Session Overview

Preface
SOA Motivation and Business Value
SOA Definition and Context
SOA Service Delivery
Legacy Migration Strategy Using SOA
AT&T SOA Journey
Summary
Preface: SOA Hype

SOA is **passing through the hype cycle and has finally ‘achieved’ a measure of derision**!

- SOA started with lots of promise and hope.
- So consultants and vendors co-opted the word to sell their engagements and products.
- Now many people confuse SOA with web services, ESBs, governance systems and more.
- Organizations are also suing consultants over projects which have failed to deliver ‘SOA benefits’.
- So it is starting to becoming fashionable to deride SOA.

The question is: how many CIOs have truly embraced SOA (and what exactly is that)?
Preface: One Size Doesn’t Fit All

Different organizations have different goals and challenges and need different SOA programs.

• **Needs may vary**
  - Process Flexibility
  - Information Agility
  - Legacy Consolidation
  - Time to Market
  - Cost Reduction

• **Organizational and ‘political’ challenges may vary**
  - Command and control effectiveness
  - Number of organizations involved
  - Outsourcing
  - Funding models
SOA Motivation and Business Value
**Traditional Interface Development**

*Different project teams create similar interface functionality on the same systems in the same release cycles.*

<table>
<thead>
<tr>
<th>Project 1 - Software Development Process Steps</th>
<th>Project 2 - Software Development Process Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Determine conceptual solution including necessary system interfaces</td>
<td>1. Determine conceptual solution including necessary system interfaces</td>
</tr>
<tr>
<td>2. Concept Gate approval without consideration of target interfaces or reuse</td>
<td>2. Concept Gate approval without consideration of target interfaces or reuse</td>
</tr>
<tr>
<td>3. Prepare Business Reqs with high level description of interfaces</td>
<td>3. Prepare Business Reqs with high level description of interfaces</td>
</tr>
<tr>
<td>4. Reqs Gate approval with no consideration of interface target or reuse</td>
<td>4. Reqs Gate approval with no consideration of interface target or reuse</td>
</tr>
<tr>
<td>5. Prepare Tier3 Requirements &amp; IAs</td>
<td>5. Prepare Tier3 Requirements &amp; IAs</td>
</tr>
<tr>
<td>6. Commit Gate approval without consideration of interface target or reuse</td>
<td>6. Commit Gate approval without consideration of interface target or reuse</td>
</tr>
<tr>
<td>8. Design Gate approval without consideration of interface target or reuse</td>
<td>8. Design Gate approval without consideration of interface target or reuse</td>
</tr>
<tr>
<td>9. Implement, Test and Deploy</td>
<td>9. Implement, Test and Deploy</td>
</tr>
</tbody>
</table>

**Project architects specify project-specific interfaces without trying to reuse/extend what exists nor create a desired target API**

**Project teams work independently with little collaboration cross-team and end up deploying similar interface functionality**

*Note: Interfaces designed in the fire of project urgency are usually not reusable in different contexts.*
Impact of a SOA Development Process

SOA cuts interface cost, complexity and time to market

OCTOBER RELEASE VIEW:

Benefits increase over time as the library of services grows

<table>
<thead>
<tr>
<th></th>
<th>Project Interfaces</th>
<th>Release-Relevant Total*</th>
<th>Dev $</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>New</td>
<td>Extended</td>
<td>Reused</td>
</tr>
<tr>
<td>Sys1</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Sys2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Sys3</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sys4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sys5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sys6</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Traditional Approach

* Includes relevant interfaces that could have been reused

Cuts development cost through interface reuse

Benefits increase over time as the library of services grows

<table>
<thead>
<tr>
<th></th>
<th>Project Interfaces</th>
<th>Release-Relevant Total</th>
<th>Dev $</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>New</td>
<td>Extended</td>
<td>Reused</td>
</tr>
<tr>
<td>Sys1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Sys2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sys3</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Sys4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sys5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sys666</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

