The Path to SOA for ISVs

Ronald Schmelzer
Senior Analyst
ZapThink, LLC

Take Credit Code: SOAISV

zapthink

ISV Constant: Change

A ISV's Business is Never STATIC

Challenge 1: Integration

- Requirement #1:
  - Make it easier to get your products working within your customer's environments
- Requirement #2:
  - Make your products work better with each other
- Requirement #3:
  - Keep necessary involvement of professional services to a minimum

The reality: too much work goes into simply getting products to work.

Challenge 2: New Functionality and Increasing Complexity

- Requirement #1:
  - Meet new customer requirements with new functionality
- Requirement #2:
  - Stay ahead of competition by matching existing features and introducing new capabilities
- Requirement #3:
  - Do so without breaking everything else
  - Do so without making Challenge #1 worse

The reality: IT products have reached a point of maximum complexity – additional complexity brings lowered value.

The increasing customer refrain:
"Don't we already have one of those…?"

Challenge 3: Versioning and Change

- (Meta) Requirement #1:
  - Solve the requirements of the masses as well as the requirements of each individual company
- Requirement #2:
  - Solve ongoing threats to product stability: security, reliability, scalability
- Requirement #3:
  - Allow incremental change without making Challenges #1 and #2 worse

The reality: ISVs can barely keep up with their own requirements for change, let alone those of their customers.

Challenge 4: Interface and Workflow

- Requirement #1:
  - Customer demands for usability are changing by the month
  - New end-user platform requirements every year
- Dilemma
  - Should you provide your own user interface or not?
  - Who is responsible for the user experience?

Reality: Most ISVs should not be in the business of building user interface. The problem is that interface is everything.

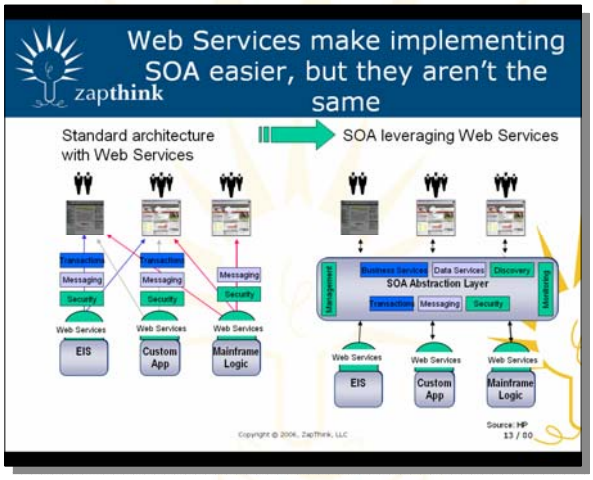
An Answer

- Don't simply put lipstick on the pig… It's not a matter of changing interfaces, it's a matter of changing how the functionality is built, packaged, delivered, and managed.

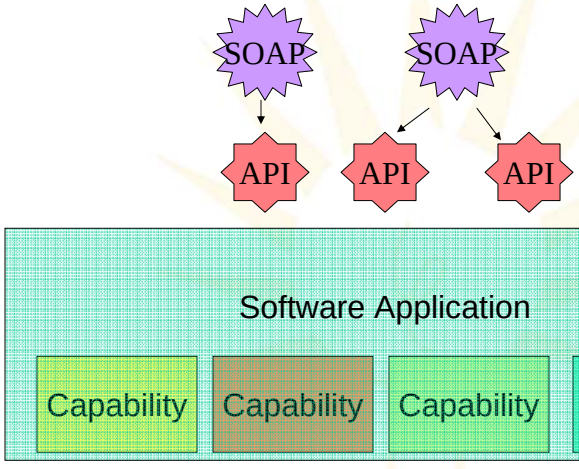
In other words… it's about *ARCHITECTURE*

Web Services or SOA?

