



Open Source SOA

Dr Mark Little
CTO JBoss, a division of Red Hat

Overview

- **SOA in a nutshell**
 - Degrees of coupling
 - The component triad
- **Relationship to WS-***
- **Open Source approach and ESBs**
 - Registries and repositories
 - Message delivery and transformation
 - Service orchestration
- **Futures**
 - Testable architectures
 - Transaction processing in a SOA

What is SOA?

- *An SOA is a specific type of distributed system in which the agents are "services"*
(<http://www.w3.org/TR/2003/WD-ws-arch-20030808/#id2617708>)
- Adopting SOA is essential to delivering the business agility and IT flexibility promised by Web Services.
- But SOA is not a technology and does not come in a shrink-wrapped box
 - **It takes a different development methodology**
 - **It's not about exposing individual objects on the "bus"**

Tightly coupled

- A distributed application consists of several distinct components
- Traditional client and server technologies based on RPC
 - **Hide distribution**
 - **Make remote service invocation look the same as local component invocation**
- Unfortunately this *tightly coupled* applications
 - **Such applications can be brittle**

Loosely coupled

- SOA is an architectural style to achieve *loose coupling*
 - **A service is a unit of work done by a service provider to achieve desired end results for a consumer.**
- SOA is deliberately not prescriptive about what happens behind service endpoints
 - **We are only concerned with the transfer of structured data between parties**
