



Service Component Architecture

“A reference architecture for SOA”

Cadec 2006

Håkan Dahl, Johan Eltes

Agenda

- The evolution of SOA infrastructure
 - *from SOAP to ESB*
- Service Component Architecture (SCA)
 - *Takes SOA from infrastructure to service modelling*
- SCA in practice
 - *Large-scale SOA within manufacturing*
- Tooling
 - *Demo of high-end SCA tooling (WebSphere Integration Developer)*
- Standardization
 - *Status of SCA standardization efforts*
- Conclusions
 - *Lego re-use and composition has finally arrived!*

The Evolution of SOA infrastructure

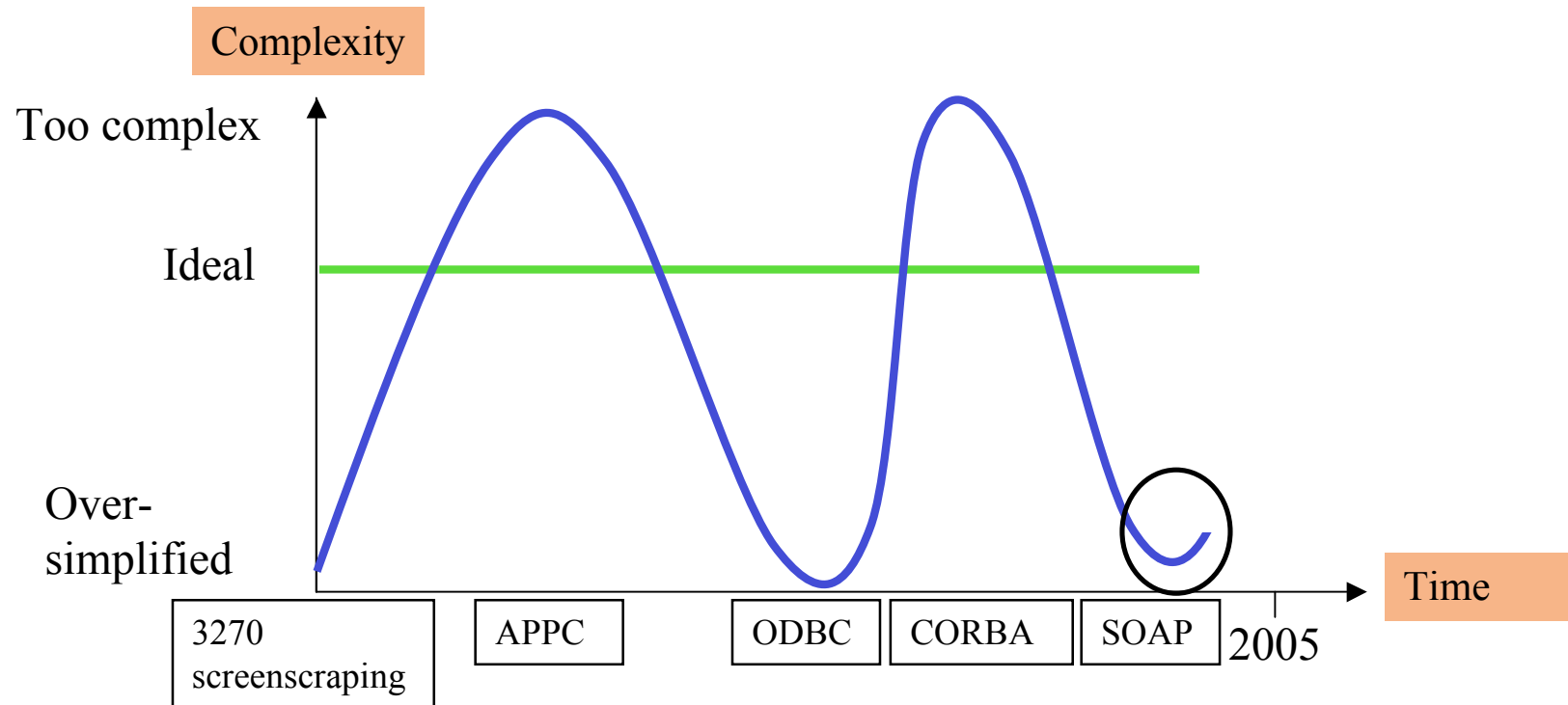
Dry and boring:
Definition of a SOA and Service...

- SOA is an architectural style of loose coupling among interacting software agents
- ...which requires...
 - A small set of simple and ubiquitous interfaces to all participating software agents.
 - The interfaces should be universally available for all providers and consumers.
 - Descriptive messages constrained by an extensible schema delivered through the interfaces
 - An extensible schema allows new versions of services to be introduced without breaking existing services

The Evolution of SOA infrastructure

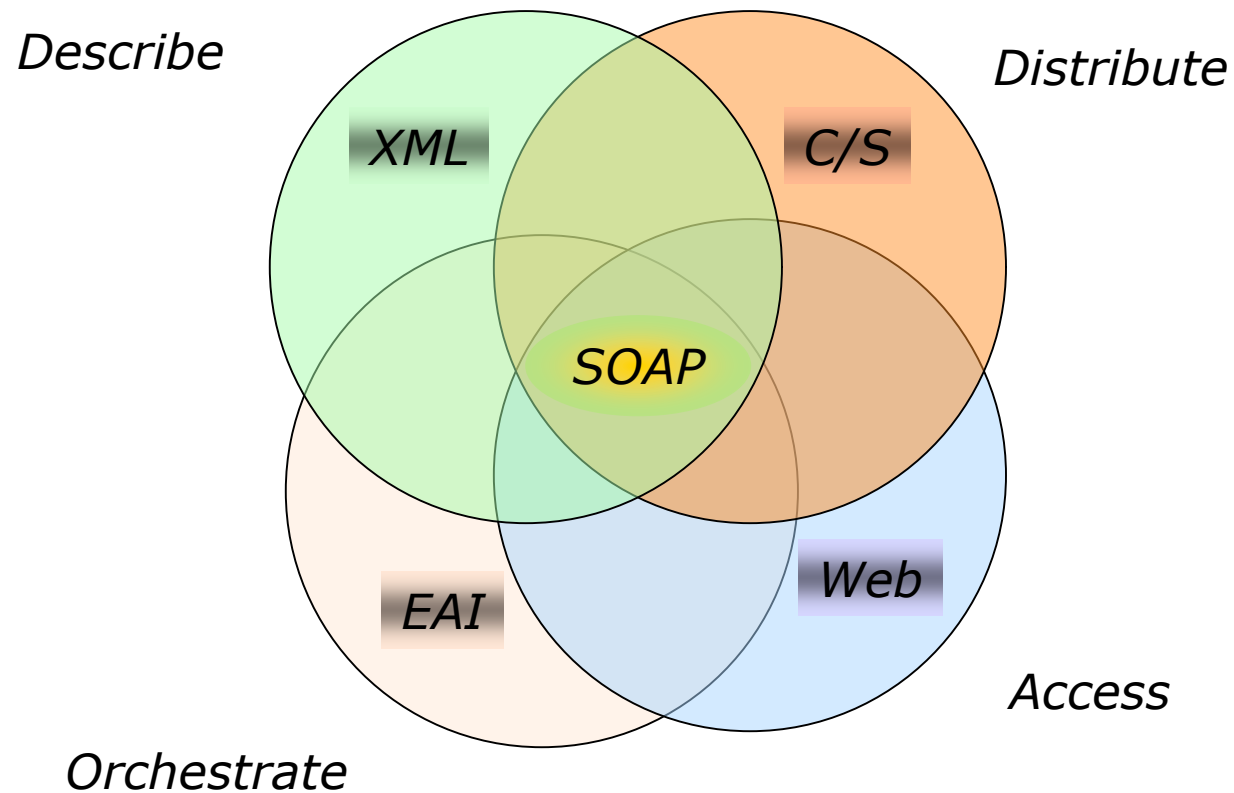
2001: SOA is distributed computing...

Distributed computing...



