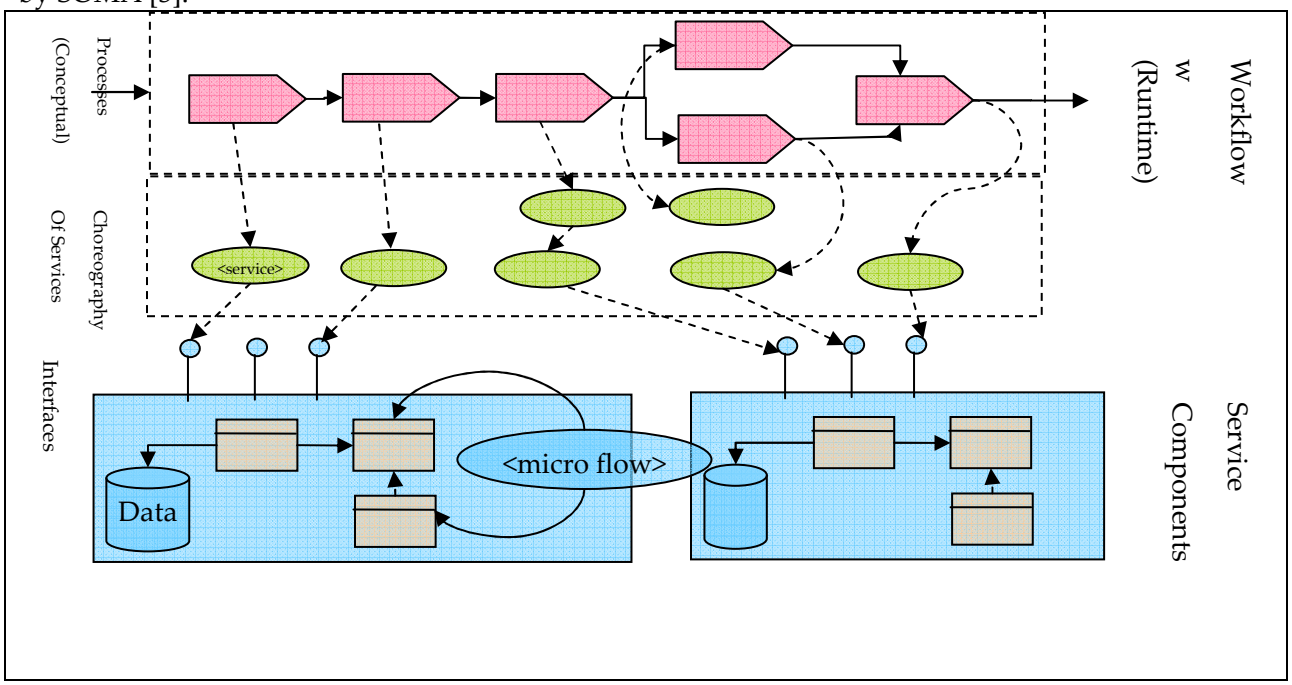

Figure 63. Top-down Business Driven

20.2 Top-down MDA

An example of supporting the top-down Model-driven Architecture process is depicted in this chart. Note that the Upfront Business Model can come from a CBM heatmap or component model within the scope of which we can conduct domain decomposition to arrive at the underlying processes, functional areas and variations.

