Guiding Principles for Modeling and Designing Reusable Services

Max Dolgicer
Managing Director
International Systems Group, Inc.
mdolgicer@isg-inc.com
http://www.isg-inc.com
Agenda

- The changing notion of applications
- The service layer model
- Stateful vs. stateless services
- How should services access data stores?
- Adapting a Service Oriented Software Development Lifecycle (SDLC) to service orientation
The Changing Notion Of “Applications”

- Applications used to have well-defined boundaries of functionality and code
- Service oriented applications are really a composition of services into business processes
  - It is more appropriate to speak about business processes than applications
  - One particular service could be part of multiple different business processes
  - This illustrates service reuse – a major benefit of SOA
    - Reduces Total Cost of Ownership (TCO)
    - Shortens time to market
    - Improves alignment of business and IT
      - Facilitates business agility
The goal of SOA (like any architecture) is to enable the development, integration and deployment of applications.
High-level Application Architecture

- How does SOA enable this application architecture?
  - Implementation of the architecture layers as collections of services
  - Standardized boundaries between layers
    - Service invocation protocols
      - SOAP, HTTP
    - Interface data formats
      - XML, JSON
The Service Layer Model

- Key architectural principles for the design of service oriented applications are:
  - Separation of concerns
  - Design for reusability

- Separation of concerns is achieved by a separation of functionality into layers
  - Breakdown into distinct layers facilitates decoupling of the services
    - One of the most important characteristics of a well-defined SOA
The Service Layer Model

- Reusability is achieved by designing key application services and infrastructure services that are autonomous and are agnostic of the business process context within which they are executed.
  - For example, the same services can be utilized by different presentation layers, for example:
    - Web interface (e.g. customer portal)
    - Thick client (e.g. Swing interface for internal users)
    - B2B channels (business partners)
A typical service layer model consists of:

- *Orchestration* Services layer
- *Application* Services layer
- *Infrastructure* Services layer
The Service Layer Model

Orchestration Services

Application Services

Infrastructure Services

Service 1
Service 2
Service 3
Service 4
Service 5
Service 6

Component 1

Legacy Systems

Legacy 1

Components
The Service Layer Model

- **Orchestration services**
  - An orchestration service acts as a controller
    - It composes application services and infrastructure services to implement a business process
  - For the most part, an orchestration service is made up of process flow logic and calls to lower level services
    - This allows orchestration services to be changed and adapted to new business process requirements without affecting the underlying application and infrastructure services
  - Orchestration services are light on code
    - Most of the code resides in application and infrastructure services
    - This allows to create new orchestration services or change existing ones fairly quickly
      - Better alignment of IT with business strategies
The Service Layer Model

- **Orchestration services**
  - Logic that is appropriate to put in an orchestration service:
    - Typically process flow logic that is specific to a particular business process
      - Cannot be reused in other processes, except as sub-process
    - Logic that needs to change often
    - Logic that is suitable to be implemented through graphical business process management or modeling tools
  - Logic that should not be put in an orchestration service:
    - Business logic that is business process agnostic and should be reused
      - Typically implemented in application services
    - Logic that requires conventional programming languages
The Service Layer Model

- **Orchestration services**
  - The orchestration services are typically the entry point for the presentation layer into the service oriented “application”

- In case of B2B scenarios, dedicated orchestration services usually form the partner facing business processes
Typical (RIA) Web application architecture

- JavaScript Object Notation (JSON) – alternative format to XML

User interaction

Out of band data exchange

Ajax

Browser

JSON

RESTful Web Service

Web Service Platform
JSON vs. XML

- The Ajax client gets data from a Web Service
  - Out of band relative to the user interaction with the Browser
- This data needs to be inserted into the HTML that the Browser displays
- XML is difficult to process within the Browser
  - Typically requires complex DOM programming
    - E.g. many method calls navigating through the nodes of the DOM data tree
  - Often introduces cross-browser compatibility issues
JSON vs. XML

- JSON as an alternative
  - Leverages the fact that JavaScript is available in all mainstream Browsers
    - And it is a popular client side development tool
  - JSON is a data format that is natural to JavaScript
  - Uses a textual data representation
    - On first glance similar to XML
    - However, much easier to process in JavaScript
The Service Layer Model