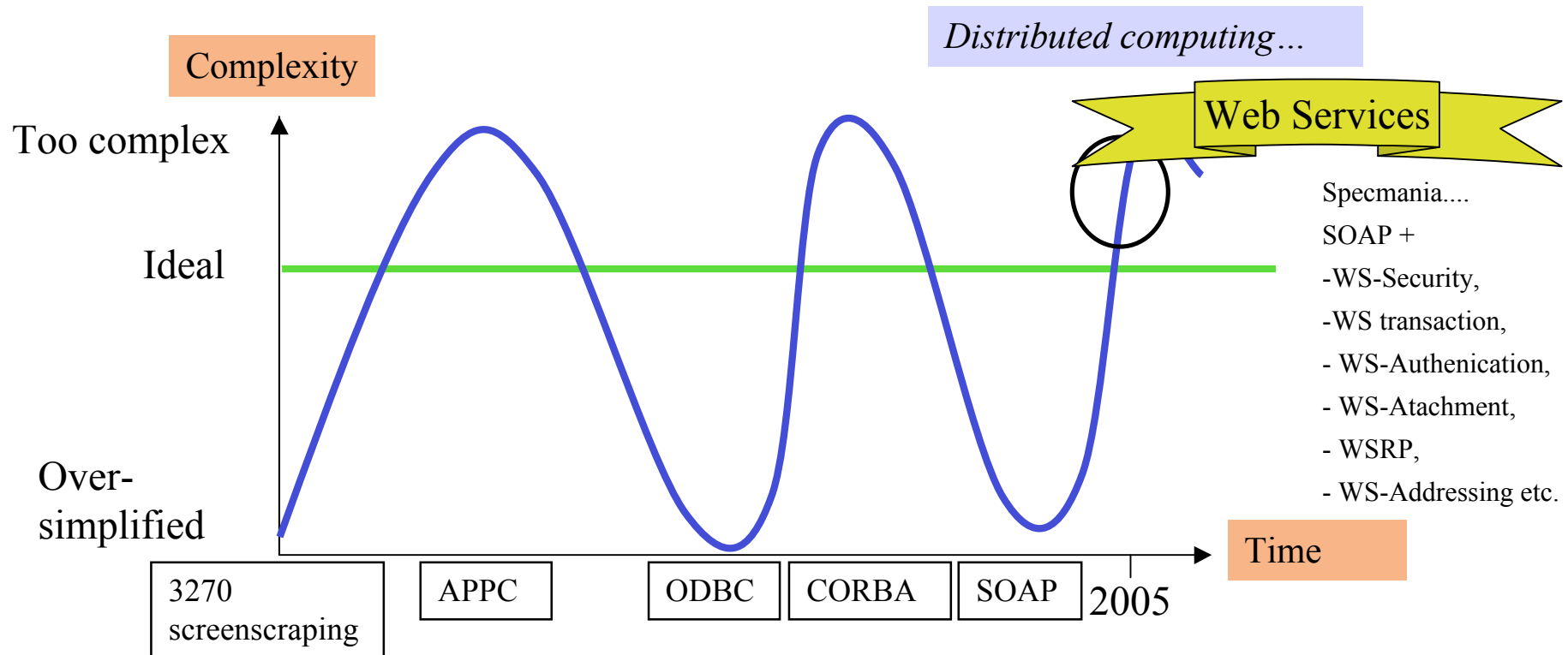
The Evolution of SOA infrastructure

2001: SOAP (Simple Object Access Protocol) - The Holy Grail



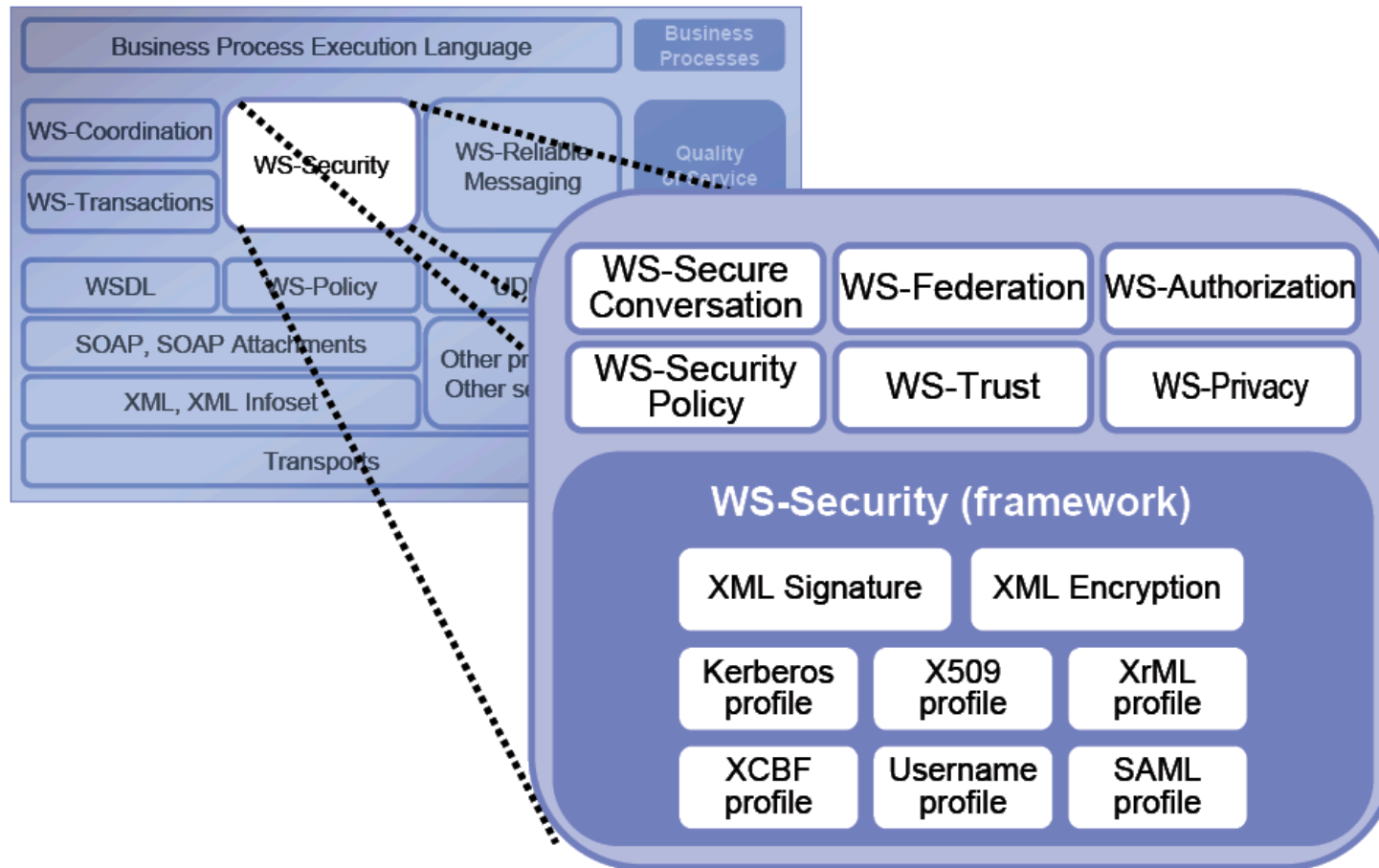
The Evolution of SOA infrastructure

2005: Web Services - Not so simple any more



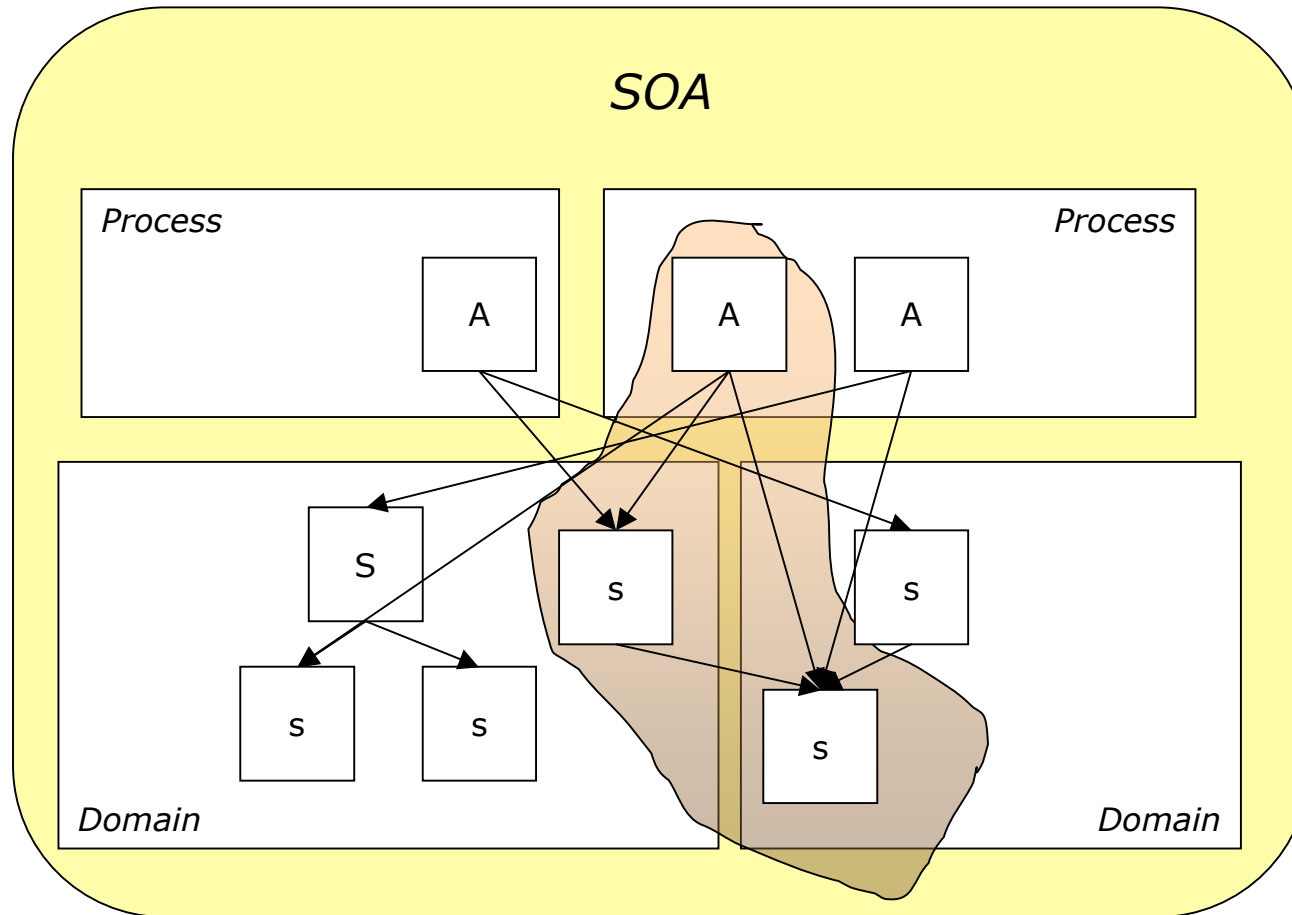
The Evolution of SOA infrastructure

2005: Web Services - Not so simple any more...



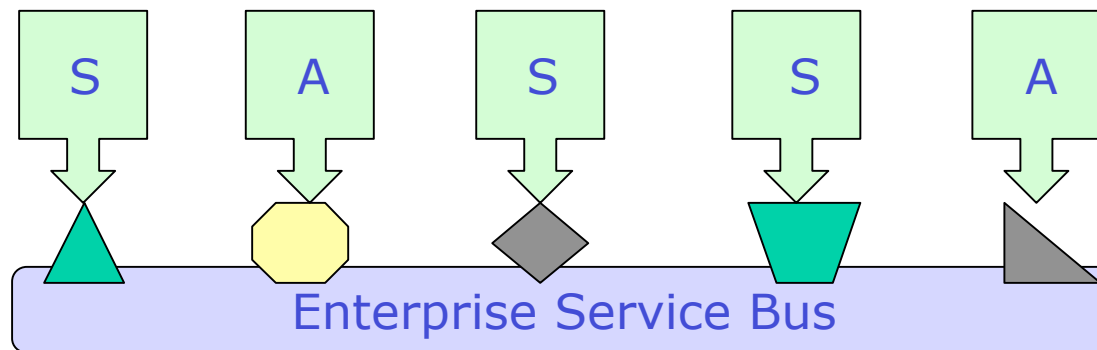
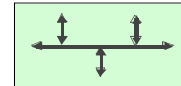
The Evolution of SOA infrastructure

SOA on WebServices - Where would it take us?