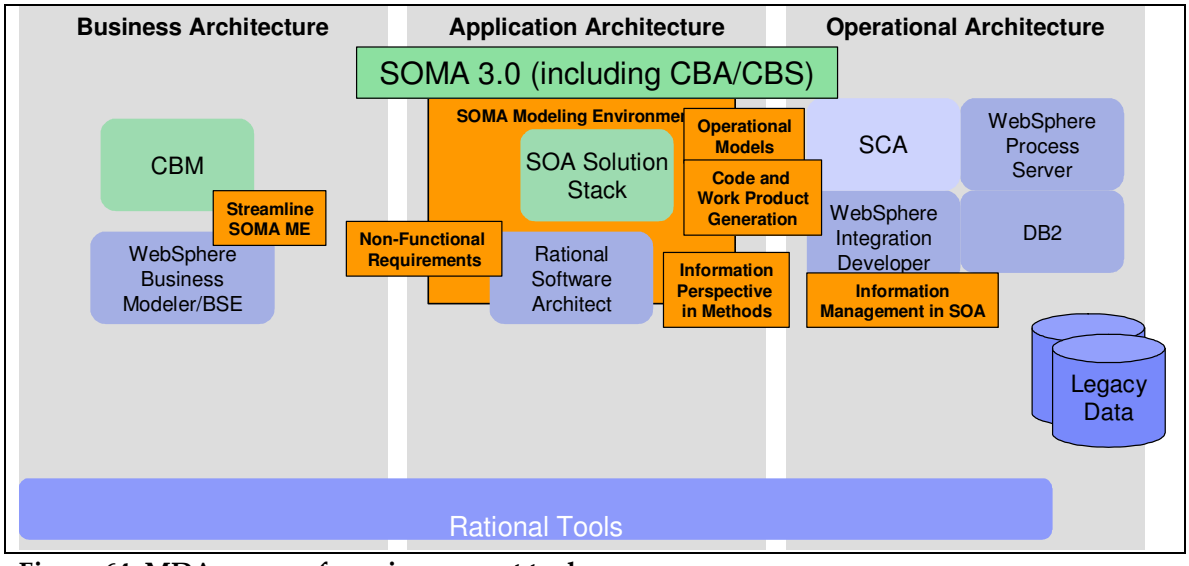

Figure 64. MDA process for using current tools

The key point is that in an MDA environment, the model of the platform independent representation is used to generate a good deal of the platform specific code with roundtrip capabilities for code-regeneration and for monitoring capabilities.

20.3 Bottom-up Legacy Transformation

Leverage legacy systems through intrusive code extraction tools such as Relativity Modernization Workbench. Expose the result as a service.

20.4 Bottom-up Legacy Integration and Consolidation

Provide an integration layer to tap into back end legacy or packages.

20.5 Integration via Messages and Event Driven Integration

Perform system integration using service-oriented integration techniques via messaging.

20.6 Data Driven

Encapsulate Data and provide a single point of access to data/

20.7 Other Scenarios

In this section we will explore several scenarios . **Note that the detailed explanation is being compiled!**

