SYSTEMS AND SOFTWARE CONSORTIUM
BUILDING BETTER SOLUTIONS TOGETHER

# SOA Tech Exchange

## Paradox: What's Good About SOA Is What's Bad About SOA
## (A Technical Perspective)

**Prepared By: Mark Blackburn, Ph.D.**

## Why This Topic?

- Members confirmed that dependability issues are more difficult to address in SOA efforts than anticipated
  - Dependability, including security, reliability, availability, integrity, confidentiality (and Quality of Service)
  - More difficult problem than with more traditional tightly-coupled systems
- Information Week survey of 273 tech pros [Sept, 4, 2006]
  - 24% say SOA & Web services projects fell short of expectations
    - Of those, 55% say SOA introduced more complexity into IT environments
    - 41% say they cost more than expected
  - Out of all respondents using SOAs & Web services, just 7% say the results exceeded expectations
- Presentation discusses some of the technical challenges
  - What's different with SOA (mostly from a Web services perspective)?
  - Verification, testing, and monitoring where testing is difficult or costly
  - Security
  - Recommendations
  - Conclusion

SYSTEMS AND SOFTWARE CONSORTIUM
BUILDING BETTER SOLUTIONS TOGETHER

## Why Should You Care?

$$$

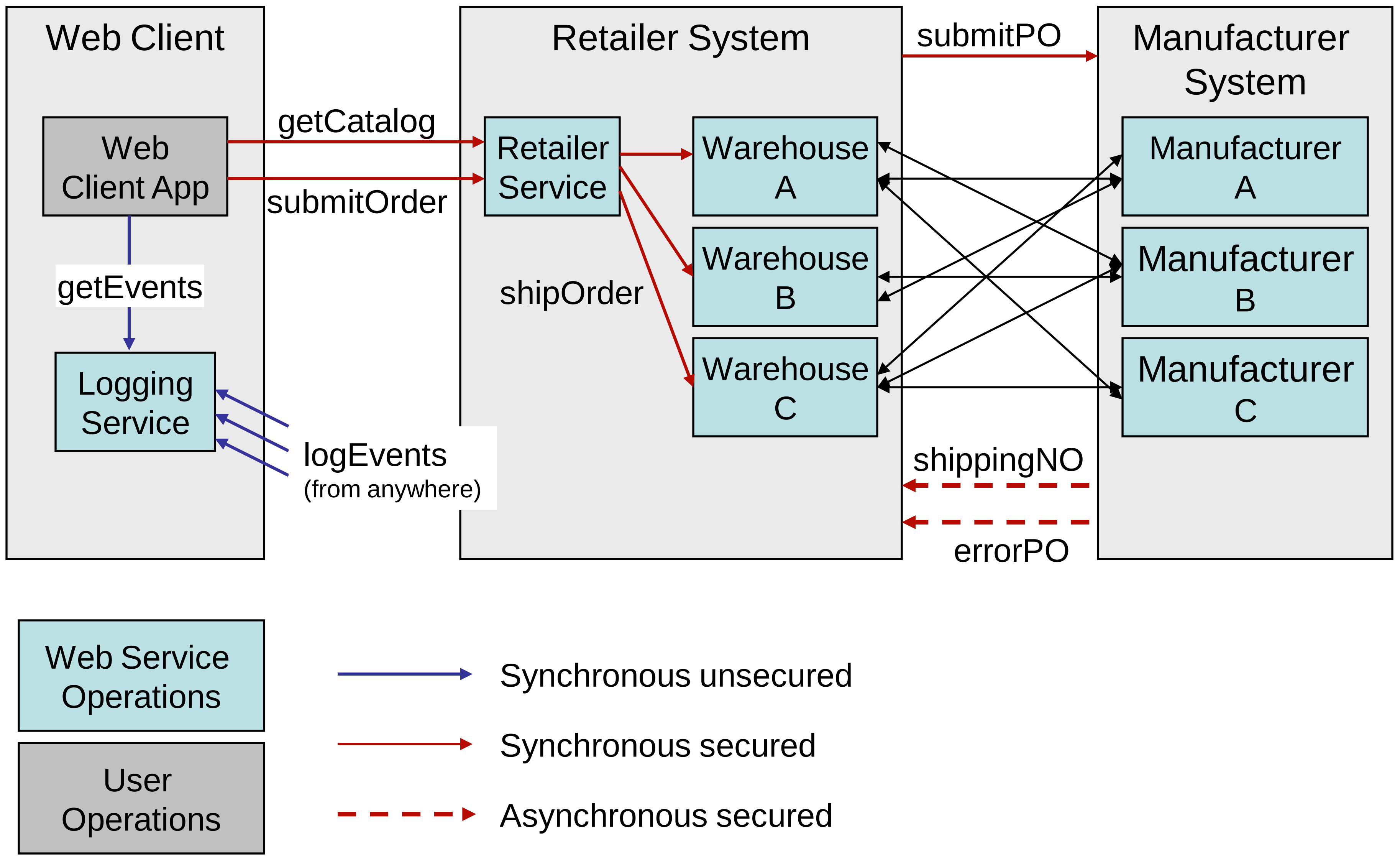
- SOA introduces a new dimension with potentially unanticipated effort/cost
- Liability costs
  - Functioning improperly
  - Not being delivered on time