The Evolution of SOA infrastructure

The Enterprise Service Bus:
SOA has learned from EAI (hub, mix of new and legacy protocols)

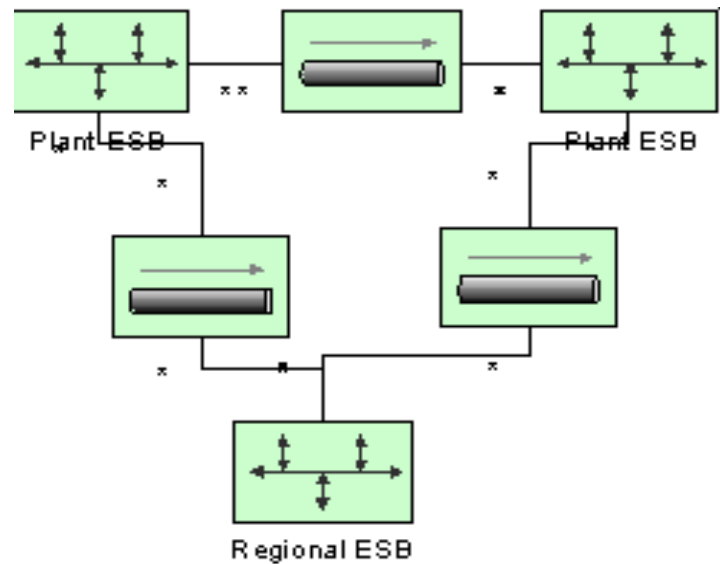


Color = Message Format (different XML schemas, legacy formats...)

Shape = Protocol (FTP, JMS, Native MQSeries, SOAP...)

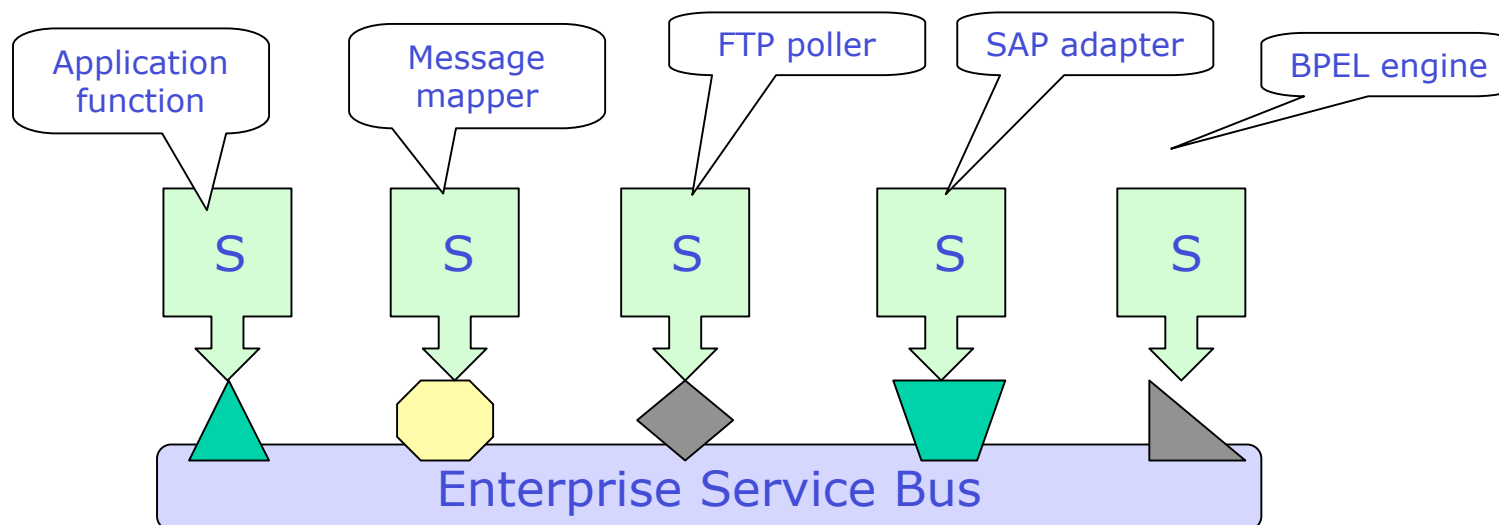
The Evolution of SOA infrastructure

EAI Message Broker = Central Infrastructure
ESB Architecture = Distributed infrastructure



The Evolution of SOA infrastructure

ESB: Everything is a service - business functionality, formatting services, process orchestration...



The Evolution of SOA infrastructure

Service characteristics

- A service participates in a message exchange
 - A service consumes ONE message and may return ONE message
 - A service exports its functionality on a protocol
 - Not necessarily networked
- A service is identified by a infrastructure-dependent endpoint URI
 - With WSDL, A method name is part of the URI
 - With queuing, the queue and eventually a message selector is the endpoint
- A service MAY be aware of quality-of-service contracts
 - Transaction scope
 - Security

The Evolution of SOA infrastructure

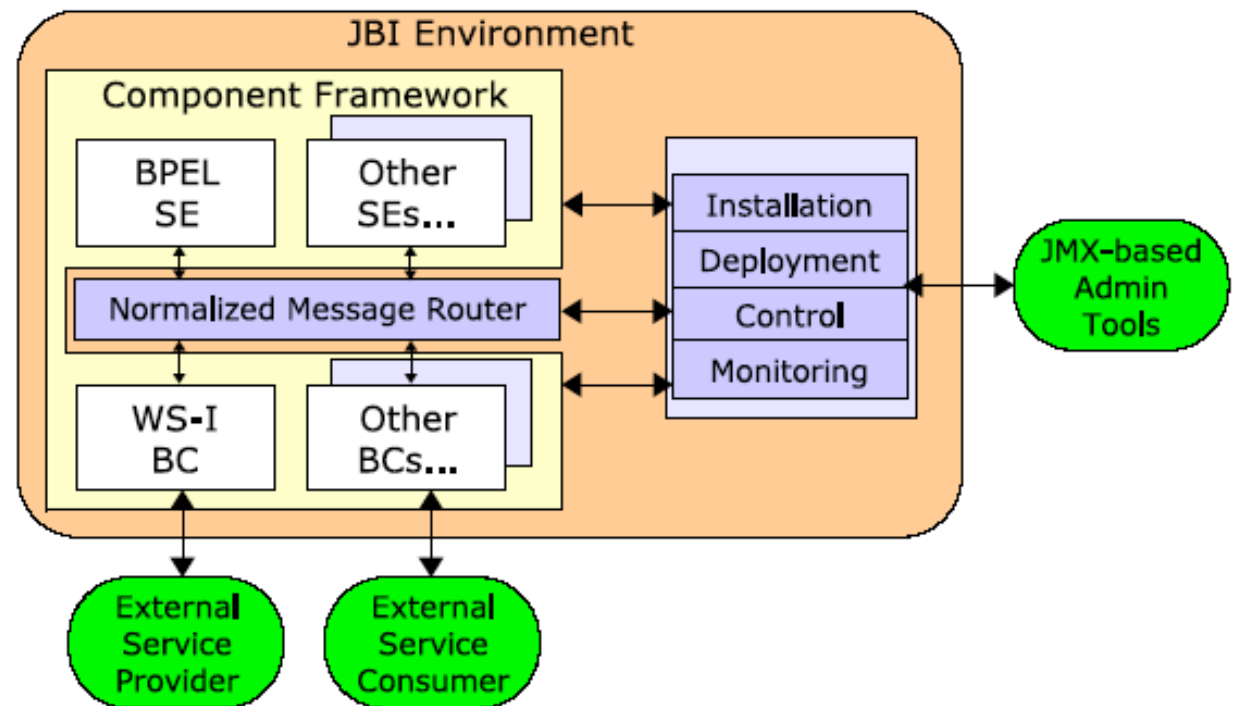
So, how does the new world of SOA look?

- Application development and integration are two aspects of the same thing
 - They depends on the same infrastructure
 - The ESB infrastructure is a container for applications (as services) and integration glue (as services)
- The ESB infrastructure supports transparent, distributed deployment
 - Application services can execute anywhere
 - Integration glue services can execute anywhere
- Same tooling for application development and integration glue
 - Some integration glue requires full-scale Java development
 - Some application services are best realized through code generation from process models (e.g. BPEL)

The Evolution of SOA infrastructure

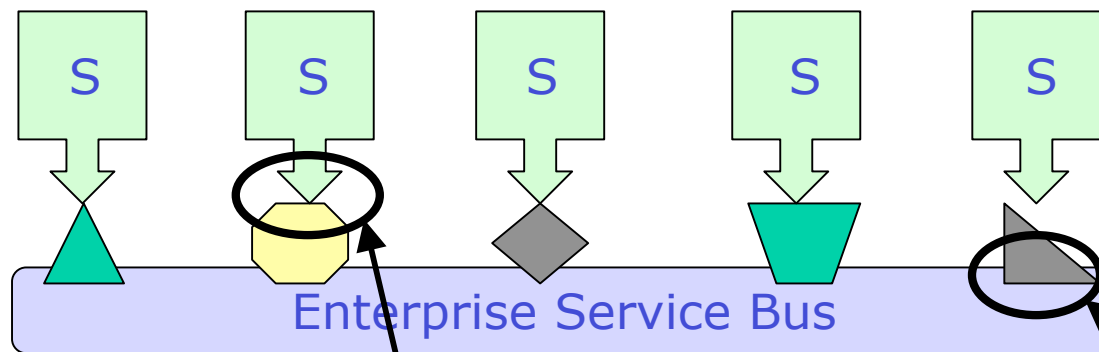
ESB standardization
JBI - Java Business Integration

- JBI defines a container for service implementation containers and protocol bindings
- May replace J2EE as SOA infrastructure standard
- Limited support - All vendors implement an ESB, but differently



The Evolution of SOA infrastructure

ESB - what are we missing?