With SOA Development Process

Cuts development cost through interface reuse

Benefits increase over time as the library of services grows

<table>
<thead>
<tr>
<th></th>
<th>Project Interfaces</th>
<th>Release-Relevant Total</th>
<th>Dev $</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>New</td>
<td>Extended</td>
<td>Reused</td>
</tr>
<tr>
<td>Sys1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Sys2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sys3</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Sys4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sys5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sys666</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Cuts development cost through interface reuse

Benefits increase over time as the library of services grows

<table>
<thead>
<tr>
<th></th>
<th>Project Interfaces</th>
<th>Release-Relevant Total</th>
<th>Dev $</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>New</td>
<td>Extended</td>
<td>Reused</td>
</tr>
<tr>
<td>Sys1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Sys2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sys3</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Sys4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sys5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sys666</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Cuts development cost through interface reuse

Benefits increase over time as the library of services grows
SOA Case Study

By 2005 AT&T had documented over $40 million in savings from SOA, as in this example of a system that accrued $2.6 million in 2 years by reusing one service across 5 clients.

Highlights:

- Reuse of a single service saved 50%-85% of the cost of building custom interfaces.
- Savings will continue to accumulate as more clients are added.
- Maintenance costs will be lower (not shown) because fewer interfaces need to be versioned and maintained.
- Operational efficiencies will be higher (not shown) because of increased consistency across SOA customer/client interfaces.
SOA Value to AT&T

The SOA benefit model was recast and zeroed out in 2005. It projects additional savings in excess of $100M by 2009.

SOA Benefit Model:

- Service reuse contributes an average 50% reduction in integration cost.
- Includes engineering efficiencies from use of standards, models and repositories.
- Includes development efficiencies from use of standard integration toolkits.
- Without SOA costs and complexity continue to increase.

Key Assumptions:

- Constant annual development budget spend at 2005 levels.
- Rate of re-use of existing services is approximately 3 times per service during a 10 year period.
  - Note: The system on the previous slide provided 5 instances of reuse within 2 years.
- SOA adoption rate grows from 25% of projects in 2006 to 90% of projects by 2009.
- Average overhead to create SOA services for the first time is 10% over the current costs.
- Cost of a new interface is $(att proprietary) on average.
Over the years, many enterprises have developed ‘accidental architectures’ made up of the gradual accretion of systems and applications interconnected with diverse middleware.

The ‘accidental architecture’ misses the primary aim of architecture, which is to break down a complicated problem into simple pieces and drive out complexity to make construction and maintenance easy.

**AFTER – Service Oriented Architecture**

SOA partitions and encapsulates existing capabilities behind a well thought out set of target services.

Solution teams reuse and extend this portfolio services, instead of redeveloping functionality to their specific preferences. Reuse of services cuts cost and speeds time to market.

Once encapsulated, internal infrastructure can be consolidated, enhanced and/or retired.
# SOA Business Value Summary

*With the correct execution strategy, SOA will deliver significant benefits across the enterprise.*

<table>
<thead>
<tr>
<th>Driver</th>
<th>Description</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduced development costs</td>
<td>Reuse &amp; less reinvention of functionality across projects</td>
<td>20% reduction in development cost; 50% savings per reuse</td>
</tr>
<tr>
<td>Reduced maintenance costs</td>
<td>Fewer interfaces, versions and middlewares to maintain</td>
<td>Ongoing cost savings beyond development</td>
</tr>
<tr>
<td>Reduced complexity</td>
<td>Encapsulates complexity behind simple service interfaces</td>
<td>Teams see SOA services; not legacy systems and technology</td>
</tr>
<tr>
<td>Reduced effort in design &amp; testing</td>
<td>Complexity reduction leads to easier design and testing</td>
<td>Higher quality features; reduced fallout</td>
</tr>
<tr>
<td>Accelerated time to market</td>
<td>Reuse and complexity reduction cuts time required to deliver new features</td>
<td>Greater responsiveness to competitive pressures</td>
</tr>
<tr>
<td>Increased solution assembly</td>
<td>Solutions are delivered by orchestrating a library of existing services</td>
<td>Process centric solutions; more time for business logic</td>
</tr>
<tr>
<td>Easier systems consolidation</td>
<td>Breaks the direct link between users and legacy assets</td>
<td>Business can dictate when &amp; where to rationalize assets</td>
</tr>
<tr>
<td>More integrated &amp; agile processes</td>
<td>Process tasks leverage a growing library of SOA services</td>
<td>Reduced re-keying &amp; input errors</td>
</tr>
</tbody>
</table>
SOA Definition and Context
The Challenge of Realignment