SYSTEMS AND SOFTWARE CONSORTIUM
BUILDING BETTER SOLUTIONS TOGETHER

## What Makes SOA Challenging?

- Intrinsically distributed
  - Unknown number of possible configurations
  - Focus is on interoperability versus integration
  - Many dependencies - can take a lot of coordination to complete a call through entire chain
  - Web services abstract applications from back-end systems performing the processing
    - We might not know who we’re “talking” to, resulting in need for new type of requirements => added cost and effort
  - Security issues at every interface
- Infeasible to complete testing of all business workflows across heterogeneous technology layers at system and component levels
- Changing requirements and evolving systems
  - What worked yesterday might not tomorrow

# Simplified Example



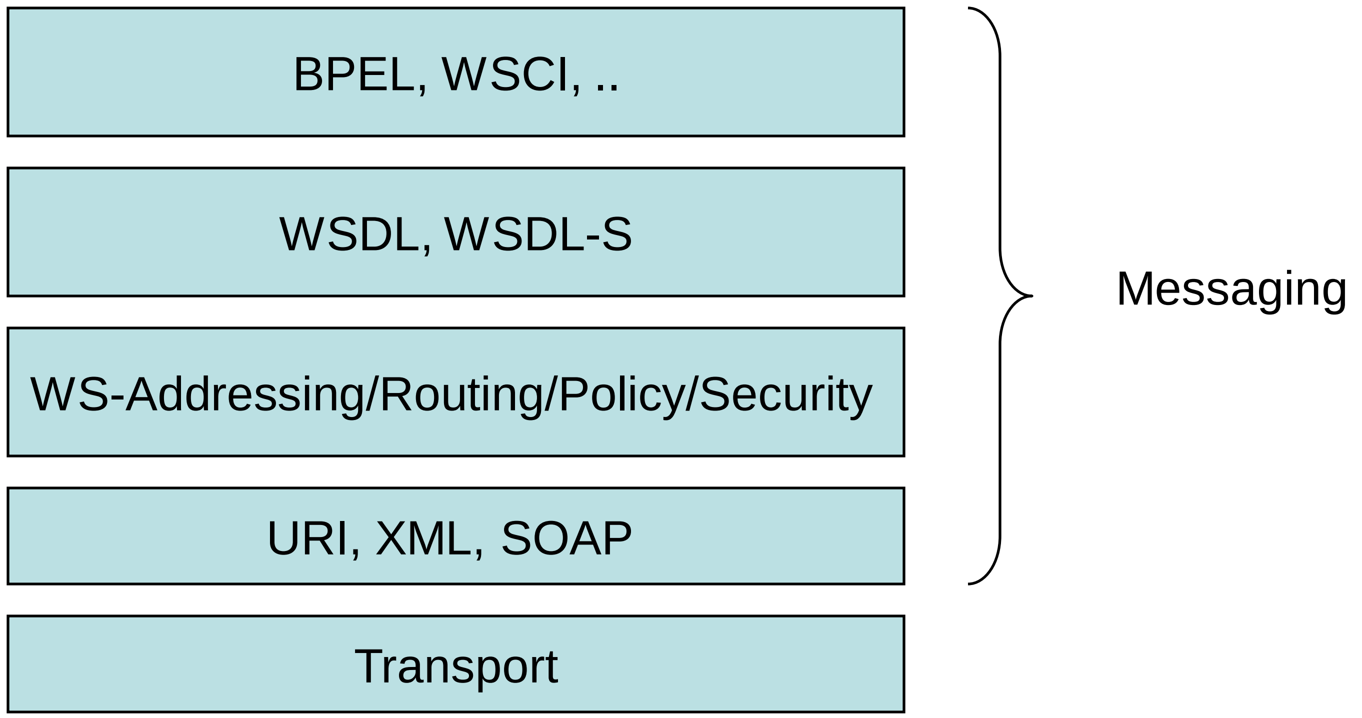
SYSTEMS AND SOFTWARE CONSORTIUM
BUILDING BETTER SOLUTIONS TOGETHER

## SOA Verification

- Still need testing:
  - Functional, performance, interoperability and vulnerability
  - Unit, integration, system, regression, and acceptance
- Need testing at implementation layer (service verification)
  - Service functionality needs to be more robust than ever, because it may be used or attacked in unexpected ways
- Need testing at messaging layer
  - All services must operate as defined by interface - Web Services Definition Language (WSDL)
  - Semantics needed too (WSDL-S)
  - Example of semantics issues
    - Excel spreadsheet example from Genetics Community
  - Need verification of metadata