~~Decommissioning of
middleware~~



Interoperate with new
services through new
protocols



The Evolution of SOA infrastructure

The ESB is here. Where does that leave us?

- Plumbing, plumbing, plumbing...
 - We have tooling and concepts to get interoperability across existing platforms and systems
 - We know the platform and infrastructure landscape will continue to change
- What would the next step be?
 - To model, develop and compose services independent of SOA infrastructure
 - What does it take to make the vision of “software Lego” become a reality?

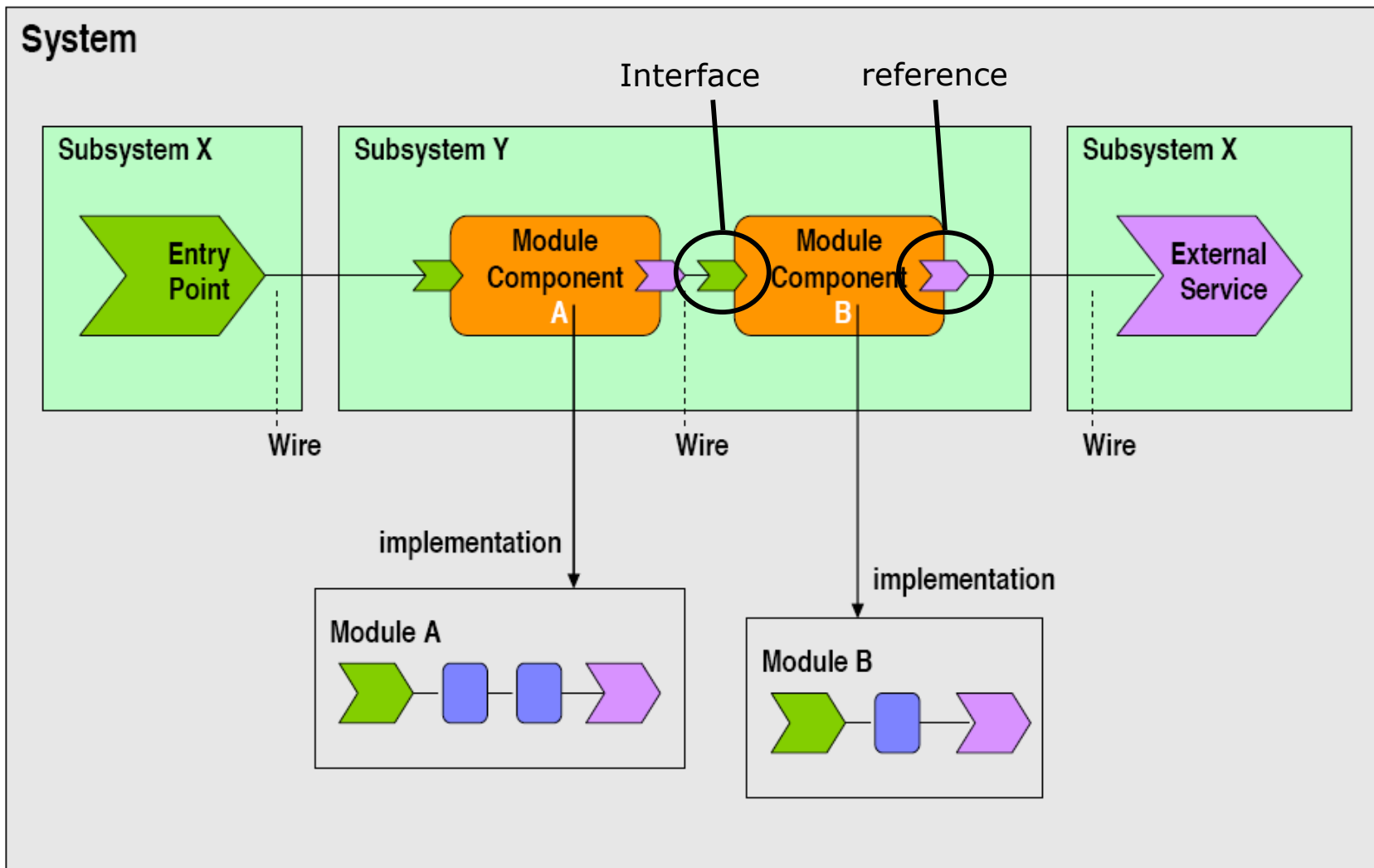
>Service Component Architecture (SCA)

- The evolution of SOA infrastructure
 - *from SOAP to ESB*
- **Service Component Architecture (SCA)**
 - ***Takes SOA from infrastructure to service modelling***
- SCA in practice
 - *Large-scale SOA within manufacturing*
- Tooling
 - *Demo of high-end SCA tooling (WebSphere Integration Developer)*
- Standardization
 - *Status of SCA standardization efforts*
- Conclusions
 - *Lego re-use and composition has finally arrived!*

Service Component Architecture (SCA)

- SCA is a reference architecture for building a SOA, covering new development and integrating existing applications
- Architectural style of recursive composition of services out of services into a SOA (-system)
- ...without building any assumptions of protocols or deployment topology into the service implementations
- Dependency Injection is the implied mechanism for bridging the gap between SCA services and SOA infrastructure

SCA Core concepts



> SCA in practice

- The evolution of SOA infrastructure
 - *from SOAP to ESB*
- Service Component Architecture (SCA)
 - *Takes SOA from infrastructure to service modelling*
- **SCA in practice**
 - ***Large-scale SOA within manufacturing***
- Tooling
 - *Demo of high-end SCA tooling (WebSphere Integration Developer)*
- Standardization
 - *Status of SCA standardization efforts*
- Conclusions
 - *Lego re-use and composition has finally arrived*

Large-scale SCA within automotive

- Mission
 - Replace a large number of plant-specific systems (20 plants) with a common (global) portfolio of systems
- Business case
 - Disarm the legacy bomb
 - Reduce cost of maintenance and operations
 - Improve business agility and business process support
- Business areas
 - Part manufacturing
 - Vehicle Assembly
- System domains
 - Material ordering
 - Quality control
 - Material Distribution
 - **Customer order slotting**
 - **Production control**
 - **Production planning**
 - **Device communication**

Integration
of Packaged
Solutions

New
development



Architectural non-functional goals

- ❑ Minimize impact on plant operations
- ❑ Avoid redundant implementation of business logic
- ❑ SOA without trading performance
- ❑ Investment in business logic must sustain 20 years of technology evolution
- ❑ Quality (automated server-side build, test, integration, deploy)
- ❑ Support light-weight (low-cost) deployment (20 plants)
 - Support deployment virtualization (run multiple plants in a single deployment)

Minimize impact on plant operations

The system architecture need to be comprised of small, asynchronously integrated services. Only services that depends on a new feature should need to be stopped and re-installed. Xml schema upgrades must be possible without upgrading every consumer and provider of a service built on previous version of a schema.

□ SCA contributions

- Support for asynchronous service invocations (business events)
- Separation of service implementation and service modules
- Mediation service module may be introduced to resolve versioning issues

□ Gaps

- Versioning strategy for backwards / forwards compatibility of XML schemas / service interfaces.
- Binary versioning of service modules
 - Dependency management of versioned modules

Avoid redundant implementation of business logic

Each piece of business logic should only be coded once.

- SCA contribution
 - Remote / service deployment not required for re-use
 - A component can be wired into multiple modules
 - A module can be wired into multiple subsystems
- Gaps
 - None identified

SOA without trading performance

The goals of a SOA need to be achieved without depending on networked deployment of individual services.

□ SCA contribution

- Dependency injection allows service modules to be deployed “in process” within each consuming subsystem, instead of independent networked agents.

□ Gaps

- None identified

Investment in business logic should sustain 20 years of technology evolution

Protocols, technologies, middleware has and will change over time. Minimal platform footprint: J2SE (->Java SE). Unlike .Net, J2SE has a proven track record of platform stability (backwards compatibility). Not a guarantee, but we need some platform.

□ SCA contribution

- Infrastructure abstractions through Dependency injection
- Standard simplifies container/vendor migration
- SCA itself *can* have zero footprint in business logic

□ Gaps

- SCA introduces APIs that may be tempting to use, which creates binding to SCA itself

Quality

Upgrade deployment in with minimal impact on shop floor activities. High quality essential. Even with global maintenance team (change code you didn't code your self)

□ SCA contribution

- Dependency injection makes component-, module- and subsystem test scripting efficient
- Lightweight infrastructure can be “injected” to minimize time for running tests (1000:ths of tests are run for each build)

□ Gaps

- Excuses need to be sought elsewhere...