- **Application services**
  - Application services implement business logic
  - There are two implementation approaches:
    - They can be developed from scratch
    - They can be implemented as a wrapper for legacy systems and packaged applications
  - It is a best practice to implement wrapper services in the application services layer
  - The wrapper service performs a variety of functions:
    - Map the legacy API into a service oriented API
    - Implement security
    - Aggregate several legacy functions into a compound function
The Service Layer Model

- **Application services**
  - What Quality of Service (QoS) does the application service have to provide
    - Best effort, guaranteed, once-and-only-once delivery of messages
  - Application (and infrastructure) services often form chains of service invocation
    - Do we need transaction context propagation across services?
    - There are various options how to deal with security:
      - Delegation of credentials
      - Assigning the service as a “super user” of databases and ERP systems
      - Do we need end-to-end security, e.g. encryption only outside the firewall or close to the final application service or legacy system
The Service Layer Model

- **Application services**
  - Application services often integrate with back-end systems
  - This can raise a number of issues that need to be evaluated
    - Should the application service communicate with the back-end system synchronously or asynchronously?
      - What is feasible for legacy systems without requiring significant code changes?
      - How does the application service communicate with the services that invoke it – this may require a mapping
      - Should there be a push or pull relationship?
    - How much coupling should there be with a back-end system?
The Service Layer Model

- **Infrastructure services**
  - These services provide functionality that is technology specific
  - Examples include:
    - **Communication service**
      - Provides an abstraction to several communication mechanisms
        - HTTP, RMI, JMS, SOAP
    - **Notification service**
      - Facilitates sending a notification message to operations personnel, e.g. via email
    - **Security service**
      - Provides authentication, encryption and decryption
    - **Transformation service**
      - Transforms data between business partner formats, internal standard formats, and the format of legacy systems
The Service Layer Model

- **Infrastructure services**
  - Infrastructure services provide the highest degree of reusability
    - They are typically more agnostic of business processes than application services
  - Some infrastructure services are very generic in nature
    - For example, the notification service can be utilized by all business processes in the same way
  - Other infrastructure services require customization
    - E.g., the transformation service may be configurable to handle various data formats, but there are always new requirements that may necessitate custom coding
Degrees Of Service Orientation

- Wrong mindset:
  - Web Service = service oriented
  - Component = object oriented

- Better approach:
  - There are various degrees of service orientation
  - Determined by
    - Interface style
    - Service interaction metaphor
  - Web Services are not always the best answer
  - Choose the most efficient mechanism for particular requirements
  - A typical architecture will contain a mix of different mechanisms
# Degrees Of Service Orientation

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
<th>WSDL</th>
<th>Schema</th>
<th>XML</th>
<th>Binary</th>
<th>SOAP</th>
<th>HTTP</th>
<th>JMS/WCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lightly Coupled Objects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Asynchronous Services</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>REST-based Web Services</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Traditional Web Services</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Interface Style**
- **Interaction Metaphor**
When discussing state one has to distinguish between session state and server state

- Server state consists of persistent data that is kept on the server
  - In a database, ERP system, etc.
  - For example the balance of a bank account

- Session state describes the status of a session that a user (i.e. client application) maintains with a service
  - Keeps track of a multi-step conversation between the user and the service

For this discussion we are concerned about **session state**

- Services in a SOA **should** be stateless
Stateful vs. Stateless Services

- Stateful services maintain *session state* between service requests
  - Web-based services have an additional issue:
    ▶ Identifying the user/session such that it can be related to session state that is maintained on the server side

- Stateless services require that the client application passes sufficient conversational information with each request to the service

- Stateless services provide ultimate scalability
  - Each request is independent of previous requests and can be handled by any available service instance
  - Load balancing and failover are much easier to achieve
Stateful vs. Stateless Services

- **Server state** is not an issue when discussing stateless services
- However, server state obviously needs to be maintained
  - How do we achieve *persistence*?
  - Before SOA, we used object-relational mapping tools – is this still a valid approach?
How Should Services Access Data Stores?

- Typical application architecture, including a persistence service for managing data access
How Should Services Access Data Stores?