SOA is a ‘Realignment’ challenge rather than a ‘Turnaround’ or ‘Startup’ challenge.

- Realignments challenge established cultural norms that hamper high performance.
  - Realignments are important to business success.

- But in realignments, the situation is not dire, leading many to feel change is not necessary.
  - So realignment advocacy is like farming: cultivating awareness of the need for change, influencing opinion leaders, keeping the pressure on.

- Unlike startups or turnarounds, the impact of realignment is not always appreciated.
  - If abandoned, most will not know what could have been.

- Major realignments like SOA need strong executive support to overcome inertia and resistance.
AT&T Definition of SOA

SOA is an overloaded term which requires definition and alignment.

“SOA” is an approach to architecture & solution design which:

• Decomposes a domain or application into a set of abstract, highly reusable target functional interfaces (called target ‘services’).
• Brings governance to the design and selection of services as projects flow thru the development cycle, encouraging both reuse and build-out of target.

To support SOA:

• A foundation of middleware, taxonomy and naming standards must be put in place, along with repository / management tools and governance.
• Target architects & lead engineers must functionally decompose their applications & enterprise domains into a set of highly reusable target services, which solution designers can reference in their designs and build out over time.

A common misconception is to equate SOA with Web Services or integration technologies like ESBs.
The SOA Reuse Challenge

Reuse requires a repository of existing & target interfaces, plus governance or minimally buy-in from app owners.

Typically, 1/3 of development is spent on interfaces, many of which are project-specific and not easily reused. API functionality is reinvented by parallel teams sometimes in the same release.

Reusable interfaces are difficult to think up in the fire of project urgency. If they were, reuse would be widespread today.

Reuse of existing interfaces saves 40-80% over building new. Where new interfaces must be built, they should be built for reuse.

There are no simple guidelines for generating a reusable target—it’s doing good design: layering, modularizing, data flows/models, use cases, strategy for dynamic elements... But an online handbook will help the community.

If they buy into their target, then even without governance oversight, app engineers will propose & advocate the target.

Reuse will require: 1) analyzing & specifying reusable interfaces outside the fire of project delivery; 2) applying that target to projects during development.

A good target is agreed by architects, engineers and developers: reps should be appointed for each app and/or functional area.

Savings: development (reuse), maintenance, ops & design (simplicity). Costs: target definition, tools & governance.
SOA Actors
A SOA program needs to successfully engage five teams.

STRATEGIC SOA

Target Architecture:
Target Systems and Functional APIs

Solution Realization:
Project Feature Development

Governance:
Project Management & SDLC

Executive Support

TACTICAL SOA

Standards & Tools: Middleware, SOA, Naming, Taxonomy

Specifies target capabilities to be built out over time; works with project teams to resolve issues

Builds production capabilities referencing the target; specifies SOA service impacts in solution designs
**Building the SOA Puzzle**

The challenge of SOA in a large enterprise can be related to the assembly of a complex puzzle.

