IBM SOMA Method
and
OMG SoaML

Jim Amsden
SoaML Goals

- **Intuitive and complete** support for modeling services in UML
- Support for **bi-directional asynchronous services** between multiple parties
- Support for **Services Architectures** where parties provide and use multiple services.
- Support for **services defined to contain other services**
- Easily mapped to and made **part of a business process specification**
- **Compatibility with UML, BPDM and BPMN** for business processes
- Direct mapping to web services
- **Top-down, bottom up or meet-in-the-middle modeling**
- **Design by contract** or **dynamic adaptation** of services
- To specify and relate the **service capability and its contract**
- **Minimal changes to UML** (just one)
SOA Maturity: The characteristics, structure and behavior of parts of the organization that are undergoing transformation and are undergoing adoption – depicted using the Service Integration Maturity Model (SIMM)
The Service Integration Maturity Model (SIMM) helps define a roadmap for incremental IT transformation linked to business transformation.
SOA involves different aspects for different roles...

“There are different views of an SOA… all requiring design and development”

- **Business**
  - A set of value propositions delivered through capabilities that a business wants to expose/provide to their customers and partners, or other portions of the organization

- **Architecture**
  - An architectural style which relates service providers and consumers through a set of service descriptions (interfaces) [and policies]
  - A set of architectural principles to be used in this relation, patterns, principles and criteria which address characteristics such as modularity, encapsulation, loose coupling, separation of concerns, reuse, composability, and single implementation

- **Implementation**
  - A programming model complete with patterns, standards, tools, and technologies such as Web Services

SOA is Different Things to Different People….
Business Focused SOA Using Model Driven Architecture

MDA Terms

- Business Concerns
  - Business Model
  - Enterprise Services (e-SOA)
  - Roles, Collaborations & Interactions
  - Process & Information

- Logical System Model
  - Technology Services (t-SOA)
  - Components
  - Interfaces, Messages & Data

- Technology Specification
  - JMS, JEE, Web Services
  - WSDL, BPEL, XML Schema

Line-Of-Sight

Refinement & Automation

Computation

Independent Model

Platform Independent Model

Platform Specific Model

SoaML Specification – Revised UPMS Submission
SOMA is IBM’s End-to-End Method for Building Enterprise Scale Applications and Solutions based on Service-Oriented Principles

1. The fractal phases will contain capabilities that can be leveraged as needed in different sequences
2. Realization, for example is leveraged in all phases
3. No rigid sequencing is implied here; there is a separate diagram showing a typical sequence
4. In a fractal model, phases will consist of capabilities that may be used by other phases

Regardless of whether the implementation will use web services ...
Services, Service Participants and Service Channels Are SOA Modeling Constructs

Business Processes – represent the flows of activities required to complete a business process. They are compositions of services targeted to achieve business goals.

Service Interfaces – the main structuring element defining what is required and provided by participants. Defines functionality, protocol and quality of service, all of which are externalized within service descriptions/policy – separate from implementation and use.

Participants – that realize not only the functionality of the services they expose but also ensure their quality of service (the QoS advertised by the Service provider implementing (“realizing”) the services.

SOMA was created to specifically address modeling (analysis, identification, specification, realization, implementation) of all three constructs and associated Information, Composition, Rules & Policies, Context, Events.
The Role of SOMA in SOA Development Is to Provide a Prescriptive Method Necessary to Create an SOA

SOMA is all about the identification, specification, realization, implementation, and deployment of services, components, and flows based on TOG SOA Reference Architecture.
The Soma Lifecycle is Iterative and Incremental

- **Identification**: of candidate services and flows, existing assets
- **Specification**: of services to be exposed, flows, and components (for realization of services)
- **Realization**: Capture realization decisions (concurrently with first two phases), explore feasibility of realization scenarios, instantiate SOA Reference Architecture
- **Implementation**: Implement service components and services
- **Deployment**: Package and provision

**How we do it?**

- **Identification**
  - Selection of Solution Templates, Method Adoption
  - Domain Decomposition
  - Goal-Service Modeling
  - Existing Asset Analysis
  - Service Flow Specification
  - Subsystem Analysis
  - Component Specification
  - Message & Event Specification
  - Component Flow Specification