- The data access layer removes the responsibility of detailed data access code from the business applications.
- Generally, two approaches can be distinguished:
  - Specific **data access services** are created that are used by application services whenever they need to manipulate data.
  - Since services are usually implemented based on components, **object-relational mapping** tools can be employed to mediate between the database structures and the object model.
    - TopLink, Hibernate etc.
A Tale Of Three Data Models

- Services expose interfaces and are accessed though messages
  - Often based on WSDL
- Services are implemented through components
  - Components are based on an object model and rely on an object/relational mapping to interact with a relational database
- Databases employ a relational model
Data Access Services

- Data access services insulate databases from different applications
- Avoids the problems that occur when each application group writes their own data access code
  - Some may use SQL, others stored procedures, Hibernate, TopLink, JDBC, etc.
  - Unmanaged access often leads to freezing the database schema due to the dependencies introduced by multiple data access methods
Data Access Services

- Data access services ensure consistent, documented data access for all application development teams.
- All teams utilize the same data access services to implement their business services.
- No matter how many applications are using the data access services, it appears as one application to the database.
Building a successful SOA requires an appropriate development methodology

- Start from the business process
- Focus on the composition of process flow that orchestrates services into a business process
  - Take a process-centric approach to development
  - In contrast, object-oriented development is focused on component design and coding
  - Composition of processes can be accomplished through graphical tools rather than coding
Comparing Object Oriented Analysis & Design (OOAD) and Service Oriented Analysis & Design (SOAD):

- **OOAD** is focused at a level of granularity that is too small for SOA
  - Class level modeling doesn’t fit well with business service modeling
- **SOAD** must be predominantly process driven
- A service oriented development methodology must combine **SOAD and OOAD**, since services and components need to be **combined** for a complete solution implementation
Combining Services and Components

Orchestration Services

Application Services

Infrastructure Services

Components

Legacy Systems

Service 1

Service 2

Service 3

Service 4

Service 5

Service 6
## Component vs. Service Oriented Development

<table>
<thead>
<tr>
<th>Component-oriented development</th>
<th>Service-oriented development</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interface contract</strong></td>
<td><strong>Service contract</strong></td>
</tr>
<tr>
<td>➢ Object-oriented interfaces define a contract between objects</td>
<td>➢ Service contracts define a contract between service provider and service requestor</td>
</tr>
<tr>
<td>➢ E.g. Java interface, Interface Definition Language (IDL)</td>
<td>➢ E.g. Web Service Definition Language (WSDL)</td>
</tr>
<tr>
<td><strong>Object composition</strong></td>
<td><strong>Service orchestration</strong></td>
</tr>
<tr>
<td>➢ Objects are typically composed into a business process by implementing calls between objects on the code level, instead of through an abstracted layer</td>
<td>➢ Services can be orchestrated into business processes through a higher level, standards-based language (BPEL)</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td><strong>Services</strong></td>
</tr>
<tr>
<td>Methods can assume that they are being invoked with correct types</td>
<td>Services should assume that input messages can be incorrectly formatted</td>
</tr>
</tbody>
</table>
### Component vs. Service Oriented Development

<table>
<thead>
<tr>
<th>Object-oriented development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications cannot easily change because of the implicit relationships of the classes</td>
</tr>
<tr>
<td>- Object-oriented interfaces are maintained by tightly coupled groups of developers</td>
</tr>
<tr>
<td>Object reuse has never worked</td>
</tr>
<tr>
<td>- Developers need to be intimately familiar with the objects they want to use</td>
</tr>
<tr>
<td>- Inheritance, for example, breaks the autonomy of objects</td>
</tr>
<tr>
<td>The smallest deployment unit is a complete application, not an object</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service-oriented development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Services should be stable in the long term, while applications can change rapidly</td>
</tr>
<tr>
<td>- Impractical to communicate service changes to a wide audience of service consumers</td>
</tr>
<tr>
<td>- Service contracts need/have more flexibility than object-oriented interfaces</td>
</tr>
<tr>
<td>Services should be autonomous</td>
</tr>
<tr>
<td>- Requires minimal knowledge about the service before it can be used within an application</td>
</tr>
<tr>
<td>Services can be deployed as an atomic unit, before any application has been conceived</td>
</tr>
</tbody>
</table>
**SDLC Activities**

- The major software development project activities are the same as in traditional projects
  - Example: RUP methodology

<table>
<thead>
<tr>
<th>Activities</th>
<th>Phases</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inception</td>
</tr>
<tr>
<td>Business Modeling</td>
<td>Initial</td>
</tr>
<tr>
<td>Requirements</td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; Design</td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
</tr>
<tr>
<td>Deployment</td>
<td></td>
</tr>
<tr>
<td>Change Management</td>
<td></td>
</tr>
<tr>
<td>Project Management</td>
<td></td>
</tr>
<tr>
<td>Environment</td>
<td></td>
</tr>
</tbody>
</table>