- SOA turns business functions into services that can be reused and accessed through standard interfaces.
 - **Should be accessible through different applications over a variety of channels.**

But ...

- **There are degrees of coupling and you should choose the level that is right for you**
- **At the one extreme**
 - Defining specific service interfaces, akin to IDL
 - Easier to reason about the service
 - Limits the amount of freedom in changing the implementation
- **At the other extreme**
 - Single operation (e.g., doWork)
 - More flexibility in changing the implementation
 - Well, almost ...
 - More difficult to determine service functionality a priori
 - Need more service metadata

What about Web Services?

- **Popular integration approach**
 - XML
 - HTTP
 - Pretty much universal acceptance (see bullets above!)
- **Not specific to SOA**
 - Web Services began life as CORBA-over-HTTP
 - XML-RPC
- **Web Services+SOA gives benefits**
 - Loose coupling
 - Interoperability
 - Enterprise capabilities, e.g., security and transactions

Fortunately ...

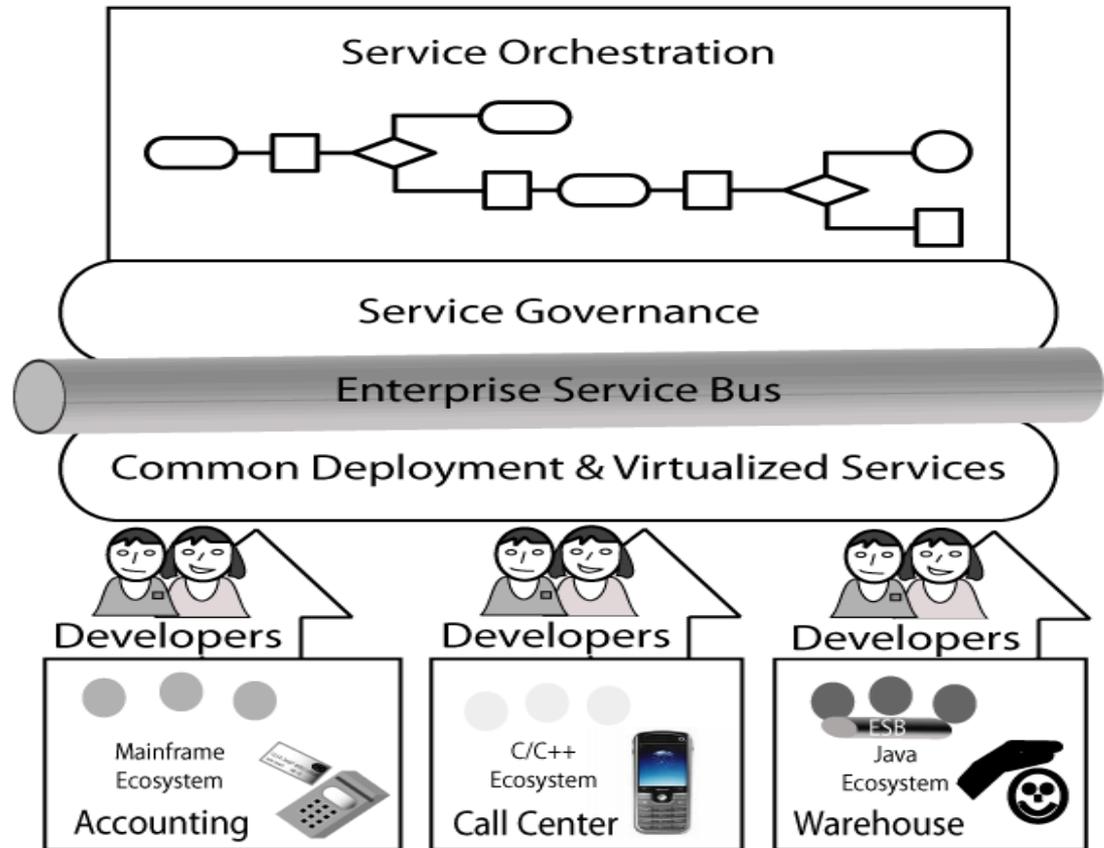
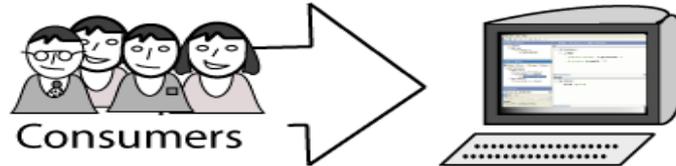
- **SOA is technology agnostic**
- **WS-* offers the potential for interoperable SOA**
- **But it is just as easy to develop closely-coupled applications in WS-***
- **Most vendor WS-* tools are direct mappings of distributed object tools**
 - SOA != distributed objects with angle brackets
- **A SOA infrastructure should support and encourage SOA principles**
 - Sometimes it is easier said than done