**Arrows signify incremental iteration**

**What we do?**

- **Realization Decisions**
  - Solution Template & Pattern Selection and Instantiation
  - Technical Feasibility Exploration
  - Service Flow Specification
  - Service Specification
  - Message & Event Specification

- **Implementation**
  - Construction
  - Generation
  - Assembly Integration
  - Integration Testing
  - User Acceptance Testing
  - Deployment (Packaging/Provisioning)

- **Deployment**
  - Build/Assembly
  - Testing

- **Close**
  - Monitoring & Management
  - Governance
Focus on the business requirements and service identification

**Business Concerns**

- Business Model
  - Business Services (e-SOA)
  - Roles, Collaborations & Interactions
  - Process & Information

- Logical System Model
  - Technology Services (t-SOA), Components
  - Interfaces, Messages & Data

- Technology Specification
  - JEE, JMS, Web Services
  - WSDL, BPEL, XML Schema
SOMA Identification is a method for identifying business relevant services that maximize use of existing assets

- Goal-Service Modeling
- Domain Decomposition
  (Top Down Analysis)
  - Process Decomposition
  - Functional Area Analysis
  - Information Analysis, Modeling, and Planning
  - Rule and Policy Analysis
  - Variation Oriented Analysis

- Existing Asset Analysis
  (Bottom up Analysis)

- Additionally, Service Refactoring and Rationalization
  - Service Litmus Tests
  - Exposure Decisions including Exposure Scope
Apply Complementary Techniques to Identify Services, Flows, and Components, Information and Rules and Policies

- **Domain Modeling & Decomposition (Top-down Analysis)**
  - Process Decomposition
  - Functional Area Analysis
  - Information Analysis
  - Rules and Policy Analysis
  - Variation-Oriented Analysis (Processes, Structure (data/semantics), and Rules)

- **Goal-Service Modeling**
  - Services fulfilling subgoals
  - Services supporting KPIs

- **Existing Asset Analysis (Bottom-up Analysis)**
  - API Analysis
  - Industry Models and Standards
  - Existing System Interfaces

Examples:
- **e.g., Process Decomposition:** Decompose processes and model the ones that are within scope
- **e.g., Look at API’s transactions, existing system interfaces and choose the relevant ones**
Goal-Service modeling with Business Motivation Model and SoaML

- The business motivation and strategy describes what the business is attempting to do, how and why.
- The services model indicates what capabilities are needed to realize the business motivation.
- Business motivational elements can be realized by:
  - Capabilities
  - Business processes
  - Services architectures
  - Service contracts
  - Service interfaces
  - Use cases
  - etc.
Requirements can also be captured in requirements management tools.
Use Goal-Services modeling to identify the capabilities necessary to realize business goals

- Capabilities define the ability of an organization or entity to effect change that meets its objectives.
- Capabilities describe what the business needs to realize its goals or has to provide its value propositions.
- Capability operations can provide additional detail identifying specific functions that are needed.
- Capabilities can indicate what capabilities they depend on.
Or use processes decomposition to identify needed capabilities

- Business analysts provide process models that describe and verify their business operational requirements
- Domain decomposition can be used to identify additional requirements and capabilities
Capabilities can also be decomposed and organized into functional areas

- Capabilities can have behaviors that identify the capabilities they need
- Processes of a capability can indicate how the capability operations might be provided
- The actions in the processes can use the techniques of domain decomposition to identify other needed capabilities
Capabilities can be discovered or realized by analyzing existing assets

- The Risk Assessment capability has been identified as realized by an existing EJB session bean accessed from RAM
  - The capability could be identified from the existing session bean or
  - The capability could have been defined previously and through an asset search, the potentially realizing session bean could be discovered

- Capabilities decouple requirements from existing assets making it easier to envision system requirements and then go back and search for applicable assets
Service Litmus Tests (SLTs) are gating criteria used to determine what capabilities should be exposed by service interfaces.