**Chart:**

- Phases: Inception, Elaboration, Construction, Transition
- Iterations: Initial, E1, E2, C1, C2, CN, T1, T2
However, each activity needs to be adapted to support a service oriented project

- Business modeling
- Requirements
- Analysis and design
- Implementation
- Test
- Deployment

Adapting SDLC Activities to SOA
Business Modeling Activities

- Start from the business process
- Focus on the composition of process flow that orchestrates services into a business process
  - Take a process-centric approach to development
  - In contrast, object-oriented development is focused on component design and coding
- **BUT:** most organizations are structured around their business functions
  - Business processes span functions
- Starting with process based modeling could produce process based “silos” of services
  - Another, initial task is required
Business Modeling Activities

- Aligning business strategy to IT has often failed
  - Due to lack of traceability from business to IT architecture and implementation
- A successful SOA must start with an architectural representation that the business sponsors can understand
- Next, create an audit-trail from business strategy through IT architecture to project implementations
Business Modeling Activities

- Identify a business function
  - E.g. inventory management
- Identify the owner of the business function
  - His/her role and responsibilities
  - His/her priorities
    - E.g. minimize inventory (just in time production)
- Identify actors that relate to the business function
  - E.g. supplier, customer
  - Identify their priorities
- Identify tasks that the business function performs
  - E.g. shipping, re-stocking
  - Identify pre- and post-conditions for each task
Since a major focus of SOA is reusability, the requirements activities need to account for a wider scope:

- Cross-project or enterprise wide
- Need to bring a variety of different stakeholders to the table
  - Business owners, project managers, IT managers, etc.
- Set the project scope
  - This is more important in SOA, since addressing the immediate business requirement (e.g. of one particular line of business) has to be balanced against building services that are of enterprise value for later reuse
- Business process agnostic requirements need to be determined
  - This is important for the services that will be designed as application or infrastructure services
Analysis & Design Activities

- High-level service modeling should be performed
  - Business process focus
  - Use business models to guide the service modeling process
    - Opportunity to improve alignment of business and IT
- Develop layered services model
  - Classify services
    - Orchestration (business) services
    - Application services
    - Infrastructure services
  - Separation of candidate services into the three layers
    - Group required functionality into application and infrastructure services
Analysis & Design Activities

- Define high level service contracts
  - Functional decomposition
    - Derive service interfaces
  - Service Level Agreements (SLAs)
- Create an inventory of existing services that can be reused for the project
  - This could include services outside the enterprise (B2B)
- The potential for encapsulating legacy systems as services needs to be analyzed
New services can be provisioned by mining existing systems
- Encapsulate legacy applications
  - Build wrappers that provide a service-based interface to access existing applications
- Restructure legacy applications
  - Break legacy applications apart to achieve modularization that is aligned with the service model

Both approaches require to cope with impedance mismatch
- Synchronous service invocation ↔ asynchronous application processing
- Stateless service ↔ stateful applications
When existing business logic and data stores are reused to create services, logic and data often reside in multiple places.

Requires enterprise wide analysis to determine:
- What system would be best suited to provision the service such that duplicate systems could be retired in the long term.
- Which system is considered the “master”
  - I.e. the authoritative source of business rules or up to date data.
- What kind of rationalization could be introduced via the service wrapper
  - E.g. to transform legacy data into a new enterprise standard representation.
Analysis & Design Activities

- Identify shared services
  - Three types of shared services
    - **Identical services**: different business areas or business processes can use the identical service
      - Or multiple instances of the same service
    - **Operationally distinct shared services**: different business areas or business processes can use the identical service in terms of its logic, but require operational separation
      - Can be achieved through configuration
      - For example a service that supports internal or external (i.e. partner) users
    - **Common base services**: different business areas or business processes can use a similar implementation, but require customization
      - Common component code base
Analysis & Design Activities

- Since a major focus of SOA is adherence to standards, the design phase needs to follow the standards that have been established by the overall enterprise architecture
  - Define data entities in XML and Schema
  - Are WSDL and SOAP required?
  - What is the appropriate degree of service orientation for each particular service?
- How will the requirements for synchronous and asynchronous service interaction be accommodated?
Analysis & Design Activities