<table>
<thead>
<tr>
<th>Contributor</th>
<th>Output</th>
<th>Puzzle Analogy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tactical Aspect of SOA</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Solution Designers</td>
<td>Specification of SOA service impacts in solution designs</td>
<td>Each solution team assembles a few pieces; over time they assemble the whole puzzle.</td>
</tr>
<tr>
<td><strong>Strategic Aspect of SOA</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Architects</td>
<td>The ‘To Be’ view of functional capabilities by domain*</td>
<td>Definition of the puzzle to be built out over time, including the shape of each piece.</td>
</tr>
<tr>
<td><strong>Foundation of SOA</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical Architects and SOA Support</td>
<td>Interoperability standards, SOA tools and governance</td>
<td>Puzzle piece requirements and governance of the assembly process.</td>
</tr>
</tbody>
</table>
SOA Service Delivery
Strategic SOA: Target Services
Strategic SOA is tasked with functionally decomposing the architecture into a set of abstract target services*.

- Decomposition starts by dividing the architecture into domains.
- Working top-down, domain teams define the target services they will provide to other domains.
  - Domains expectations must be vetted cross-team.
  - Service functionality is described abstractly without regard to protocol or implementation details.
- The hard work is the mental analysis that must be performed.
  - Modeling tools can help document decisions but are not essential to the actual analysis.
- Service definitions must extend to the data flowing in/out of operations or the target won’t be stable or usable.

* Note: while target interfaces are optimal for the enterprise as a whole, they may not be optimal for all projects or clients.
Data Standardization

SOA APIs express an implicit data model, which ideally should be identified to increase the comprehensibility & consistency of service definitions.

Target Object Requirements:
- Technology for modeling and describing objects
- Standards for reuse and aggregation of base components
- How to identify, propose and approve objects

SOA Service Best Practices:
- Strategy for using complex objects in SOA APIs
- Strategy for object-agnostic services

Data Dictionary:
- Store and discover target objects
- Link to SOA Repository operation parameters
- Document data translations for key applications and flows
Benefits of Target Data Objects

Establishing target objects provides a number of benefits

- **Knowledge Capture**
  - Synthesizes knowledge drawn from diverse participants
  - Models knowledge precisely

- **Reduces Fallout**
  - Reduces confusion about the meaning and usage of data leading to reduced order fallout

- **Enhances SOA Service Reuse**
  - Systems speak a common language with all of their clients
  - Curtails the practice of legacy systems asking servers to provide interfaces in their proprietary data abstractions
  - Systems must translate their legacy abstractions to the canonical standard but are freed from having to do pair wise translations with every interfacing system

- **Improves Data Quality**
  - Supports the initiative to consolidate legacy data into standard data services

- **Enhances Use of Legacy**
  - Legacy can be more easily accessed when wrapped with SOA interfaces leveraging standard middlewares and standard data abstractions

- **Cost Reduction & Time To Market Improvements**
  - From reduced fallout, reuse of data translations, enhanced understanding, reduced complexity, increased service reuse.
A SOA framework is used for consistent delivery of SOA services across parallel teams.

- A SOA Taxonomy divides the enterprise into domains
- SOA Naming Standards improve service discovery
- Best Practices provide guidance on service scope
- Architects specify the ‘To Be’ Target View for each domain
- An Inventory of existing services is performed
- A SOA Repository captures service definitions online
- A Solution Design Flowchart specifies how Designers interact with Target Architects
- A Service Inventory template captures service impacts in SOA Design Templates
SOA Tools

SOA tools are used to manage adoption, performance & reuse of SOA services, plus compliance with standards.

- A SOA Repository captures service definitions online:
  - Promotes reuse of services and communicates the target
- A simple SOA Logger may be used to log SOA activities at various levels of detail:
  - Captures clients, versions, access frequency, latency, dependencies and more
- A SOA Dashboard tracks reuse and adoption of the ‘To Be’ target by application & domain
- A Data Dictionary is the database of record for target data objects and abstractions

<table>
<thead>
<tr>
<th>Log Level</th>
<th>Type of Logging</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Logging</td>
<td>Turn off all logging, Low latency under load</td>
</tr>
<tr>
<td>1</td>
<td>Invocation Logging</td>
<td>Log Service invocations only, Frequency of access, client dependencies, version changes</td>
</tr>
<tr>
<td>2</td>
<td>Authorization Logging</td>
<td>success or failure, Probing and attacks can be detected</td>
</tr>
<tr>
<td>3</td>
<td>Exit Logging</td>
<td>Log all exits from the Service, Service Performance can be derived</td>
</tr>
<tr>
<td>4</td>
<td>Call Trace Logging</td>
<td>Log all calls made from the Service to other Services, Service dependencies and call traces can be graphed</td>
</tr>
</tbody>
</table>
‘Target’ and Implemented Services

The SOA Repository supports both target & implemented services.