**Candidate Services**

- Capability
- Capability
- Capability
- Capability

**Exposed Service Interfaces**

Apply SLTs (Universal and Custom)

Apply priority and weight and calculate service rating for each litmus test

Make exposure decisions

Determine Exposure Scope

**Exposure Scope**

1. Department/Division
2. Line-of-Business
3. Enterprise
4. Eco-system

**Service Litmus Tests**

**Universal (out-of-the-box)**

1. Business Alignment
2. Composability
3. Externalized Service Description
4. Redundancy Elimination/Reusability
5. Feasibility of Implementation
6. Business Entity based Services (for Information Services only)

**Customized**

7. Client/Project Defined SLTs
Capabilities can be exposed by service interfaces or realized by participants

- Capabilities can be used to identify candidate services that should be exposed through service interfaces or realized by participants.
- The SOMA Service Litmus Test can be used to determine which capabilities should be exposed as services.
Services architectures can specify the participants and interactions needed to realize the business motivation.

A ServicesArchitecture (or SOA) is a network of participant roles providing and consuming services to fulfill a purpose. The services architecture defines the requirements for participants and their interactions through service contracts.

The services architecture puts a set of services in context and shows how participants work together for a community or organization without required process management.
Service contracts may be used to formalize interactions between participants

- The service contract specifies the roles played by the interacting participants
- The role types are service interfaces describing the responsibilities of the participants
- Behaviors owned by the service interface describe the protocol for providing and using the service operations
- Service interfaces can be derived from or fulfill service contracts
Producing the logical systems model addressing and realizing the business concerns

Business Concerns

Business Model
Business Services (b-SOA)
Roles, Collaborations & Interactions
Process & Information

Logical System Model
Technology Services (t-SOA), Components
Interfaces, Messages & Data

Technology Specification
Web Services
WSDL, BPEL, XML Schema
SOMA Specification Uses Comprehensive Techniques to Specify Services, Flows, and Service Components that Realize Services

- **Information Specification**
  - Data Model, Message Model, Business Glossary

- **Existing Asset Analysis – Fine Grained**
  - Determine the technical viability of existing applications and approaches to realize services

- **Service Specification**
  - Elaborates the **Service Model**, for example, service dependencies, service composition and flow, rules and policies, event specification, service operation, service message specification, QoS requirements, design decisions, and so on

- **Subsystem Analysis**
  - Partitions subsystems into service components that will be responsible for service realization

- **Component Specification**
  - Details component modeling, flow, information architecture, messages
Service

- Service is as work of value provided by one to another

- The access to the service is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service contract. A service is provided by a participant acting as the provider of the service—for use by others. The eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider. [OASIS RM]
Services interfaces are defined for exposed capabilities

- A ServiceInterface defines the interface between participant components for a particular service.
- It defines the provided and required interfaces and operations.
- And it specifies any protocols for using and implementing those operations.
- It may also define QoS policies and link to goals and value propositions in the business motivation model.
- A ServiceInterface can fulfill ServiceContracts or realize use cases.
Service interfaces can also be very simple or represent simple callback functions

- The Purchasing service interface is a simple UML Interface
  - It has no required interface or protocol
- The ShippingService requires a simple callback
- Scheduling is also a simple service interface
Domain data models represent the persistent information used to implement the services.
Service data is an abstraction of this information exchanged between participants

- Service data is usually a selection and projection of domain data for a particular service interchange
- Value objects or data transfer objects are used to reduce coupling
- Both document-centered and RPC style parameters are supported
- MessageType parameters are used for document-centered style
  - Operation can have at most one input and one output
Participants provide and consume services

- Participants represent logical or real people, systems or organizational units that participate in services architectures and/or business processes.
- In SoaML participants provide and use services, defining their external interface.

- Participant
- Service port – point where the OrderProcessor offers the purchasing service
- Request port – point where the OrderProcessor uses the shipping service
Participants may have internal structure for their state and behaviors

- Attributes represent the participant state and are shared context for the behaviors
- Behaviors are the methods for providing and using service operations
- These behaviors use service operations available through request points
- Every provided service operation must have a method, but this can be deferred to implementation
Participant methods show how services are *provided* and *used*

- Participant methods can be specified in a number of ways:
  - Activity
  - Interaction
  - StateMachine
  - (soon) BPMN process

Activity partitions represent the request points. Operation invocations are made through these ports.