- Services typically communicate via XML based messages
  - Messages need to be defined as part of the analysis and design activities
  - They represent the interface of a service
  - As such they can be compared to a class interface, BUT:
    - Messages have to be designed in a context that goes beyond one service
    - Messages should be decomposed into a hierarchy of reusable message component layers
Test Activities

- Since a major focus of SOA is reusability, the testing phase needs to account for usage scenarios that go beyond the requirements of the current project
  - Different types of clients, exception scenarios, interoperability requirements, etc.
  - Not all service requestors may be known to the service development team. Therefore, particular focus must be given to test service versioning.
- Greater emphasis on performance and scalability tests
  - XML based interfaces can be verbose
Top-down vs. Bottom-up Approach

- Should a SOA-based project be approached with a top-down or bottom-up strategy?
  - Determines which SDLC activities might be cut short and others that might deserve a greater emphasis
  - There is a significant difference between component based and service oriented projects
    - Service oriented projects (should) focus on business processes (orchestration)
    - They should **not** focus on the codification of business functions
Top-down vs. Bottom-up Approach

● Top-down approach
  ◆ Start with a business process analysis
    ▲ Includes business process, business functions, business owners and their roles and responsibilities
    ▲ Focus on alignment of SOA with business models
    ▲ The definition of new services (or reuse of existing ones) that will provision the required functionality of the business process follows later
  ◆ Align the service interfaces (Schemas) with an enterprise data model
    ▲ Avoid that service designs come up with their own data model
    ▲ Ensures that service semantics are aligned with enterprise standards
    ▲ Simplifies data transformation requirements in the long run
Top-down vs. Bottom-up Approach

- Bottom-up approach
  - Focus on application centric requirements
    - .. of one particular business process or one project
  - Often employed in an application integration context
    - Building wrapper services for legacy systems
    - Using auto-generation of service wrappers for recently developed component based applications.
  - Typically starts with the service design
  - Often propagates legacy structures into a flawed SOA
  - Focused on fulfilling particular requirements of one (narrow) project
    - Within aggressive time and cost constraints
# Top-down vs. Bottom-up Approach

<table>
<thead>
<tr>
<th><strong>Top down</strong></th>
<th><strong>Bottom up</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pro</strong></td>
<td></td>
</tr>
<tr>
<td>Achieves business and IT alignment.</td>
<td></td>
</tr>
<tr>
<td>More likely to create a SOA that is not constrained by legacy architectures.</td>
<td></td>
</tr>
<tr>
<td>Enables standardized, repeatable, enterprise scale integration solutions.</td>
<td></td>
</tr>
<tr>
<td>Can align services with the enterprise data architecture.</td>
<td></td>
</tr>
<tr>
<td>Ensures that key goals are met in the long term (development and operational efficiency, business efficiency).</td>
<td></td>
</tr>
<tr>
<td><strong>Con</strong></td>
<td></td>
</tr>
<tr>
<td>High up-front cost.</td>
<td></td>
</tr>
<tr>
<td>Lengthy timelines could conflict with business demands.</td>
<td></td>
</tr>
<tr>
<td>Often difficult to obtain required information (e.g. business models).</td>
<td></td>
</tr>
<tr>
<td><strong>Pro</strong></td>
<td>Cheaper and faster, since there is less upfront analysis and design effort.</td>
</tr>
<tr>
<td>IT can demonstrate quick, albeit perceived, success.</td>
<td></td>
</tr>
<tr>
<td><strong>Con</strong></td>
<td>Achieves only limited reuse; creates service silos.</td>
</tr>
<tr>
<td>No adherence to enterprise data models.</td>
<td></td>
</tr>
<tr>
<td>No strategic alignment between business and IT.</td>
<td></td>
</tr>
<tr>
<td>Re-creates existing architectures that are not truly SOA.</td>
<td></td>
</tr>
<tr>
<td>Proliferates more one-off integration points.</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- Adhere to the key architecture pattern for service oriented applications – *the service layer model*
  - Provides classification of services
  - Facilitates for a clean architecture through separation of concerns
  - Optimizes reusability
- Focus on *loose coupling*
  - On the architecture level as well as the service design level
Conclusions

- Enterprise-level requirements should drive the standardization of infrastructure and application services
  - Building an *enterprise-class infrastructure and application services framework* is key to achieving many of the benefits that motivate SOA