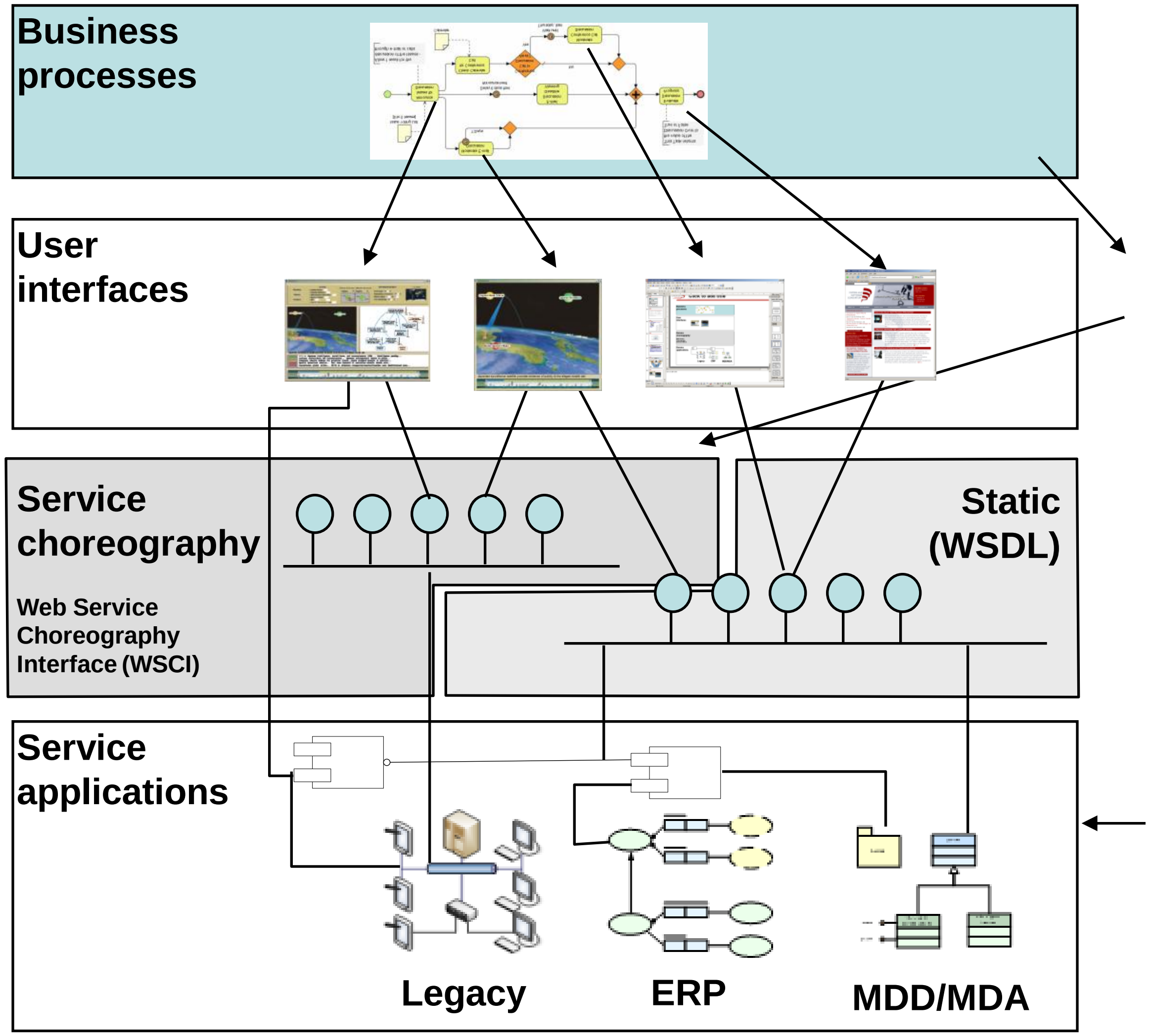
# Example: SOA Messaging Elements

- Verification must consider all messaging elements of metadata



SOAP: Simple Object Access Protocol
BPEL: Business Process Execution Language
WSCI: Web Service Choreography Interface
WSDL-S: Web Service Semantics

# Enterprise-wide SOA From Applications Perspective



Composite services orchestration defined with Business Process Modeling Notation (BPMN) to generate Business Process Execution Language (BPEL)

WSCI provides interfaces for exchange of messages, WSDL provides interface, SOAP supports messaging

Services may be commercial, legacy or developed using more modern approaches such as Model Driven Development (MDD) focused generation of code for the service applications

SYSTEMS AND SOFTWARE CONSORTIUM
BUILDING BETTER SOLUTIONS TOGETHER

## Other Verification Issues

- Lack of control
  - Services run on an independent infrastructure and evolve under control of provider
  - Is regression testing comprehensive and feasible for every version change?
- Verification of long sequences of asynchronous events is difficult testing task
  - Continuous self-checking of a services/system by monitoring it during execution
    - Alternative to testing
  - Monitoring impacts the design process
    - Requires verification to ensure monitoring works properly
    - More unanticipated cost

SYSTEMS AND SOFTWARE CONSORTIUM
BUILDING BETTER SOLUTIONS TOGETHER

## Other Verification Issues (cont.)

- Lack of observability
  - Loose coupling is a good thing, but introduces testing challenges
    - Logging services may support testability
    - Be a discriminator in selecting a service to use
  - Web services client developers typically have access to interfaces (WSDLs) only
    - No access to the code or structure
- Dynamic behavior
  - Service changes or service is no longer available
  - From earlier example:
    - Choosing between warehouses and manufacturers
    - When/if a shipping number is generated and/or PO error is sent
    - Potential need to provide mechanisms to substitute alternatives for unavailable product
    - Creates need for more error handling => more effort and cost

SYSTEMS AND SOFTWARE CONSORTIUM
BUILDING BETTER SOLUTIONS TOGETHER

## SOA Security

- Distributed nature of SOA applications adds complexity
  - Traditional applications often use single point for identification and authentication
  - SOA applications may require users to be identified and authenticated to multiple servers during a transaction
- No over-riding security context for composite services
  - Services must determine that call to it is from an authenticated user, with authority to perform the action
  - SOA applications may require sophisticated identity management and security policy infrastructure
- Metadata – what information is provided to an attacker?
  - If a service can find another service, an attacker can too
  - What happens if discovered services is established by an attacker? And, you send your data to it?
- Security considerations at each interoperable interface

# Example Requirements for Security*

- Security functionality must be considered at every operation where there is an interface boundary