‘Target services’ describe ‘To Be’ functional capabilities to be built out over time, as defined by architecture.

“Implemented services” are actually placed in production, and are further qualified by a “Phase” attribute.

The SOA Repository is a design time discovery tool leveraged heavily by Solution Designers working on time to market projects. The UI is optimized for fast assimilation of enterprise service functionality down to the data passed to/from operations.
**Tactical SOA: Delivering Solutions**

Tactical SOA is tasked with specifying and reviewing abstract SOA services in project solution designs.

- Design Templates and Governance Processes must be modified to capture and review service choices & impacts.
  - Specific target/production services must be identified, but the focus is on abstract service functionality (down to the data level); not on middleware.

---

### Service Inventory Template

<table>
<thead>
<tr>
<th>Facility</th>
<th>Group</th>
<th>Name</th>
<th>Operation Name</th>
<th>Description</th>
<th>Date Panned</th>
<th>Data Returned</th>
<th>Impact</th>
<th>New System</th>
<th>Middleware</th>
<th>Fix/Not Fix/Middleware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facility</td>
<td>Group</td>
<td>Name</td>
<td>Operation Name</td>
<td>Description</td>
<td>Date Panned</td>
<td>Data Returned</td>
<td>Impact</td>
<td>New System</td>
<td>Middleware</td>
<td>Fix/Not Fix/Middleware</td>
</tr>
</tbody>
</table>

**Service Impacts must be specified as summarized on the next slide.**

---

**SOA Services Architecture Diagram**

- Event Subscription Service
  - Operation: subscribe
  - Invoked via an event received by the GenericEventListener WSDL

- Event Broker

- The eTOM popsicle stick convention shown at right is used to depict service-operations. The circle represents a service-operation and extends out from its host. Clients connect to it.

- Sys1: Capacity Check Service
  - Operation: initiateCapacityCheck
  - Create a target operation
  - Expected
  - Forum Approval
  - Approved Exception

- Sys2: Event Submission Service
  - Operation: notify
  - Create a non-target operation
  - Expected
  - Forum Approval
  - Approved Exception

- To Be Deleted

---

Note: Every service listed here must be present in the SOA Repository! Leverage the Target Architects and SOA COE per the Design Solutioning Flowchart.
Governance

SOA drives benefit by governing the creation and use of interfaces

Architects & Engineers Design COE

Architects & Engineers

Create & Refine Target APIs

Review (Naming, Taxonomy, Granularity, Data Objects)

Review (Naming, Taxonomy, Granularity, Data Objects)

Review (Naming, Taxonomy, Granularity, Data Objects)

Design COE

Manage Existing APIs

Enter Implemented APIs

Approve (Middleware Technical Standards)

SOA Repository

SOA Enable Design Templates

Review (Naming, Taxonomy, Granularity, Data Objects)

Approval

Design Solution

SOA Modified Development Process Design Templates

Discover Existing/Target APIs

Discover Existing/Target APIs

IRB COE

Approve (Naming, Taxonomy, Granularity, Data Objects)

Design COE

Log Invocations (Per SOA Log Level)

Receive & Process (Batch or Real Time Updates)

SOA Logger

SOA Dashboard

Reports

( Clients, Dependencies, Repository Errors, & Progress)

Architects

Express/One Team

Existing APIs

Existing Production APIs

Tactical Projects

Strategic Groundwork

Supporting Artifacts and Infrastructure

Runtime Activities

Governance

Tactical Projects

Strategic Groundwork

Supporting Artifacts and Infrastructure

Runtime Activities

SOA drives benefit by governing the creation and use of interfaces
Foundation Integration Strategy

A real world Enterprise will blend two fundamentally different ways of approaching integration.