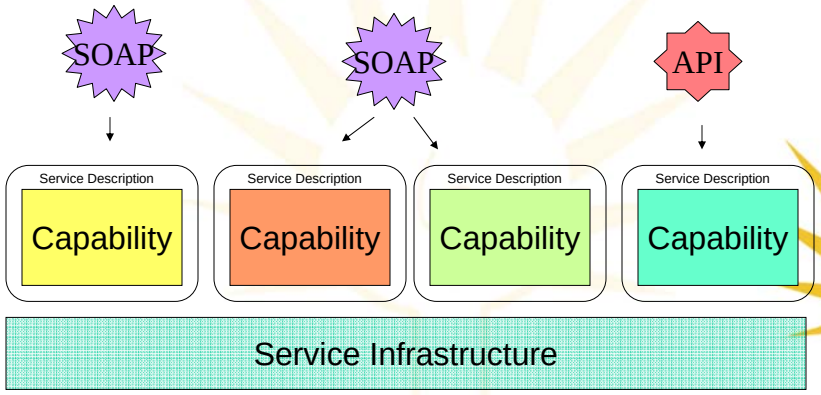
- Let's revisit a slide….



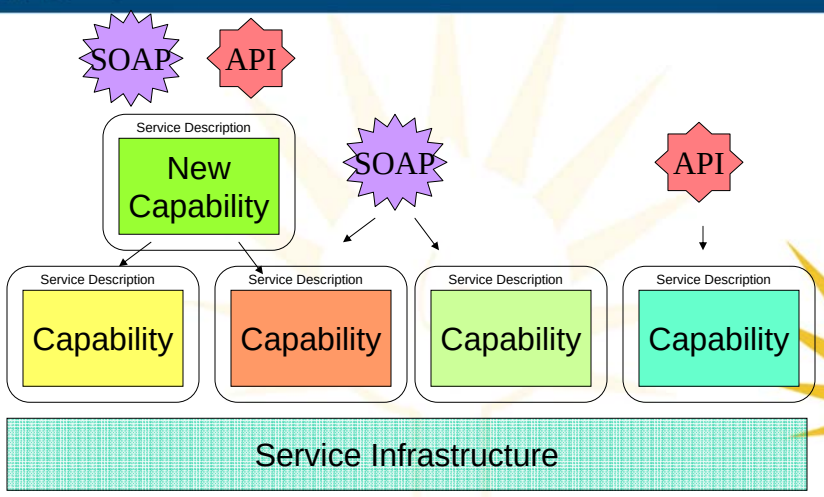
The Baby Step: Service Interfaces



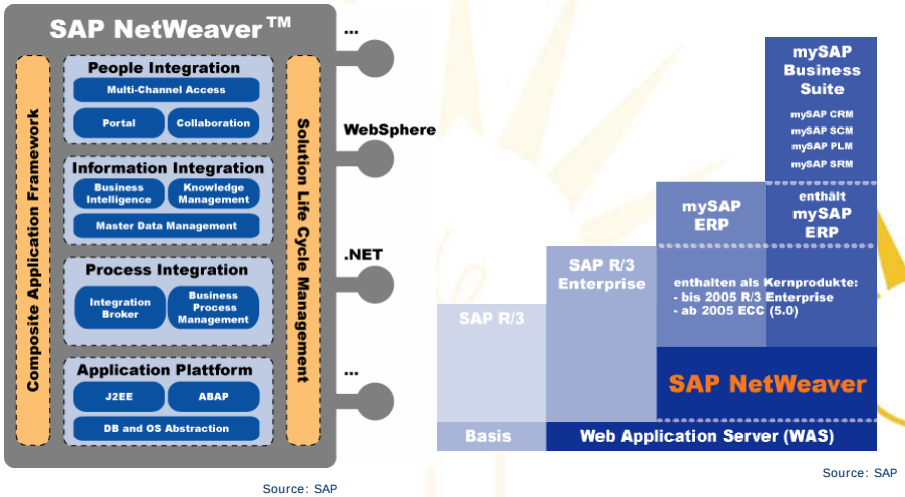
A Bigger Step: Loosely-Coupled Services



One Step Further: Service-Oriented Process



NetWeaver: The New SAP?