Applied SCA best-practice

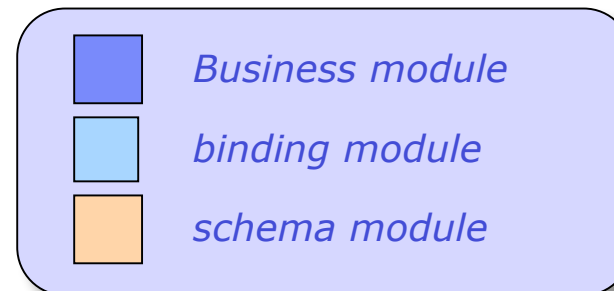
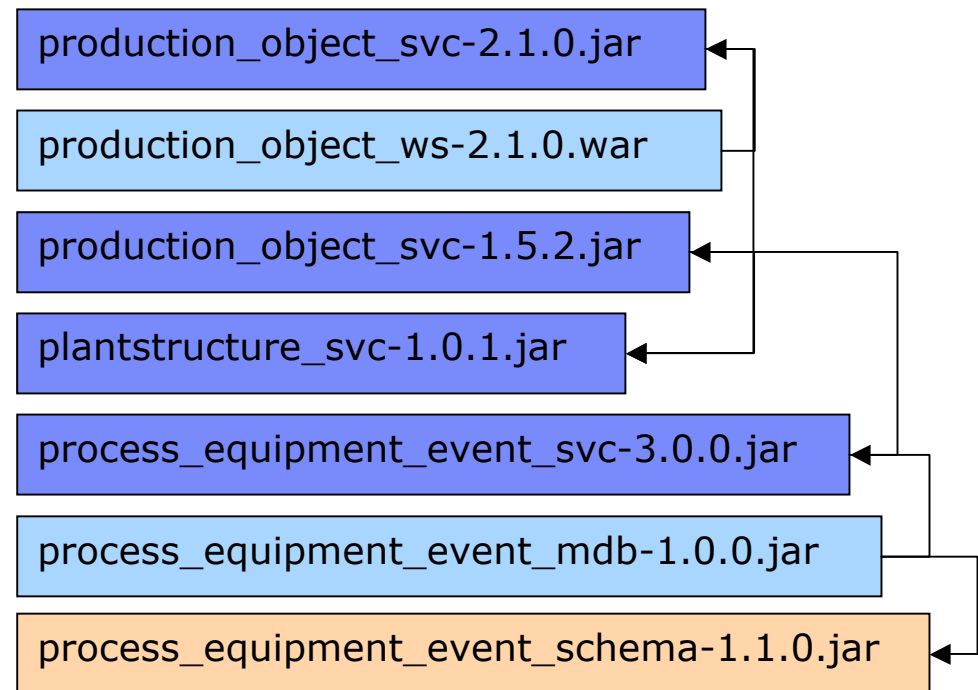
- ❑ Corporate module repository
- ❑ Business modules internally structured / wired in layers
- ❑ Modules within a subsystem are wired at build-time
- ❑ Subsystems (SOA services) within an SCA system are wired at- or after deploy-time via a system-internal ESB mediator
- ❑ SCA systems are wired at- or after deploy time by connecting system-internal ESB:s
- ❑ Integration object model ("business objects) represented as XML schemas, re-used in multiple WSDL files through "import"

Module repository

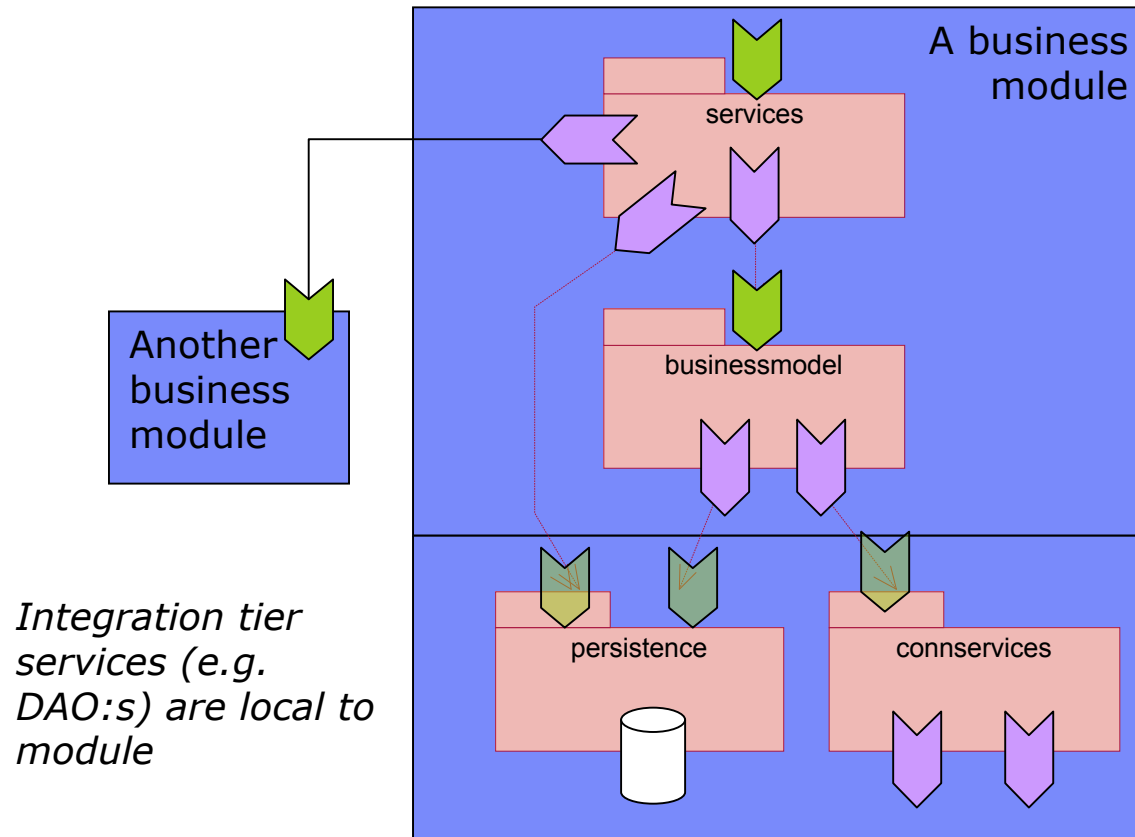
- Corporate repository of versioned modules (jar files)
 - Unit of re-use and version base-lining
- Three types of modules:
 - Business modules, binding modules and schema modules
- Business modules
 - Business logic
 - Jar files with spring config files
- Binding modules
 - Protocol binding (WS, JMS etc), data binding (message payload -> java classes) and service activation (invoke referenced services in various business modules)
 - WAR archives (for WS binding) and EJB-JAR archives (MDB:s for JMS binding)
- Schema modules
 - XML Schemas defining business objects
 - Generated binding classes (we use JAXB - not SDO)
 - (WID: "Library module")

Module Repository example

- The Maven build system is used to manage a repository of versioned binary modules of including runtime dependencies between modules
- Module names are unique across the business



Business module composition (Layered model)



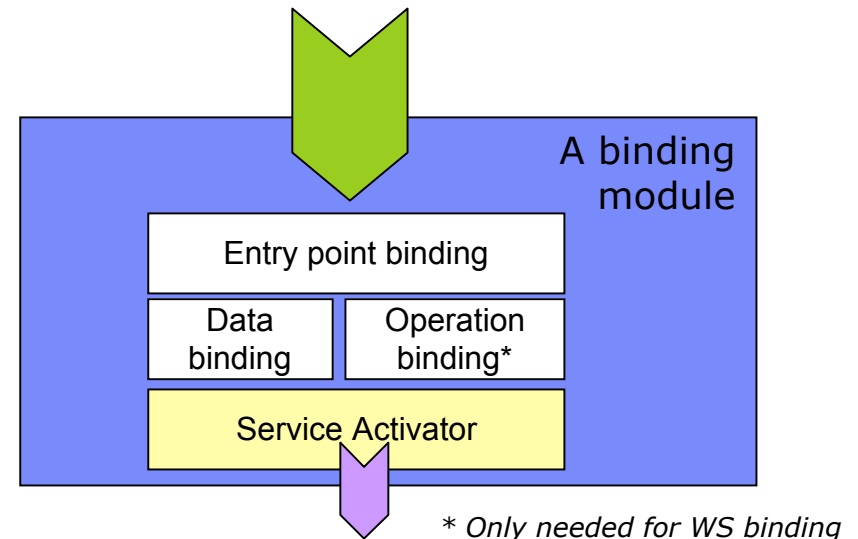
Dependency injection (wiring) is conducted by the Spring framework.

Simplified migration to SCA spec, by standardizing on map able spring features.

Integration tier services (e.g. DAO:s) are local to module

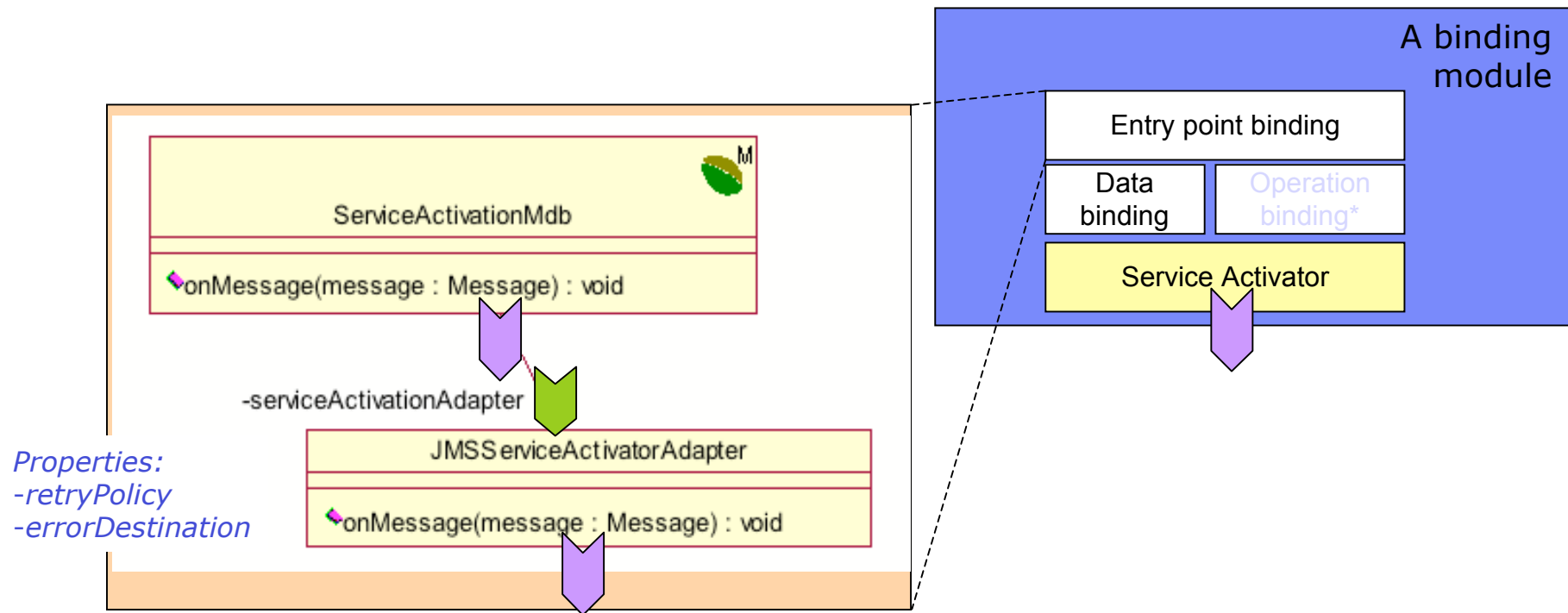
Binding module composition

- Publishes a service to the ESB
 - Explicitly defines a networked coarse-grained service (in the sense of SOA)
- Wiring of service components:
 - Entry point binding (generic component per protocol)
 - (Operation binding: WSDL -> Java stub)
 - Data binding (generic component for JAXB)
 - Service activator (custom component)
- Binding module
 - Listens to an ESB protocol
 - Conducts data binding
 - Invokes service activator
- Archive depends on type of binding
 - WS binding: WAR
 - JMS binding: EJB-JAR with MDB:s only