20.7.1 Insurance Case Study

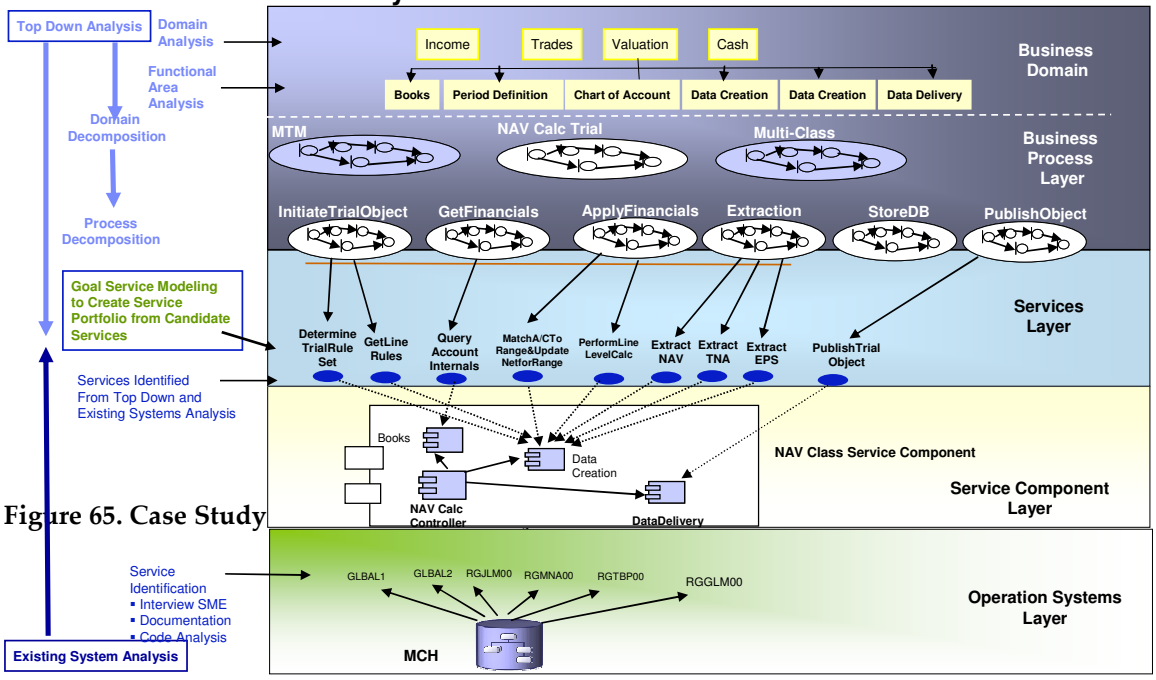

Figure 65. Case Study

20.7.2 Retail Lending

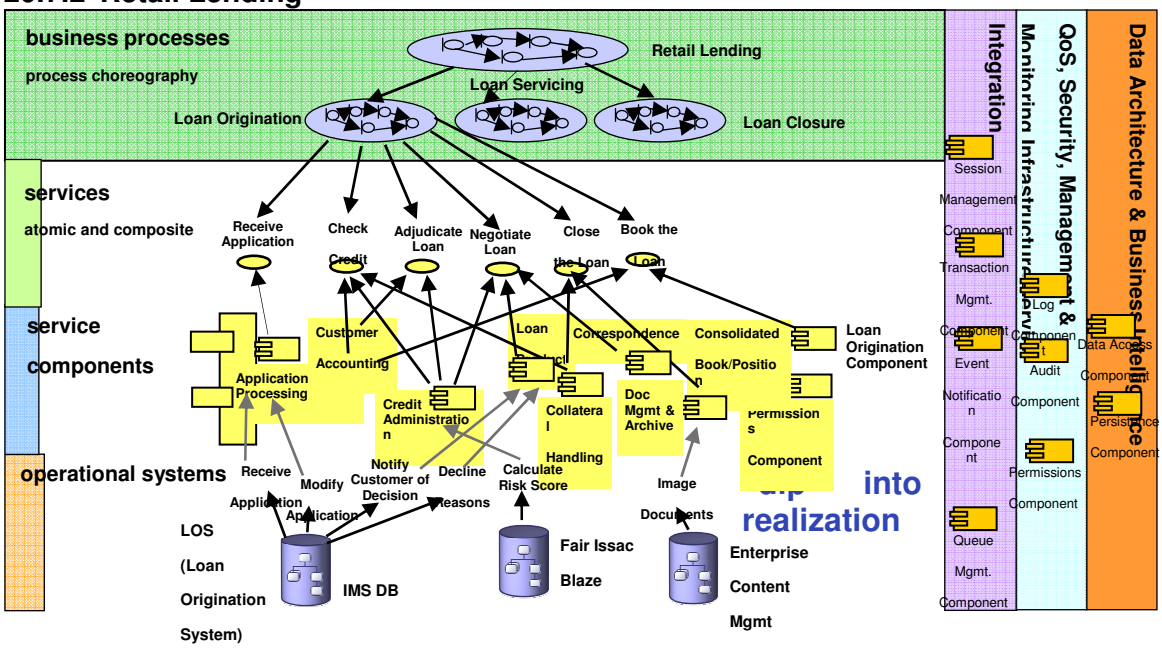

Figure 66. Case Study for Retail Lending

20.7.3 PA UCMS SOA Solution – A Sample Scenario

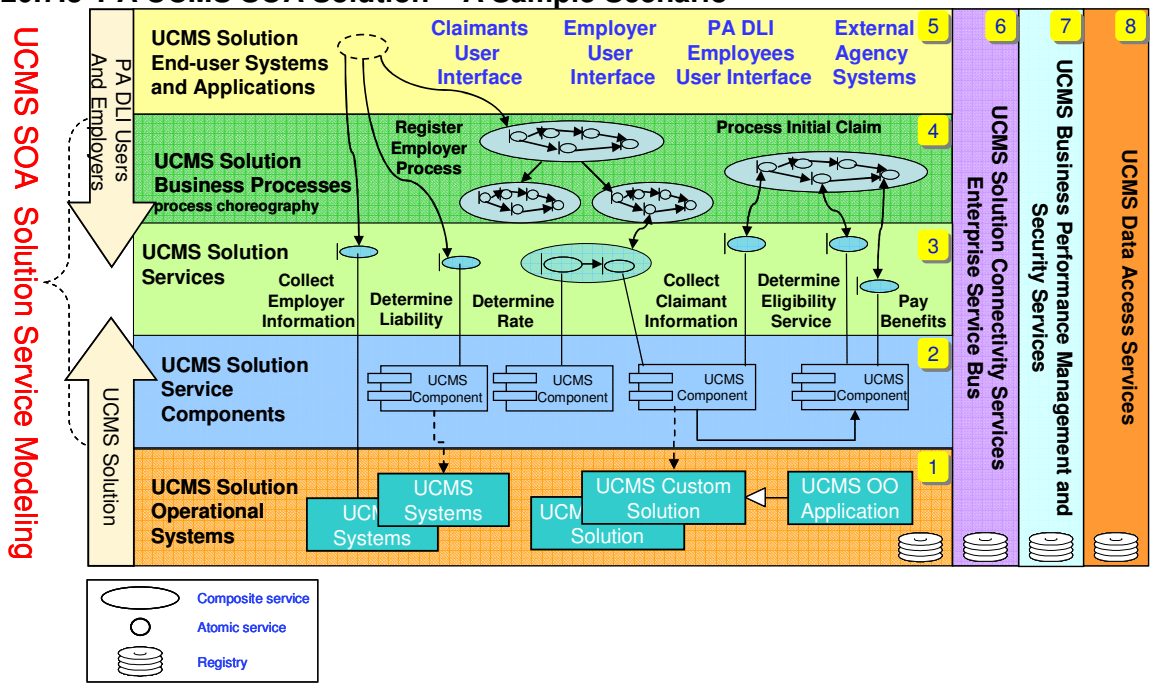

Figure 67. Case Study in PA UCMS

20.7.4 Large Electronics Manufacturer (customization)

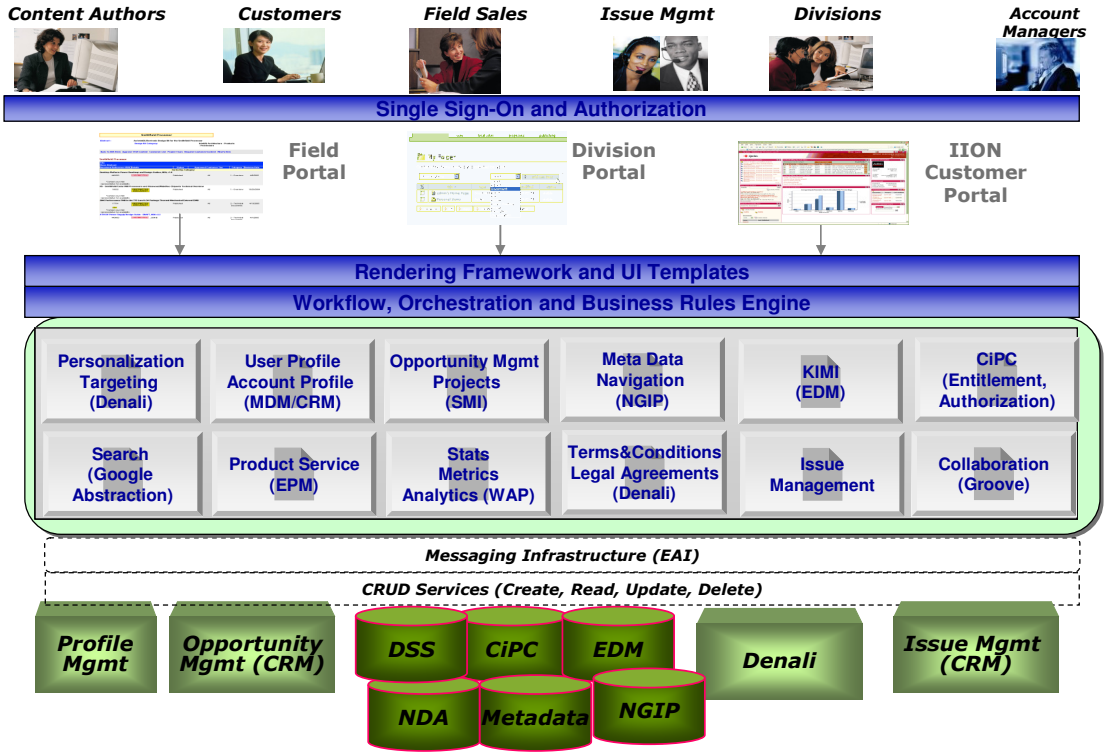

Figure 68. Case Study in Electronics Industry

20.8 A Rental Car Case Study

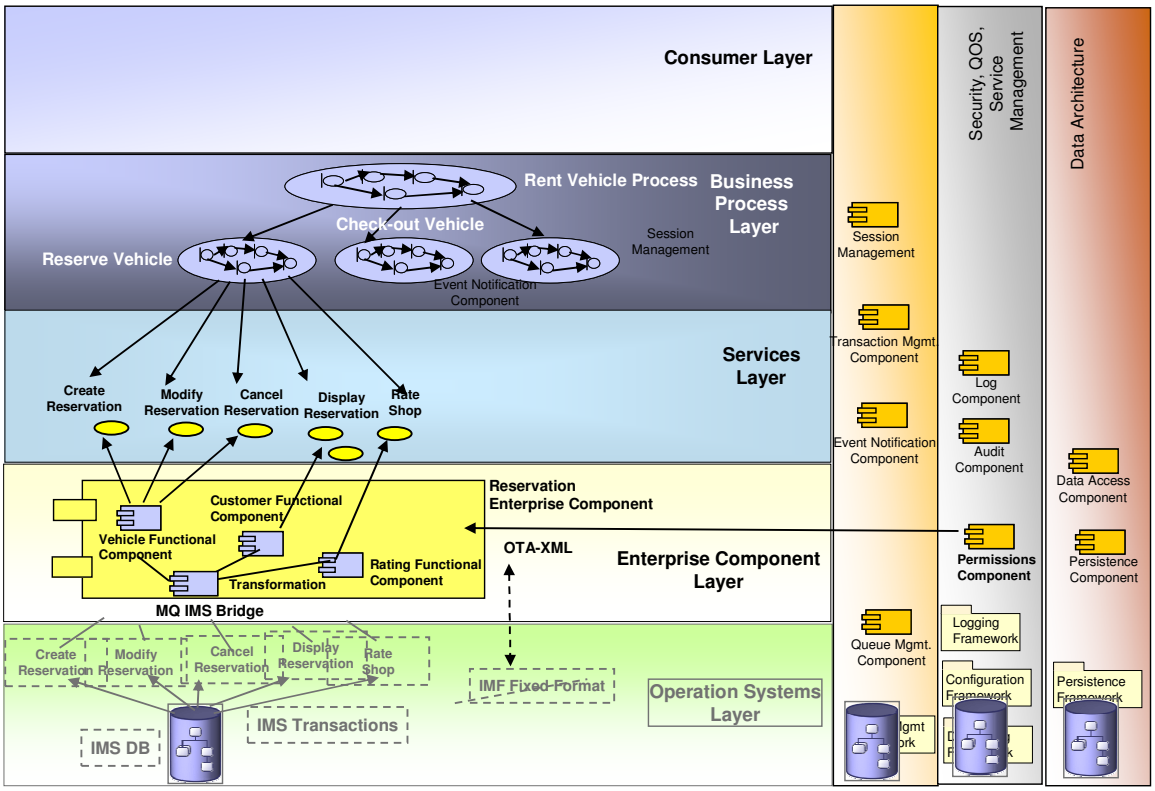

Figure 69. Case Study for Rental Car

21 Appendix D: Frequently Asked questions mapped to the layers

1. Who is responsible for converting the results (output message) of a service invocation to the format expected by the consumer?
    a. Should the provider have multiple services to support diverse consumer expectations? Or Should the responsibility of format conversion be that of the consumer?
2. Service Aggregation
    a. A service, say Get Customer Information will rely on several other services or components to construct the output message, in this example, Customer Information from multiple service sources. Clearly one service depends on multiple services.