- **The Integration Broker or ESB approach:**
  - The challenge of ESBs lies not their technology but in the centralized organization required to support and develop integrations with it.
    - Many protocols and handshaking schemes can be supported but,
    - To the extent that it gets involved in pair-wise integrations, the ESB organization must be able to provide resources when they are requested or risk being branded as a bottleneck.

- **The Interoperability Standard approach:**
  - The interoperability standard approach specifies enterprise-standard protocols and handshaking schemes, and requires all network endpoints to develop support for those standards.
    - Once exposed to the network, a service is consumable by all endpoints.

The choice of integration strategy has significant time and cost implications.
The ESB / Integration Broker Approach

- Supports many protocols and handshaking schemes
- Extra development (relative to an interoperability approach)
- The ESB Team has great control over enforcement of SOA Standards
- Potential bottleneck if the ESB organization gets involved in most client-server integrations (organization scalability issue)

Overview:
- Organization A
  - Service ‘A’
    - Client ‘1’
      - Protocol ‘1’
        - Handshaking ‘1’
    - Client ‘2’
      - Protocol ‘2’
        - Handshaking ‘2’
- Client ‘1’
  - Service ‘A’ Adapter
- Client ‘2’
  - Service ‘A’ Adapter
- ‘A’ Client
  - Client ‘1’ Organization
  - Client ‘2’ Organization
The Interoperability Standard Approach

- All endpoints support standard handshaking & protocols
- Control over SOA standards is more difficult since integration are implemented by many teams
- Leveraging new protocols is as easy as with the ESB approach

Service providers may have to express their services in more than one protocol (as dictated by the standard)

There is no need to interlock with a centralized ESB organization

Interoperability Standard

- Organization A
  - Service ‘A’
  - Standard Protocol ‘1’
  - Standard Protocol ‘2’

‘A’ Client
- Client ‘1’ Organization
- Client ‘2’ Organization
Interoperability Standard
An Interoperability standard must minimally cover:

- **Protocol Standards**
  - Strategy for requirements that exceed protocol capabilities
  - Web Services has issues with large messages and extreme performance

- **Handshaking**
  - Credentials, callbacks, message ID, guaranteed delivery and other message semantics

- **Vendor Product Standards**
  - Definition of the interoperable subset of protocol capabilities
  - WSDL & XML Schema Best Practices for Web Services

- **Versioning strategy**

There is a danger in relying on emerging industry standards since vendors often implement them inconsistently.
Application Centric SOA

If enterprise wide modification of the SDLC is not possible, an application centric SOA approach may be used instead.

- Prioritize key applications
- Inventory their existing services
- Specify target services in detail
- Implement target interfaces
- Drive reuse thru the time/costing process
  - Build the NPV of the future stream of maintenance and operational costs into estimates for non-standard functionality:
    - Make non-target unacceptably costly!

AT&T has had success with different approaches. Think out of the box!
Legacy Migration Strategy Using SOA
Legacy Transformation Starting Point

The system consists of a number of different applications with app-to-database, app-to-app & app-to-user interfaces.
Thread 1: DB Transformation Step 1A

Build a target services layer around the legacy databases while retaining existing direct application access to those legacy databases.

Introduce a target services layer around the DB while maintaining legacy I/Fs
Thread 1: DB Transformation Step 1B

Incrementally wed the legacy apps to the new target services and retire the direct DB accesses.

Incrementally wed the legacy apps to the new target services (& retire the direct DB accesses)
Thread 1: DB Transformation Step 1C

Optimize and replace the legacy DBs without affecting the existing apps (which are shielded from changes by the target services layer).