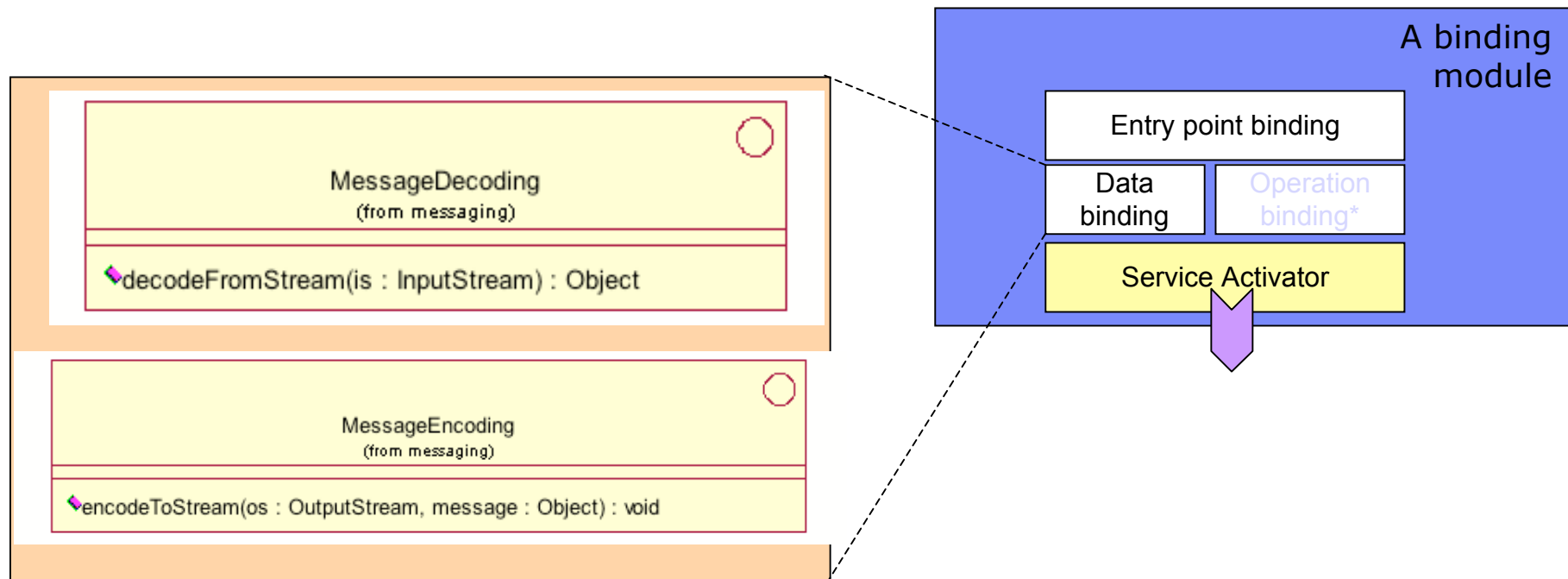
Binding module example - JMS entry point

□ Generic JMS binding



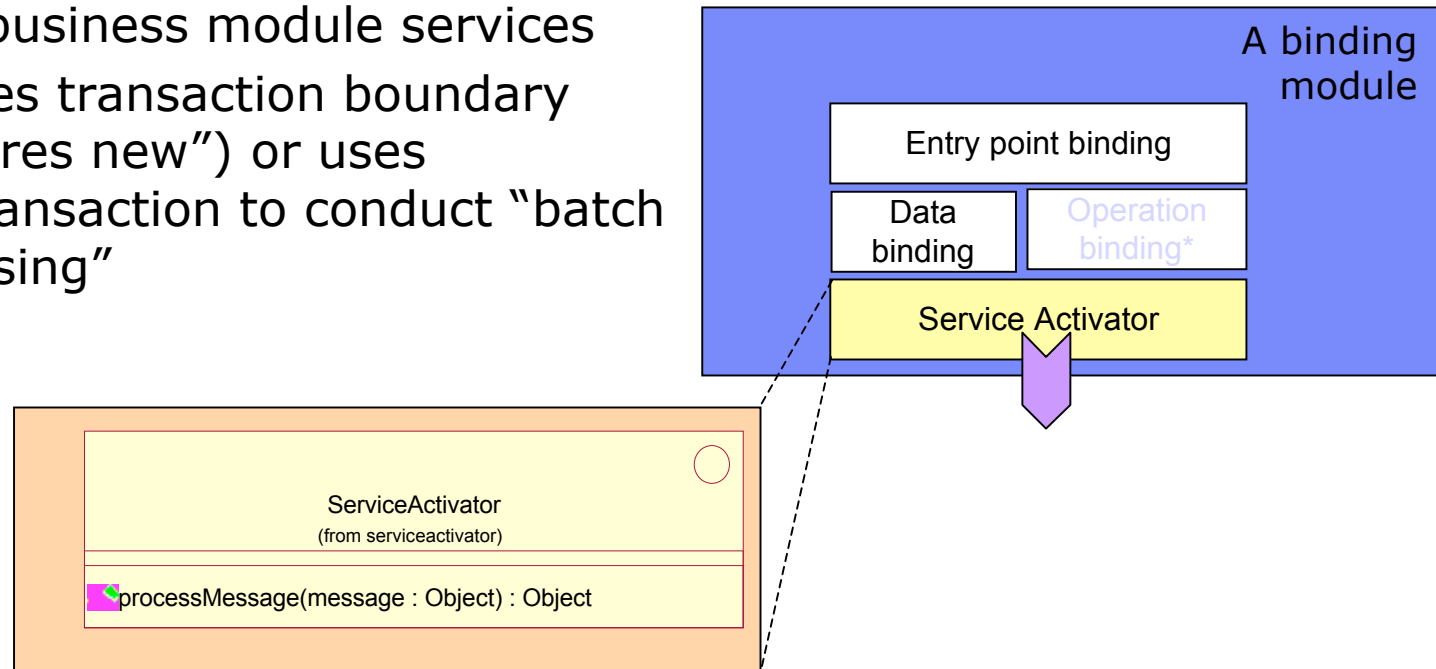
Binding module example - JAXB data binding

- Data Binding service interfaces
- Generic component (implementation) for JAXB
 - Injected with generated JAXB-factory



Binding module example - Service Activator

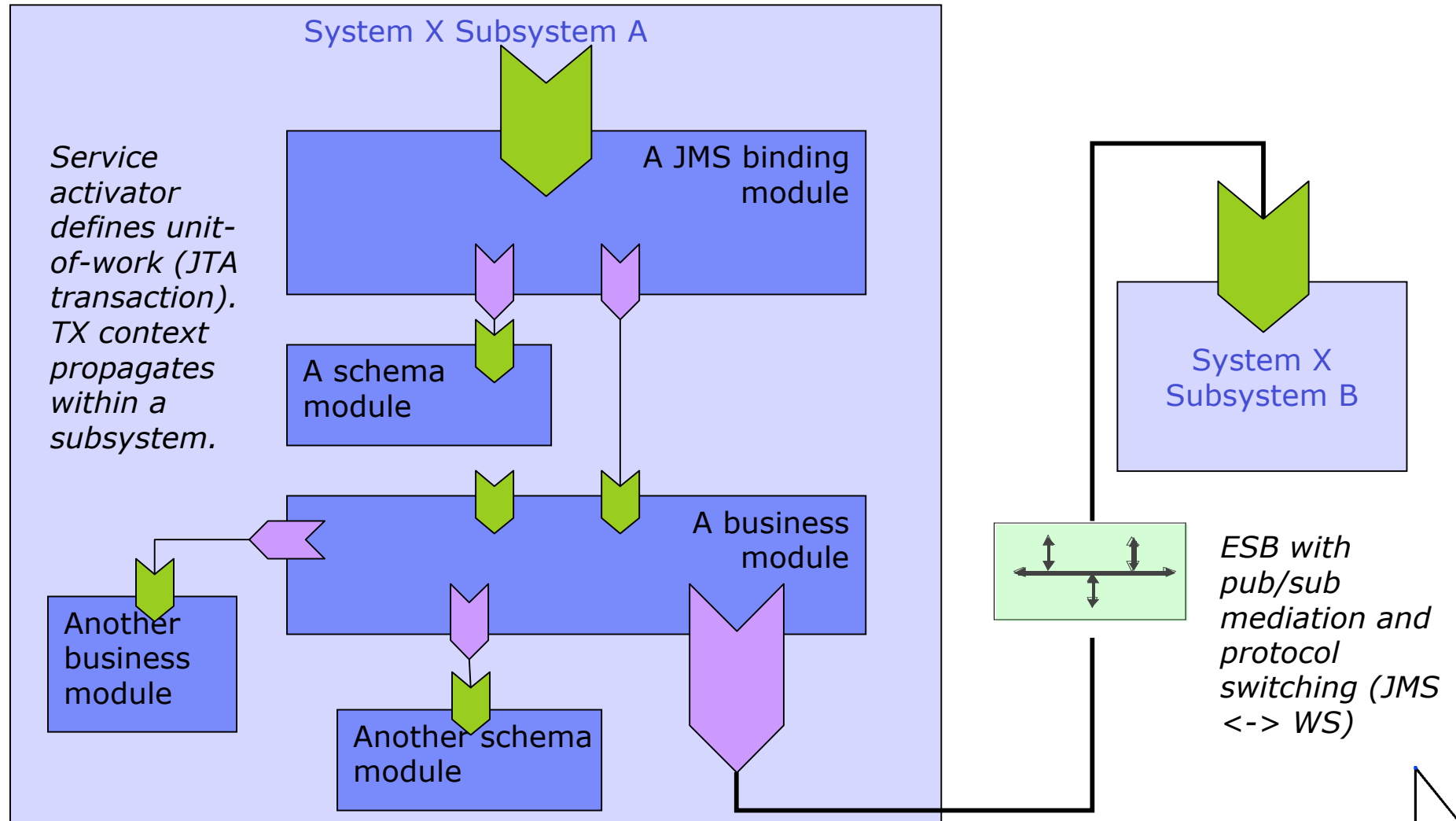
- Custom ServiceActivator component
 - Processes inbound JAXB message
 - Uses data from message to invoke wired business module services
 - Declares transaction boundary ("requires new") or uses UserTransaction to conduct "batch processing"