What is an ESB?

- **Depends who you ask!**
 - Could be JMS with MDBs
 - Could be Web Services platform
 - Moving towards SOA infrastructure
 - Next generation EAI
- **A modern ESB != an ESB from 5 years ago**

ESB and SOI

Composite & Rich Internet Applications



What should you expect?

- **SOA message-based architecture**
- **Various transports**
 - (S)FTP(S), email, HTTP(S), JMS, file, database, InVM, ...
- **Leveraging mature technology bases**
- **Task orchestration**
 - e.g., WS-BPEL
- **Web Services**
 - “Zero” coding
- **Adapters**
 - EJB(3), Spring, CICS, JCA, ...
- **Transformations**
 - XSLT
 - Support for large messages
- **CBR, transactions, clustering, registry/repository**
- **Performance, reliability, ...**

Repository is critical

- **Service metadata, which is important for contract definitions**
 - Functional and non-functional aspects
 - Transactional, secure, QoS, ...
 - Policies
 - MEPs
 - One-way
 - Request-response
 - Message structure
 - Where data resides
 - Governance
- **Service binaries**
- **Business rules**
- **Workflow tasks or process control information**

Services and messages

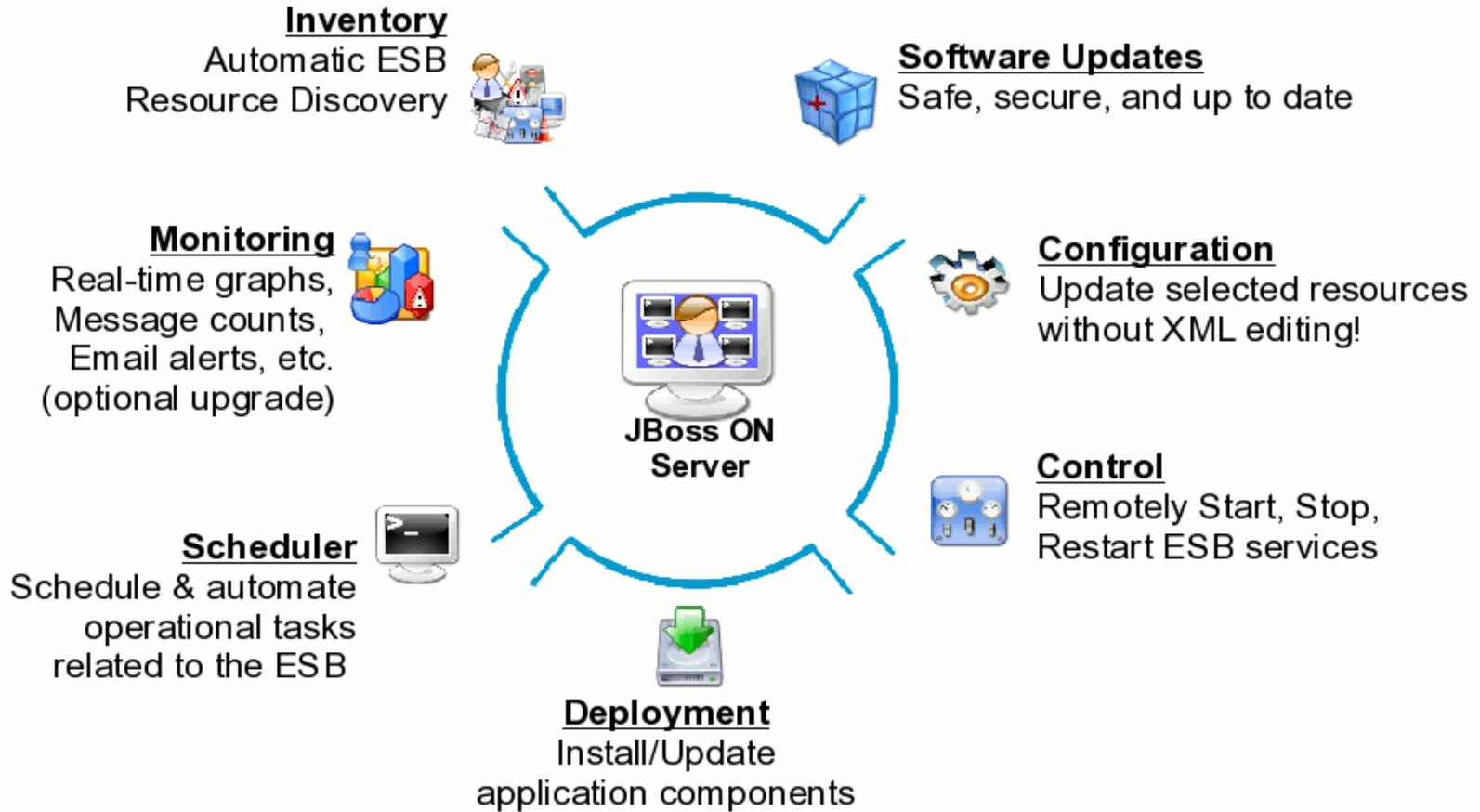
- **All services are interacted with via messages**
 - Messages are part of the contract between client and service
- **Messages do not imply specific implementations of carrier-protocol**
- **Services do not need to be bound to specific implementations of carrier-protocol**
 - Email, S-FTP, JMS, File, etc.
 - More can be added as required