This is the activity shown on the previous chart expanded to show the details.
Participants may also be assemblies of references to other Participants

- Participants may be recursively assembled into service value chains where the needs of one participant are satisfied by the compatible capabilities of another.
An important aspect of an SOA is minimizing coupling

- Avoid having components realize and use interfaces directly
  - SoaML requires all services to be provided and used through service and request points

- Avoid direct dependencies between components
  - SoaML uses service channel connections between request and service points in participant assemblies at the instance level to isolate dependencies

- Avoid dependencies between component ports
  - This is less coupling than the previous item, but still constrains all usages of the participants to a particular usage pattern

- Avoid subsystem components that contain other components
  - SoaML creates participants in packages and assembles references to them in the internal structure of other participants
  - This separates component definition, assembly and deployment

- Separate component specification from realization
Components don’t realize and use services directly

- Component dependencies result in connections between components instead of between ports.
- Any change in how the component interacts with one of its service providers requires all other interactions to become suspect.
- Ports decouple consumers and providers by focusing dependencies on specific service interactions instead of the component as a whole.
Connect usages of components, not the components themselves

- Creating usage dependencies between the participants results in increased coupling between a specific set of components.

- Rather each participant expresses its needs and capabilities through request and service points. Then instances of participants are connected in various participant assemblies which provide the context in which they are reused.

- The same Participants can be used to define parts in different assemblies for different purposes.

- This is the classic class vs. “representative instances” issue where class diagrams are used as a “context” to define the relationships between a set of representative instances.
Use connectors not dependency wires

- Dependency wiring between the request and service points reduces the coupling between components, but still results in coupling between specific participants.
- The OrderProcessor participant does need to depend specifically on the Invoicer, Productions and ShipperImpl participants.
- Rather OrderProcessor only needs to specify its request points. Any participants with compatible services can be used to fulfill the OrderProcessor's needs in the specific context in which the participants are being used.
- Connections are created at the instance level with service channel connectors, not at the class level.
- There may be some circumstances where particular participants are required for all usages of a participant. Dependency wires would be useful in this case.
Assemble participants in other participants, not subsystems

- The Manufacture subsystem contains the participants which implies they cannot be used in other subsystems.

Rather each participant is defined separately and instances of them are assembled as parts in the context of some other participant that is connecting them together for a particular purpose.

- Assembly is done at the instance level, not the class level in order to maximize reuse.

- Participant assembly and participant deployment are separate concerns.

- Participant deployments are done as a separate step, mapping participants to logical deployment topologies using RSA deployment modeling capabilities.
Coupling can optionally be further reduced by separating specification from implementation

- A specification component is like an Interface, but can have ports, owned behaviors, etc. to fully specify what realizing participants must do.
- Participant assemblies can then depend on the specification components instead of a particular implementation.
- Any realizing participant can be substituted at development, deployment or run-time.
Technology Architecture determines how the specified service will be implemented

**Business Concerns**

- **Business Model**
  - Business Services (b-SOA)
  - Roles, Collaborations & Interactions
  - Process & Information

- **Logical System Model**
  - Technology Services (t-SOA), Components
  - Interfaces, Messages & Data

- **Technology Specification**
  - JEE, JMS, Web Services
  - WSDL, BPEL, XML Schema
SOMA Realization Includes Solution Template Selection and Instantiation, Technical Feasibility Exploration, SOA Solution Stack Instantiation

- Select and instantiate solution templates and patterns
- Technical feasibility exploration
  - Examine approaches to handle client requirements
  - Examine legacy application specific considerations
- Detail SOA Solution Stack
- Realization Decisions
  - Consider alternatives
  - Select the alternative
  - Provide justification
Technical architectures can be captured in recurring patterns and transforms and applied to models to create the initial implementation.
Example Web Services Generation

```xml
<wSDL:portType name="Purchasing">
  <wSDL:operation name="processPurchaseOrder">
    <wSDL:input message="tns:processPurchaseOrderRequest"/>
    <wSDL:output message="tns:processPurchaseOrderResponse"/>
  </wSDL:operation>
</wSDL:portType>

<wSDL:portType name="Scheduling">
  <wSDL:operation name="requestProductionScheduling">
    <wSDL:input message="tns:requestProductionSchedulingRequest"/>
  </wSDL:operation>
  <wSDL:operation name="sendShippingSchedule">
    <wSDL:input message="tns:sendShippingScheduleRequest"/>
  </wSDL:operation>
</wSDL:portType>
```
Agent and Milestones