Schema module

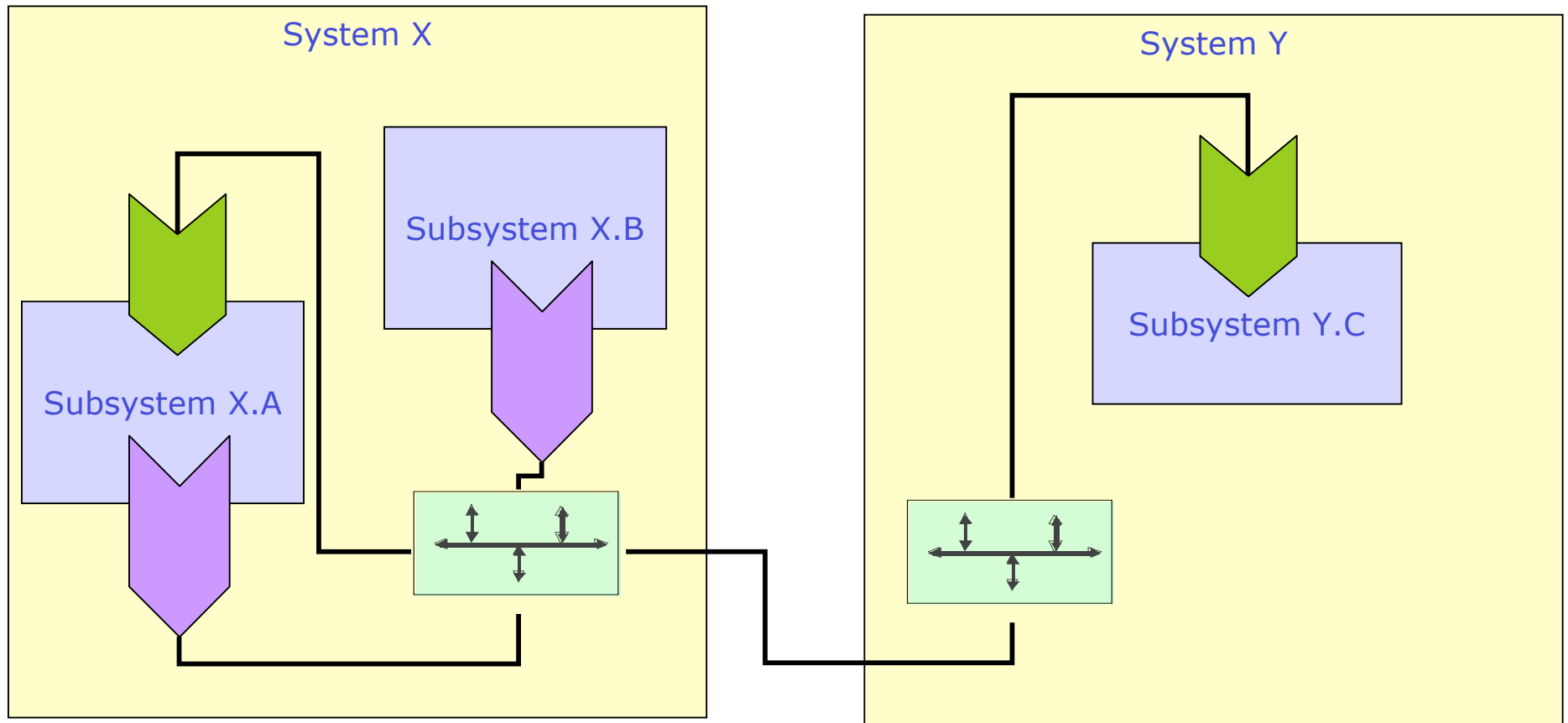
- Models business object types
 - XML schema ComplexType
 - E.g. Production order, Production Object Event
- Used to compose messages
 - Root element for JMS payload
 - WSDL message for WS / SOAP (document/literal)
- Applies versioning strategy
 - Backwards- and forwards interface compatibility through the “any”-strategy
 - Schema + generated JAXB classes in versioned jar-file
 - E.g. production_object_schema-1.1.0.jar
 - Build system automatically generates JAXB classes and builds snapshot jar when schema file is updated
- A schema of a schema module may import schemas from other schema modules
 - Integrated into build system!

Example subsystem assembly



Federation of systems

- SOA domains are linked by connecting the ESB:s



> Tooling

- The evolution of SOA infrastructure
 - *from SOAP to ESB*
- Service Component Architecture (SCA)
 - *Takes SOA from infrastructure to service modelling*
- SCA in practice
 - *Large-scale SOA within manufacturing*
- **Tooling**
 - ***Demo of high-end SCA tooling (WebSphere Integration Developer)***
- Standardization
 - *Status of SCA standardization efforts*
- Conclusions
 - *Lego re-use and composition has finally arrived!*

Why use SCA tooling ?

Power of abstraction

- hides complexity and implementation details
- allows complex service composition
- top-down or meet-in-the-middle model

Service infrastructure

- "plumbing" components in toolbox
- service runtime

Orchestration support (BPEL)

Demo tooling

Graphical design

WebSphere Integration Developer 6.0
(WID)

deploy

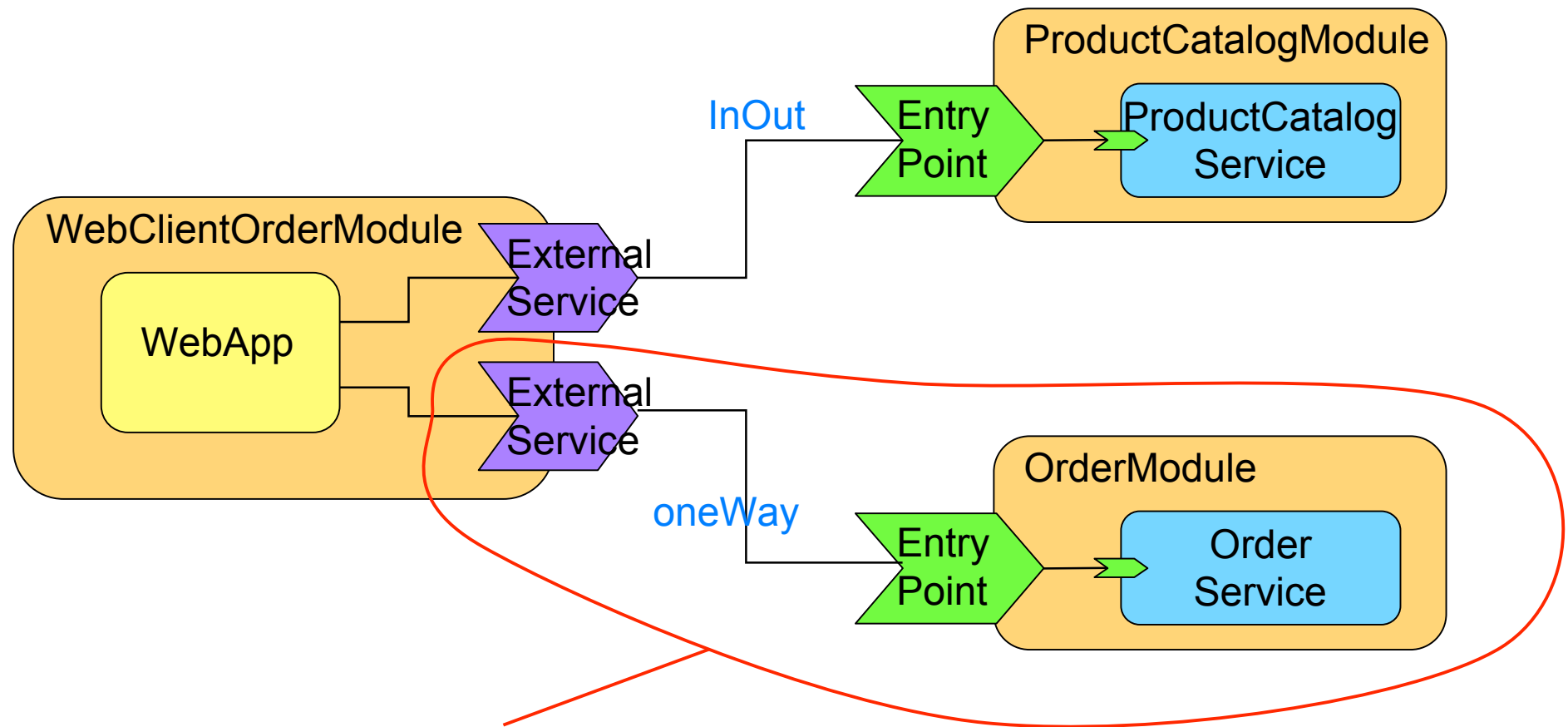
Runtime

WebSphere Process Server 6.0 (WPS)