Standards based message delivery

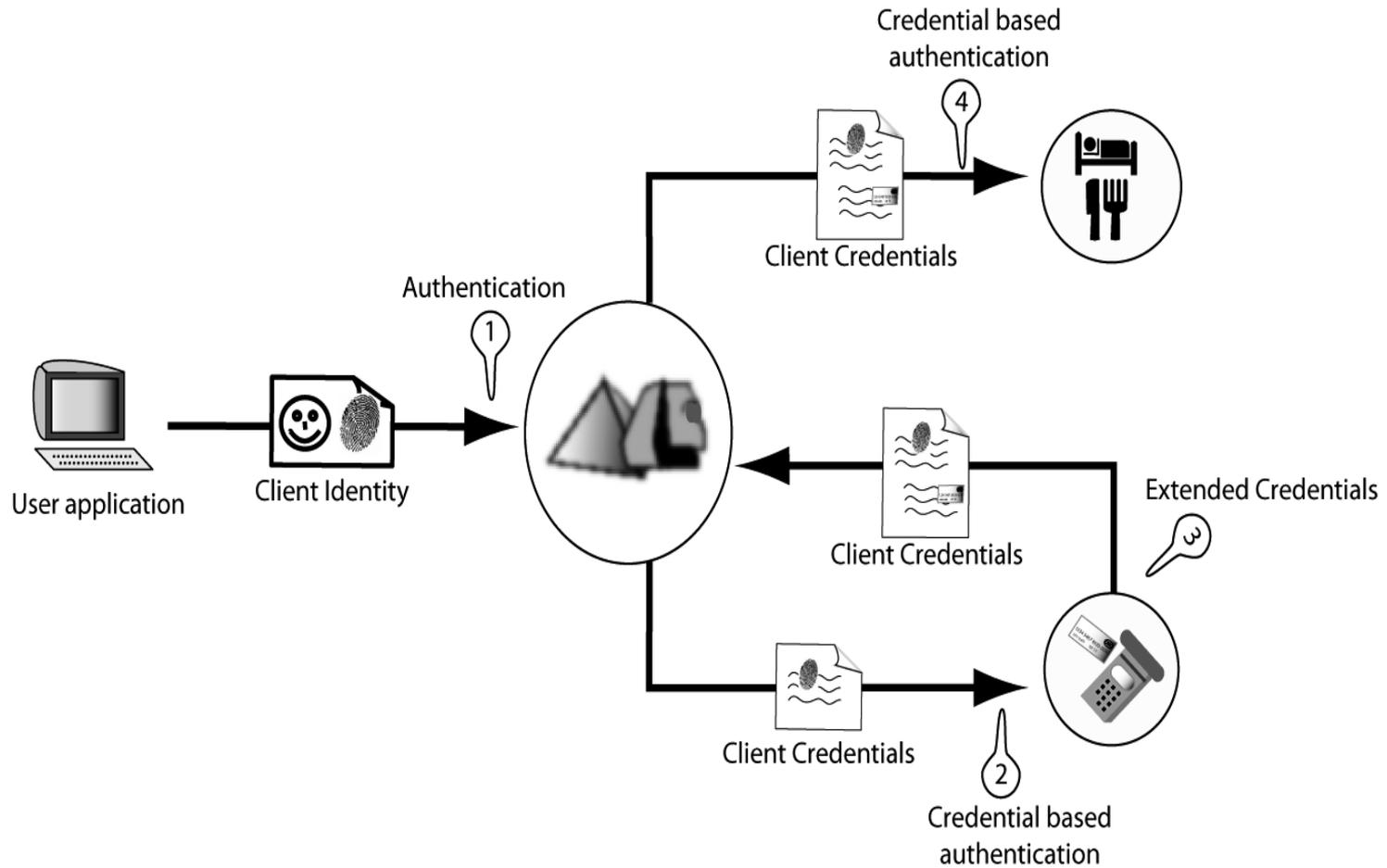
- **Addressed via WS-Addressing Endpoint References**
 - Transport agnostic
- **Supports request-response as well as one-way MEP**
- **Mandatory to define the recipient address**
- **Optional**
 - Reply address
 - Message relationship information
 - Fault address