Source: SAP

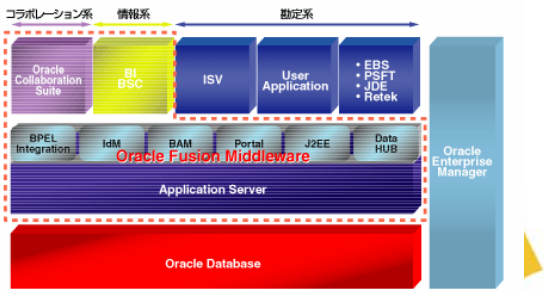
Source: SAP

NetWeaver: The Realities

- "SAP NetWeaver unifies integration technologies into a single platform and is pre-integrated with business applications, reducing the need for custom integration. The platform is based on industry standards and can be extended with commonly used development tools such as Java 2 Platform, Enterprise Edition (J2EE); Microsoft .NET; and IBM WebSphere."
- But what's really out there and complete?
- Can you say ***SAP Upgrade?!***

Oracle Fusion

Image taken from:
http://www.oracle.com/products/middleware/index.html



Source: Oracle

Does this make ANY sense? (Yet)

- Oracle Fusion Middleware
  - Application Server (all editions)
  - Business Integration
  - Business Intelligence
  - Content and Collaboration
  - Data Hubs
  - Identity Management
  - Oracle Fusion Middleware for PeopleSoft
  - Portal
  - SOA Suite
  - Web Services Manager

CA's IT Management Services

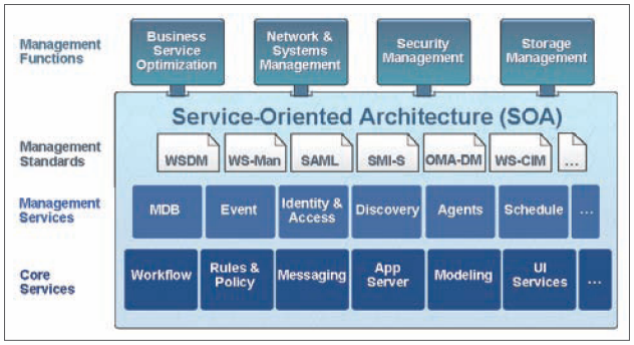
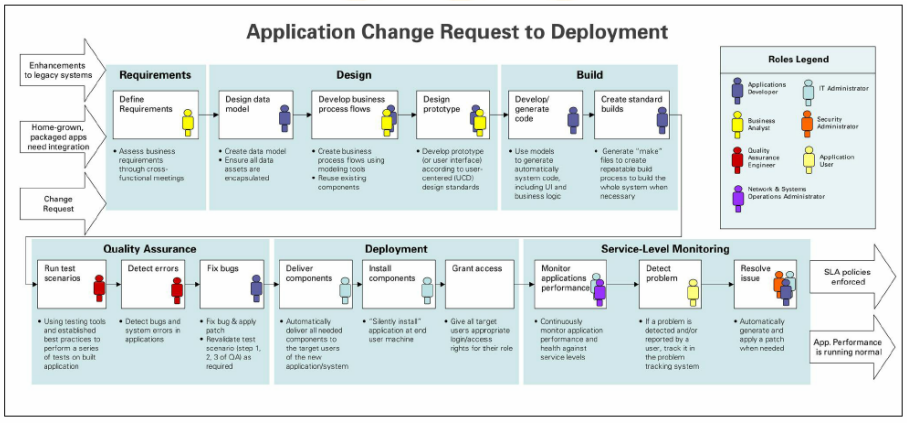


Figure 3. CA's Integration Platform

Source: CA

CA's Integrated IT Flows



Source: CA

ISVs: Services and/or SOA Platform?

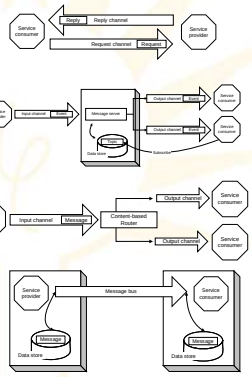
Just because you provide Services, does this mean you need to provide the runtime infrastructure?