3. How much business logic should reside in the ESB? Should the choreography be in the Bus or off the bus?
4. How should versioning be managed? Who handles versioning? The ESB, the consumer, the gateway?

22 Revision History

Version	Comments	Release Date
1.0	Initial Version of document as a PowerPoint set of slides	September 14, 2004
1.1	Ali, LJ, Abdul, Kishore, Michael	June 2005
1.23	Ali, LJ, Abdul, Min, Kishore, Michael	September 14, 2005
1.24	Prep for S3 F2F, AA placeholder additions.	October 5, 2005
1.24	Updates by Ali, LJ, Abdul, Michael, Min, Kishore, Bob	
1.25	Consolidated CommunityPaper for Team review	November 4, 2005
1.26	Review by Biffle French, Rolando Franco, Siddarth Purohit, Bertrand Portier; Finish Governance Section (Ali)	
1.27	F2F Walkthrough Week of End of Nov.	
1.28	Integration Nov 30, 2005 (The NY Integration)	Every section needs a table to describe the dependencies between layers. Elaborate if you need to in a text following the brief description in the table.
1.29	Ali Arsanjani , complete edit	Dec 5 2005
2.0	Liang-Jie (LJ) Zhang, and his ASRD SOMA-ME Team, complete edit	November 6, 2006

23 Notes