Governance

Simplifying Middleware Management for IT Administrators



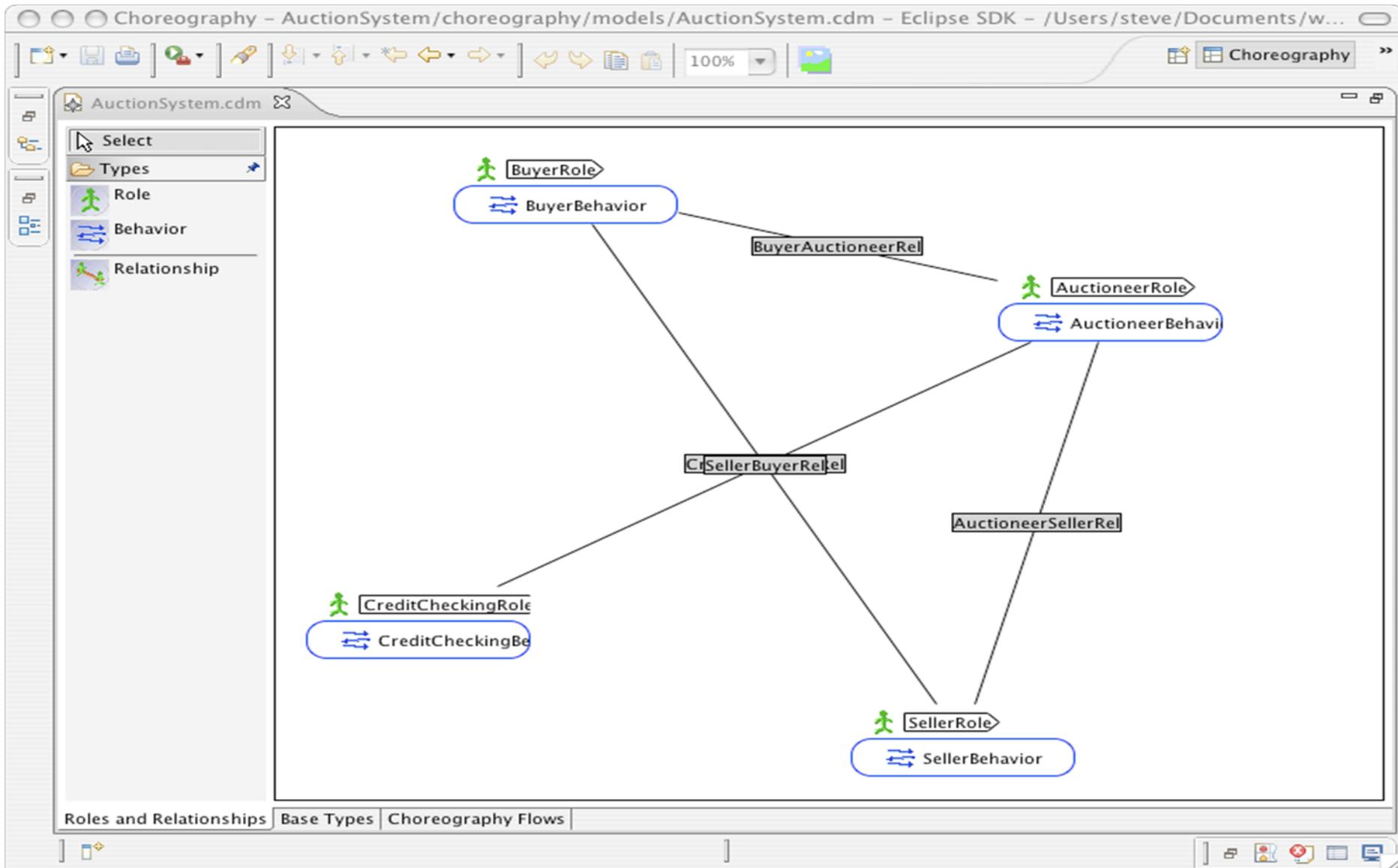
Identity management



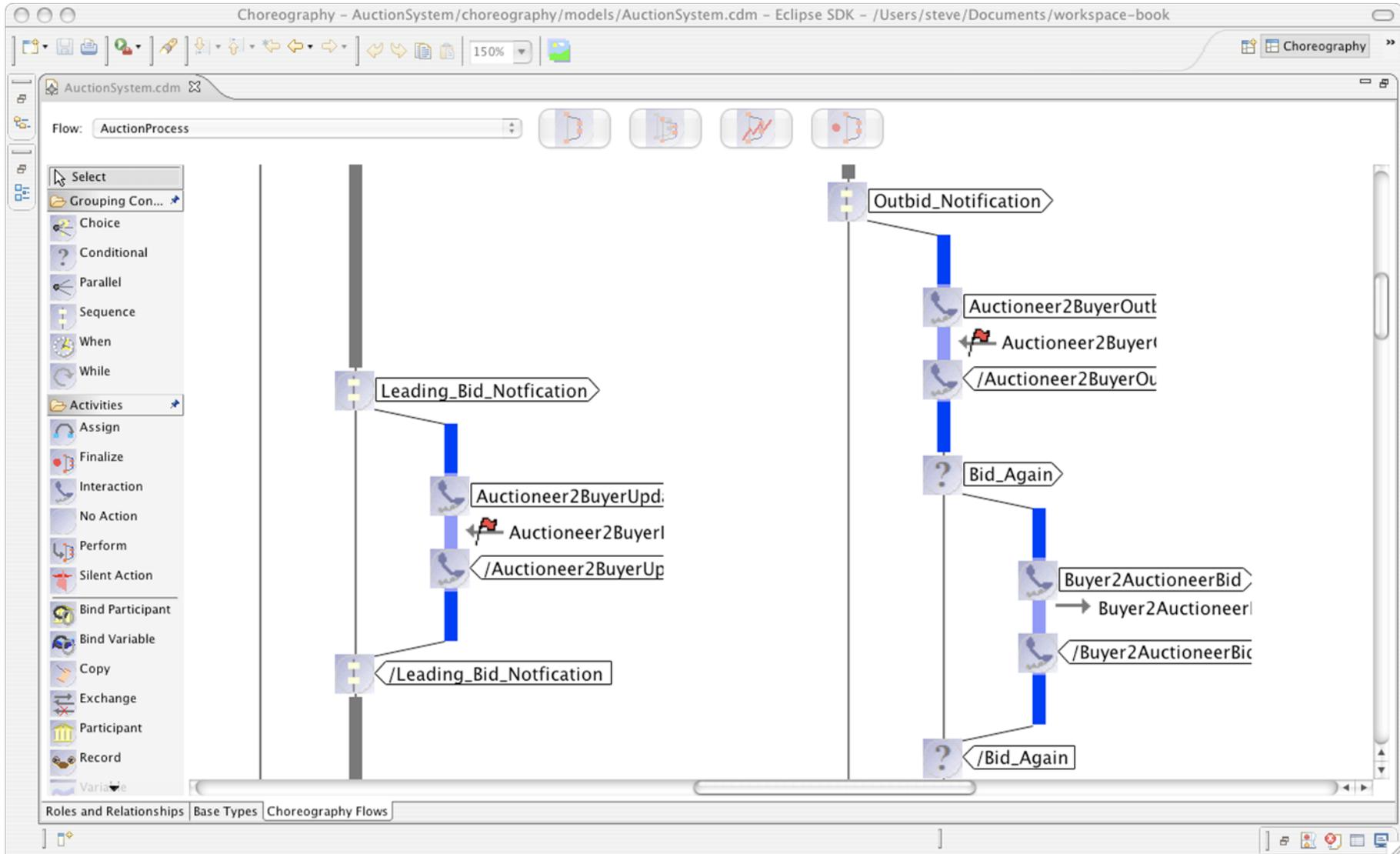
Testable architectures

- **Choreography description language**
 - NOT a competitor to BPEL
 - Compliments orchestrations
- **Provable correct distributed systems**
 - Design-time as well as run-time
 - Should be part of any good governance strategy
- **Not Web Services specific**
 - Ideal for SOA

Roles and relationships



SOA Blueprint Modeler



But where does it fit?

- **WS-CDL is a key technology for successful SOA development**
 - Most deployments have more than 2 participants!
- **Turns the discipline into engineering**
 - Being able to statically define interactions and simulate them is important
 - Being able to dynamically monitor them and enforce/terminate interactions is a major step
- **Testable architecture is core to SOA success**
 - Tied into SOA Platform
 - Tied into SOA Development Methodology
- **Being adopted by customers and partners**

Fault tolerance

- **Machines and software fail**
 - Fundamental universal law (entropy increases)
 - Things get better with each generation, but still statistically significant
- **Failures of centralized systems difficult to handle**
- **Failures of distributed systems are much more difficult**

Fault tolerance techniques

- **Replication of resources**
 - Increase availability
 - Probability is that a critical number of resources remain operational
 - “Guarantee” forward progress
 - Tolerate programmer errors by heterogeneous implementations