Optimize / replace the legacy DB without affecting the existing apps
Thread 2: App Transformation Step 2A

Analyze & document the existing functional interfaces between apps (and exposed to users via their presentation layers).

Analyze and document the functional interfaces (i.e., services) being exposed between existing apps.
Thread 2: App Transformation Step 2B

Based on the analysis of what exists, specify a set of idealized capability modules interconnected with target services interfaces.

Based on the analysis of what exists, specify a set of idealized capability modules with target services interfaces.
Thread 3: App Transformation Step 3A

*While leaving existing app interfaces in place, build out a wrapper layer over the existing apps which implements the target service interfaces.*

Build out a wrapper / orchestration layer around the existing apps which implements the target services over the existing apps.
Thread 3: App Transformation Step 3B

Incrementally wed the existing apps to the target APIs and retire the legacy app-to-app interfaces.

Incrementally wed apps to the target APIs and retire the legacy inter-app interfaces.
Thread 4: App Transformation Step 4A
Build out the target capability modules from Step 2B.

Build out the target capability modules
Thread 4: App Transformation Step 4B

*Incrementally replace the existing apps with the target capability modules (this includes wedding users to the target presentation layers).*
Thread 4: App Transformation Step 4C

Incrementally replace the existing apps with the target capability modules (this includes wedding users to the target presentation layers).

Incrementally replace the existing apps with ‘plug and play’ capability modules
Thread 4: App Transformation Step 4D

Incrementally replace the existing apps with the target capability modules (this includes wedding users to the target presentation layers).

Incrementally replace the existing apps with ‘plug and play’ capability modules.
AT&T SOA Journey
2001-2003: Web Services Strategy

A Web Services interoperability strategy was adopted in 2001 with the hope of ending redundant interface creation.

- 2001: Web Services standard adopted
- 2002: First Interop Standards Published
- 2002: Formulate standards & test vendors
- 2003: First WS Interfaces Appear
- 2003: SDLC Process Interlock
- 2003: Political battles over interface reviews
- 2004: Executive Edict: Thou Shalt Use Web Services!
- 2004: Redundant interfaces common but the IRB detects them too late
- Realization: We Need SOA
2004-2006: SOA Strategy

In 2004, the Web Services Interoperability strategy matured into a true SOA strategy with strong Executive support.

- **2004**: SOA Center of Excellence (COE) Created
- **2004**: Data COE Created
- **2005**: SDLC Process Interlock for SOA
- **2005**: Design template modifications & SOA governance strategy
- **2005**: Product agnostic APIs & target enterprise objects
- **2006**: SOA Tool Updates & Merger Planning
- **2006**: Milestone: The business signs off on $40 Million in SOA Savings
- **2006**: SOA Tool Updates incl. Dashboard
- **2006**: Target Service Definition Begins

**Milestone**: The business signs off on $40 Million in SOA Savings

**Start of target definition**: 750 operations in 3 passes over 1.5 years

**Product agnostic APIs & target enterprise objects**

**Design template modifications & SOA governance strategy**

**SDLC Process Interlock for SOA**

**Labs wide communications & training**

**SOA plan, approach & ROI model**

**Naming, Taxonomy & Best Practices**
Summary
Summary

A SOA program offers great potential to cut cost, complexity and time to market.

- The SOA program comes into focus when the business goals are clearly articulated and quantified.

- Defining terms is very important since the words ‘SOA’ and ‘service’ are heavily overloaded and have been successfully appropriated by vendors.

- Executive support is essential to overcoming inertia & resistance. Many will question the need for SOA and the rationale for changing familiar practices.

- An ESB is not the only integration option for implementing SOA. In 2001, AT&T adopted a highly scalable interoperability strategy in which ESBs were the exception and not the rule.

- Other keys to success are: changing the development process, inventorying existing services, defining target services, deploying an online repository, and adopting a runtime management strategy & dashboard.
For more information regarding this presentation:

Please Contact
Rich Erickson
AT&T Labs
rerickson@att.com
732.567.5513