- **Agent**
  - autonomous entity
  - has its own lifecycle behavior
  - can adapt to the environment
    - also through modification of its definition

- **Milestone**
  - defines a value of progress
  - attached to behavioral elements
  - is used especially for dynamic analysis of behavior that does not necessarily end
A Milestone is a means for depicting progress in behaviors in order to analyze liveness. Milestones are particularly useful for behaviors that are long lasting or even infinite. A Milestone can be understood as a “mythical” Signal. A mythical Signal is a conceptual signal that is sent from the behavior every time a point connected to the Milestone is passed during execution. The signal is sent to a conceptual observer outside the system that is able to record the origin of the signal, the signal itself and its progress value.
Topics to Explore

- **Encapsulation**: Description of a encapsulating element or “component” including its potential interactions with other prototypical components

- **Contract**: Specification of the potential interactions between components that the agree to adhere to

- **Behavior**: Representation of a component’s internal behavioral implementation (orchestration)
  - How the component consumes and provides services according to the agreed upon contract

- **Structure**: Internal structure of a component as an assembly of other components
  - How the components are connected in order for the interactions to occur
  - At the class level and/or part level
The description of a component’s interaction with other components

- Interactions between components involve the following:
  - The specification of the messages exchanged
  - The grouping of those messages into logical, cohesive units
  - The valid sequence, protocol or choreography that constrains the order of those messages

- We need to look at how BPMN and SoaML address each of these dimensions
A ServicesArchitecture defines the specification for the interaction between a set of participants collaborating to accomplish some end.

The behavior of the ServicesArchitecture specifies how the participants interact.

References to ServiceContracts specify the agreements between the participants, specifying how they interact in this services architecture.
Participant Providing and Consuming Services

A Participant provides services through ServicPoints and consumes them through RequestPoints.

A service port represents a capability the participant exposes as a service.

A request port represents a need the participant expresses as a request for a service.

Interfaces define the available operations that do things.

Participants must provide a method implementing each of their provided service operations.

A Service Port is typed by a ServiceInterface and describes the expected interactions the participant has through this interaction point.

Participant Providing and Consuming Services

- **Purchasing**
  - `+ processPurchaseOrder (customerInfo: Customer, purchaseOrder: PurchaseOrder) : Invoice`

- **InvoiceProcessing**
  - `+ processInvoice (invoice: Invoice)`

- **OrderProcessor**

- **ScheduleProcessing**
  - `+ processSchedule (schedule: Schedule)`

- **Request**
  - invoicing: ~InvoicingService
  - scheduling: ~Scheduling
  - shipping: ~ShippingService

- **Service**
  - purchasing: Purchasing
  - ScheduleProcessing

- **Participant**
  - OrderProcessor

- **Interface**
  - Purchasing
  - InvoiceProcessing
  - ScheduleProcessing
ServiceInterface – the definition of a service

SoaML ServiceInterface defines the valid messages, message grouping and choreography and is used to define the type of Service and Request Ports.
The processPurchaseOrder Service Operation Design

This activity is owned by the participant and is the method for providing its service operation.

An activity partition can represent a request port of the owning participant.

A call operation action is an invocation of a service operation available through the request port represented by the activity partition.

This is the receipt of a callback from the consumer through the request port’s provided interface (a two-way conversation).
Assembling the Parts into a Deployable “Subsystem”

Participants can assemble reference to other participants needed to provide their services.

A participant can delegate a service to one of its parts.

These parts represent references to instances of participants that will exist at runtime.

Participants can adhere to a ServicesArchitecture by indicating what roles the parts in the assembly play in the services architecture.

Requests of one participant are connected to compatible services of another.