| *Sender → Receiver* | *Operation* | *Message* | *Message Integrity* | *Authenti-cation* | *Confident-iality* | *Algorithm* |
|---|---|---|---|---|---|---|
| *Web Client → Retailer* | *getCatalog* | *getCatalog Request* | *WC X.509: Body, UNT, Timestamp* | *UNT-user, Cert Auth* | *R X.509: Body, Signature* | *Key: RSA 1.5, Data: AES 128, Digest: SHA1* |
| *Retailer → Web Client* | *getCatalog* | *getCatalog Response* | *R X.509: Body, Timestamp* | *Cert Auth* | *WC X.509: Body, Signature* | *Key: RSA 1.5, Data: AES 128, Digest: SHA1* |
| *Web Client → Retailer* | *submitOrder* | *submitOrder Request* | *WC X.509: Body, UNT, Timestamp* | *UNT-user, Cert Auth* | *R X.509: Body, Signature* | *Key: RSA 1.5, Data: AES 128, Digest: SHA1* |
| *Retailer → Web Client* | *submitOrder* | *submitOrder Response* | *R X.509: Body, Timestamp* | *Cert Auth* | *WC X.509: Body, Signature* | *Key: RSA 1.5, Data: AES 128, Digest: SHA1* |
| *Retailer → Warehouse n* | *ShipGoods* | *ShipGoods Request* | *R X.509: Body, Config Header, Timestamp* | *Cert Auth* | *None* | *Key: RSA 1.5, Digest: SHA1* |
| *Warehouse n → Retailer* | *ShipGoods* | *ShipGoods Response* | *Wn X.509: Body, Timestamp* | *Cert Auth* | *None* | *Key: RSA 1.5, Digest: SHA1* |

*SCM Security Architecture WGD 5-00 (March, 2006)

## Security Features versus SW Defects

- Security features such as authentication, encryption, access control, etc. are necessary but not sufficient
  - Hardware appliances for networks security only “filter” - also necessary but not sufficient
- Security often breached by exploiting vulnerabilities
  - Defects (or weaknesses) in design or implementation often make the system vulnerable
  - Example SW defect: distributed error handling
    - E.g., errorPO – the asynchronous error handler

## Opportunities for Re-Design

- SOA projects will likely require redesign to support reuse
  - Such activities provide an opportunity to address security architecture as well as implementation details
- Organization that expose a component developed for use in tightly coupled environment as a service need to apply rigorous engineering for security as well as robustness
- Analyze service connections and interfaces
  - What can call what?
  - What are the vulnerabilities by the interactions?
  - What possible multi-state transactions can be used to break security?
  - What types of API are used? How are they vulnerable?

SYSTEMS AND SOFTWARE CONSORTIUM
*BUILDING BETTER SOLUTIONS TOGETHER*

## Design for Testability

- Requires element under test to have:
  - Controllability, observability, and predictablity
- Fundamental to automation
  - Well-defined interfaces and program-to-program interaction facilitates test automation
  - Automation supports rapid continual deployment, with reduced cost
- Supports automated regression testing that is even more important as services may change or go away
- SOA are not always predictable
  - Any distributed system with asynchronous communication makes the predictability of the systems more difficult

## Test at Multiple Levels

- Increase test coverage with fewer tests
- Test service implementation at the interfaces
  - Can be assured that implementation behavior is correct
  - Can be accomplished with predictability in development environment
    - Isolates implementation from communication issues
    - Reduces complexity of a test harness
- Separate business logic from client and server communication and test separately
- Earlier focus on integration testing permits message-based acceptance testing in deployment environment to focus on service-to-service interfaces

## The Right Test Automation

- Testing environment can be a strategic tool for improving implementation efficiency and reducing manual support
  - SOA testing tools generalized from Web page testing tools may be insufficient for SOA implementations – more “manual” than one thinks
  - SOA integration testing tools that simulate service requests and events allow testing in virtual environment
  - SOA verification tools should test dynamically changing business requirements reflected by metadata changes
  - Reuse functional testing assets for performance and load testing
- Model-based testing helps ensure robustness for service implementation and supports regression testing
  - Model-based testing promotes test driven development
  - Reuse of models provides high ROI
- Hard part is the distributed processing aspects of SOA

SYSTEMS AND SOFTWARE CONSORTIUM
BUILDING BETTER SOLUTIONS TOGETHER

## Conclusion – Paradox?

- What’s good?
  - Loosely coupled, reuseable, discoverable, composite services
- What’s bad?
  - Must be more robust
  - Highly distributed which are harder to verify
  - Requires additional effort to provide security at every interface
- Recommendations - leverage re-design opportunities:
  - Address security at every interface
  - Design for testability to support continuous automated testing
  - Separate business logic, from implementation service and communication
    - Supports layered testing to increase coverage with fewer tests
  - Address distributed process verification with alternatives such as monitoring to supplement testing