Layers 3&4: eliminate composite services, WSDL cannot be nested, services are black boxes, SCA explicitly couples dependencies to interfaces. S1=S2+S3+S4. Service component may have dependencies on other services.
Use of WSDL in a given layer – description of interface for one of the ABB in that layer.
The consumer layer is the visualization of the data that needs to be communicated between an actor and a business process. Consumer layer may be able to directly call any layer.
Create a section which describes the worst practice of connecting the portlet directly to the package.
Trends in telco, shared info data model, NGOSS, standard contracts defined as industry standards will gradually push vendors to comply. Information models encapsulate process?
Telco: NGOSS contracts to define interfaces, business process framework eTOM (similar to CBM) non overlapping functionally engineered components.
ACCORD messages imply process. Pushing us towards document exchaage pattern as opposed. End points intelligently process the services.
Composite applications and services to resolve.
Provider, Consumer:
How is composition by these roles?
Contract should not have anything to do

1. Separate interface from implementation
2. Contract between the provider and consumer lies in layer 3: via service description
3. Provider should have complete freedom to change or reconfigure the meta-data relating to the implementation of a service without the consumer knowing about it, without having any relation to the governance of the service specification identification itself.
    a. If you incorporate implementation related to meta-data in the service layer, then governance is impacted.
    b. SC: something that implements a service.

Scenario:
business processes
Raj:

1. Ordering App.. Bell Canada.. Could not map an application.

OrderMax, multi-channel take orders in diff channels in diff segments; portals/webbased used many services, customerProfile Services, appointment mgt services,
Did not have business process layer per se. No BPEL. Provisioning however, once order is captured there is a BPEL.

Business process is sometimes embedded in the screen flow in the portal. In an MVC sense, embedded in the controller.
2. Different business process part of portlet. Key concept around SOA: composite applications, implementation: portlets reusable. Composition of the applications. Configurator: crate a portlet, to reuse applications and in channels.