The need for SOA Infrastructure

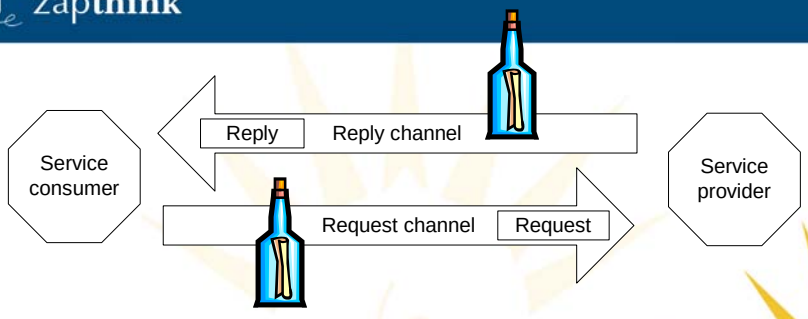
- Service-Oriented Architecture is a set of *design principles, methodologies, and best practices* for implementing distributed computing using *loosely-coupled, composable Services* on a heterogeneous network.
- However, SOA doesn't define exactly ***how*** to implement those Services, or what infrastructure to use to make sure the Services can communicate with each other or be made available on the network
- In fact, SOA as a design is ***not supposed*** to mandate specific implementation approaches – that's how we can guarantee loose coupling in the face of heterogeneity.

Invocation Mechanisms in SOA

- SOA is more flexible than client/server – supports multiple invocation mechanisms
  - Request/Reply
  - Publish/Subscribe
  - Routed Events
  - Reliable Messaging

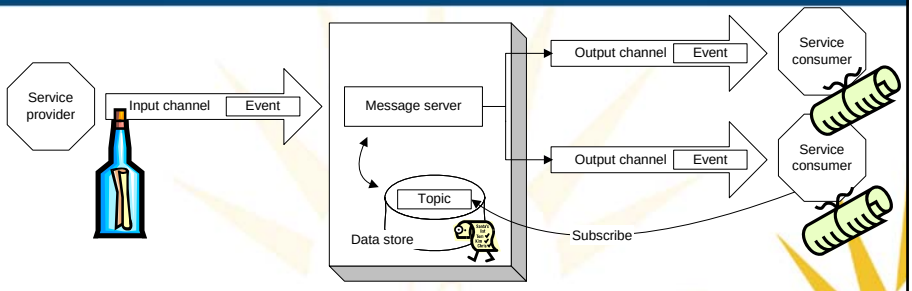


Request/Reply



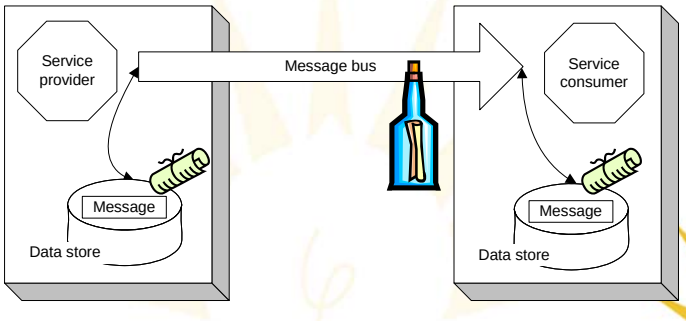
- Synchronous interaction made up of two asynchronous interactions

Publish/Subscribe



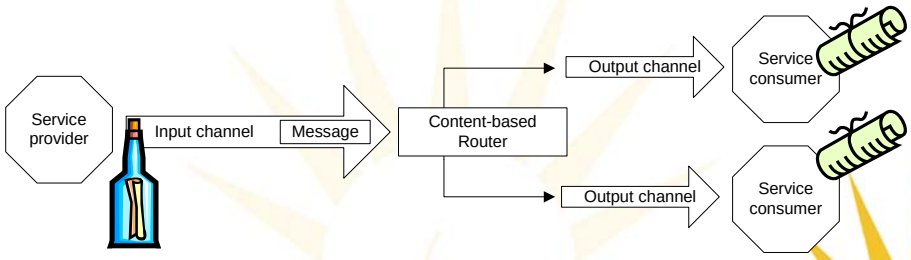
- Consumers subscribe to topic
- Provider creates event
- Message server duplicates and routes events depending on subscription info

Reliable Messaging



- Guarantees message delivery
- Can also guarantee messages sent only once, delivered in correct order

Routed Events



- Role of integration broker – transforms, routes messages
- Looks inside message
- ESBs offer this functionality

The "Enterprise Service Bus"