- **Spheres of control**
 - “Guarantee” no partial completion of work in the presence of failures
 - Most popular implementation is ACID transactions

SOA characteristics

- **Business-to-business interactions may be complex**
 - involving many parties
 - spanning many different organisations
 - potentially lasting for hours or days
- **Cannot afford to lock resources on behalf of an individual indefinitely**
- **May need to undo only a subset of work**
- **Need to relax ACID properties**

Transactions for SOA

- **Relax isolation**

- Internal isolation or resources should be a decision for the service provider
 - E.g., commit early and define compensation activities
 - However, it does impact applications
 - Some users may want to know a priori what isolation policies are used
- Undo can be whatever is required

- **Relax atomicity**

- Sometimes it may be desirable to cancel some work without affecting the remainder
 - E.g., prefer to get airline seat now even without travel insurance
- Similar to nested transactions
 - Work performed within scope of a nested transaction is provisional
 - Failure does not affect enclosing transaction

Conclusions

- **SOA is an important design-time and use-time approach**
 - SOA is NOT a product
 - Requires changes to organizational view of software components (services)
- **Web Services are important**
 - Interoperability
 - Internet-scale computing
 - But SOA applications are not inherent in WS-*
- **Open Source (e.g., JBoss SOA-P) can bridge the divide**
 - A single infrastructure that provides SOA support
 - Open!
 - Standards based