- What is an ESB? TRICK QUESTION!
  - ***MOM++***
    - Messaging infrastructure or message-based middleware that can natively deal with Service interfaces and composition
  - ***Service Intermediary / Broker***
    - A broker-based intermediary (hub-and-spoke) for intermediating Service requests on behalf of Service consumers and producers
  - ***EAI++***
    - EAI technology "warmed over" with Service interfaces, but nothing in the infrastructure itself that is Service-Oriented
  - ***Architectural Pattern, not Product***
    - ESB is not a product at all, but an "architectural pattern" – that is, you can implement an ESB using whatever technology you want, as long as it gives you the design goals of secure, reliable, robust communication using message-based or asynchronous styles as well as synchronous.
- The real issue:
  - HTTP / Web Services by itself is too immature to support secure, reliable, and robust long-lived Service interaction, so a stronger infrastructure is needed, hence the need for "something more" than just Web Services…
  - People acknowledge that an ESB is a sort of infrastructure for implementing SOA, but there's ***NO AGREEMENT*** on what an ESB should do, nor what it should look like.

SOA is something you do -- not something you buy.

Lack of Convergence on ESB Meaning

- Things that ESBs should do:
  - Facilitate Service exposure
  - Enable Service composition
  - Abstract Service runtime heterogeneity
  - Enable event-driven and asynchronous modes of Service interaction
- Things that ESBs *can* do:
  - Business Process Management / Modeling (BPM)
  - Registry / Repository capability
  - Provide an actual message bus / message-oriented middleware (not necessary!)
- Things that ESBs *shouldn't* do:
  - EAI with Service interfaces. SOA represents an approach to integration that is fundamentally opposed to a hub-and-spoke, tightly-coupled approach to integration. Composition favored over explicit mapping

"At this point in time, no single vendor provides a complete SOA infrastructure, and certainly no single product can provide a complete infrastructure."

- Anne Thomas Manes

State of ESB Market at beginning of 2006

- Messaging Middleware Centric
  - Sonic Systems
  - Fiorano Software
  - TIBCO
- App Server Centric
  - IBM (WebSphere ESB)
  - Oracle (Fusion)
  - BEA (AquaLogic Service Bus)
- EAI Re-dos or Hub-and-Spoke Focused
  - Sun (SeeBeyond)
  - IONA
  - WebMethods (Fabric)
- Web Services Centric / Service Interface focused
  - Cape Clear

ISV SOA Platform Requirements

- Running Services Reliably
- Making *loosely coupled* Services a Reality
- Platform independence – support your customer's heterogeneity
- Web Services is not enough
- Security capabilities
- Composable business processes
- Messaging vs. RPC

Key WS-* Standards

- The "basics" (WS-I Basic Profile)
  - SOAP 1.2
  - WSDL 1.2
  - UDDI 3.0 (← our versioning)
- Security
  - WS-Security
  - SAML
- Process
  - BPEL
- Infrastructure
  - WS-Addressing
- Payload
  - XML Schema
  - Digital Certificates, XML Encryption

Service Business Models

- *Option #1:* Sell your product, your Services are another way of deriving value
  - Issue: is the CPU licensing model appropriate?
- *Option #2:* Sell access to your Services on a per-transaction basis
  - Issue: Are you penalizing use?
- *Option #3:* Host your Services and provide access on a subscription basis ("SaaS")
  - Issue: Are companies comfortable with this model, and is this appropriate for all Service functionality?

Don't confuse *Service!*

- Software-as-a-Service
  - This is a *business model* that provides access to capabilities through remotely hosted Services someone pays access for
- Application Service Provider
  - This is another name for a hosted Service provider, but these Services may or may not be Service-Oriented. Tightly-coupled Services were the death of the ASP.
- On-Demand Services
  - This is a *state of business* in which application functionality can be accessed in-house, through SaaS, or through utility computing

Conclusion

- Simply building Web Service interfaces to your existing APIs does not make you an SOA
- Furthermore, it doesn't matter much if *you* have service-oriented your solutions if your customers don't realize the benefits
- Service-orientation requires changing the way you build, package, deploy, and manage your capabilities
- Think more about your value and less about the technology that gets in the way of delivering that value

ZapThink is an industry analysis firm focused exclusively on XML, Web Services, and Service-Oriented Architecture.

Thank You!

Ronald Schmelzer
rschmelzer@zapthink.com