Upcoming open source tools:

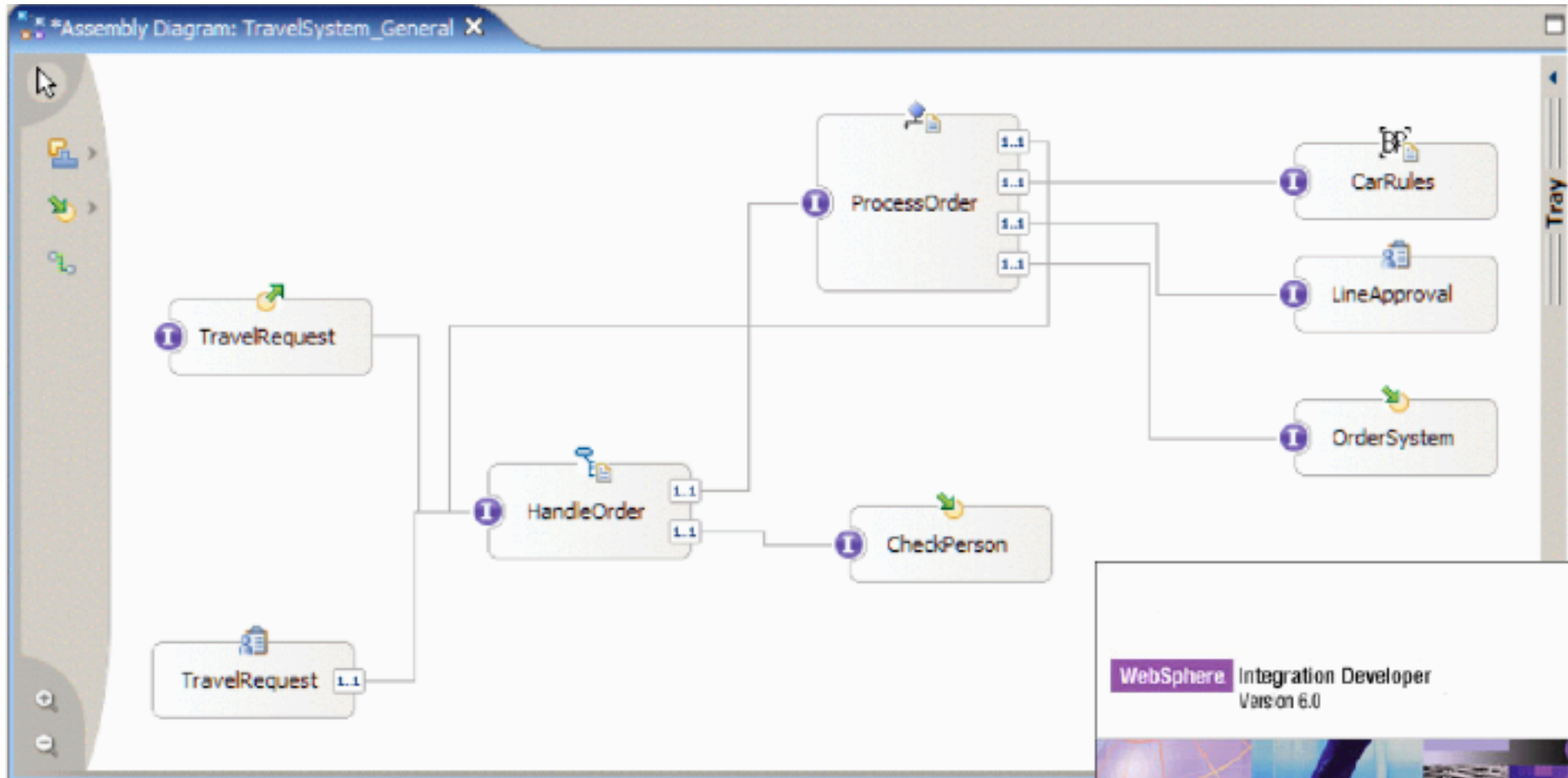
- *Apache Tuscan (in incubator)*
- *Eclipse SOA Tools Platform*

Demo overview



Focus on assembly!

Demo!



> Standardization

- The evolution of SOA infrastructure
 - *from SOAP to ESB*
- Service Component Architecture (SCA)
 - *Takes SOA from infrastructure to service modelling*
- SCA in practice
 - *Large-scale SOA within manufacturing*
- Tooling
 - *Demo of high-end SCA tooling (WebSphere Integration Developer)*
- **Standardization**
 - ***Status of SCA standardization efforts***
- Conclusions
 - *Lego re-use and composition has finally arrived!*
- SCA-get-started kit
 - *Jump-start your SCA-based reference architecture*

SCA specification – ongoing work ...

- Specified by the leading vendors (IBM, BEA, Oracle, ...)
 - architecture comes from top-of-the-line products
 - builds on common core concepts
 - tools available (*WID/WPS early implementation*)
- Specification provides common model and API's to support
 - portability
 - mix-and-match use of builders and runtimes
- Currently: version 0.9 for public review
 - immature but shows direction
- Implementation support: Java, BPEL, C++ and defines extension points

SCA spec - dependency injection

@Remotable

```
public interface MyService {  
    public void serviceMethod(String s);  
}
```

```
public class MyServiceImpl implements MyService {
```

@Property

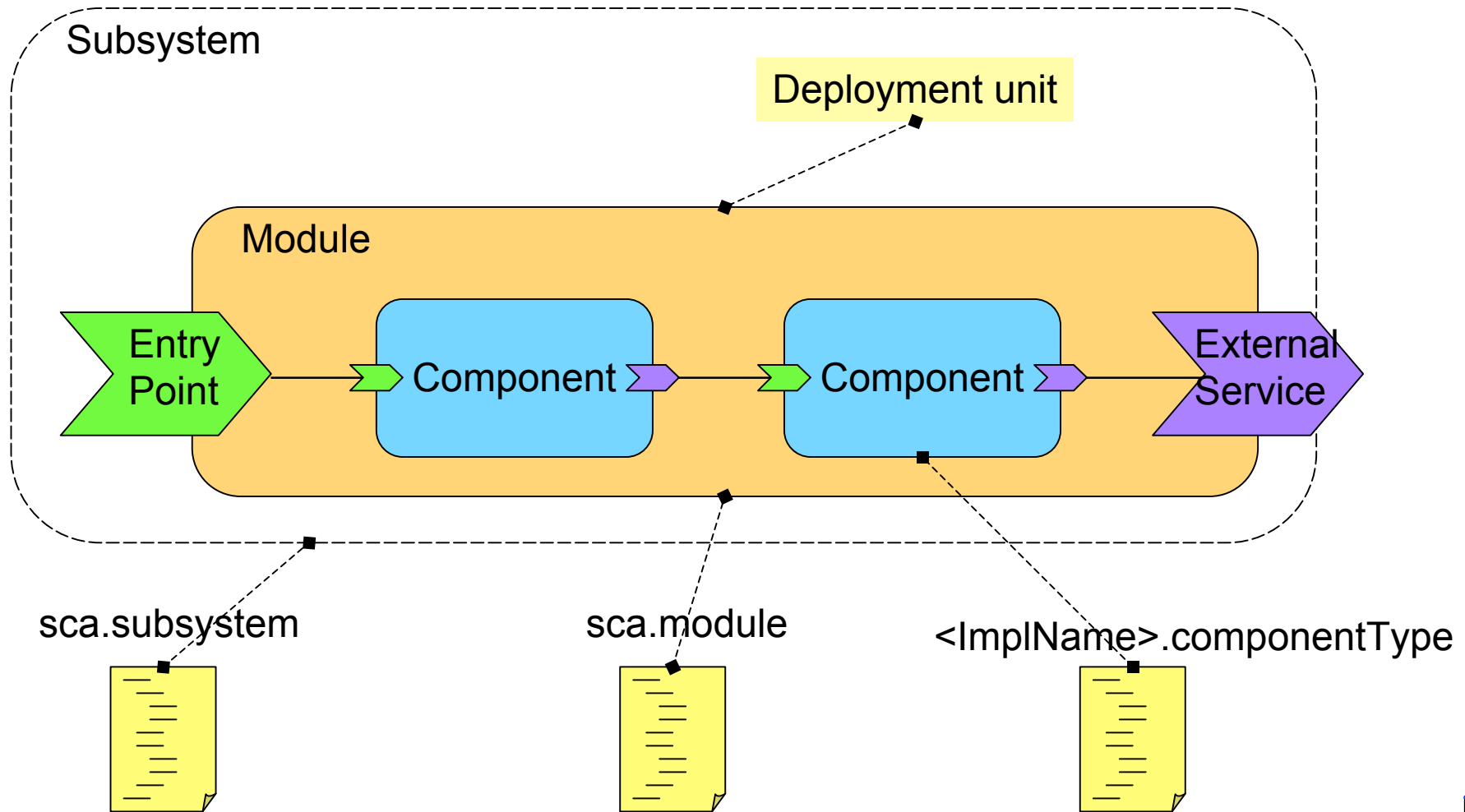
```
private String configProperty;
```

@Reference

```
private AnotherService anotherService;
```

```
public void serviceMethod(String s) {  
    // ...  
}  
}
```

SCA spec – assembly configuration



SCA spec - bindings

- <binding.sca/>
 - not vendor interoperable
 - might be implemented as EJB-remote binding
- <binding.ws port="http://www.../SomeService#
wsdl.endpoint(SomeService/SomeServiceSOAP)"/>
- <binding.ejb/>
 - stateless session EJB

SCA spec – asynch support

- Asynch support
 - callback to Java-interface
 - for WebServices: pass callback endpoint
 - conversational

SCA spec - infrastructure capabilities

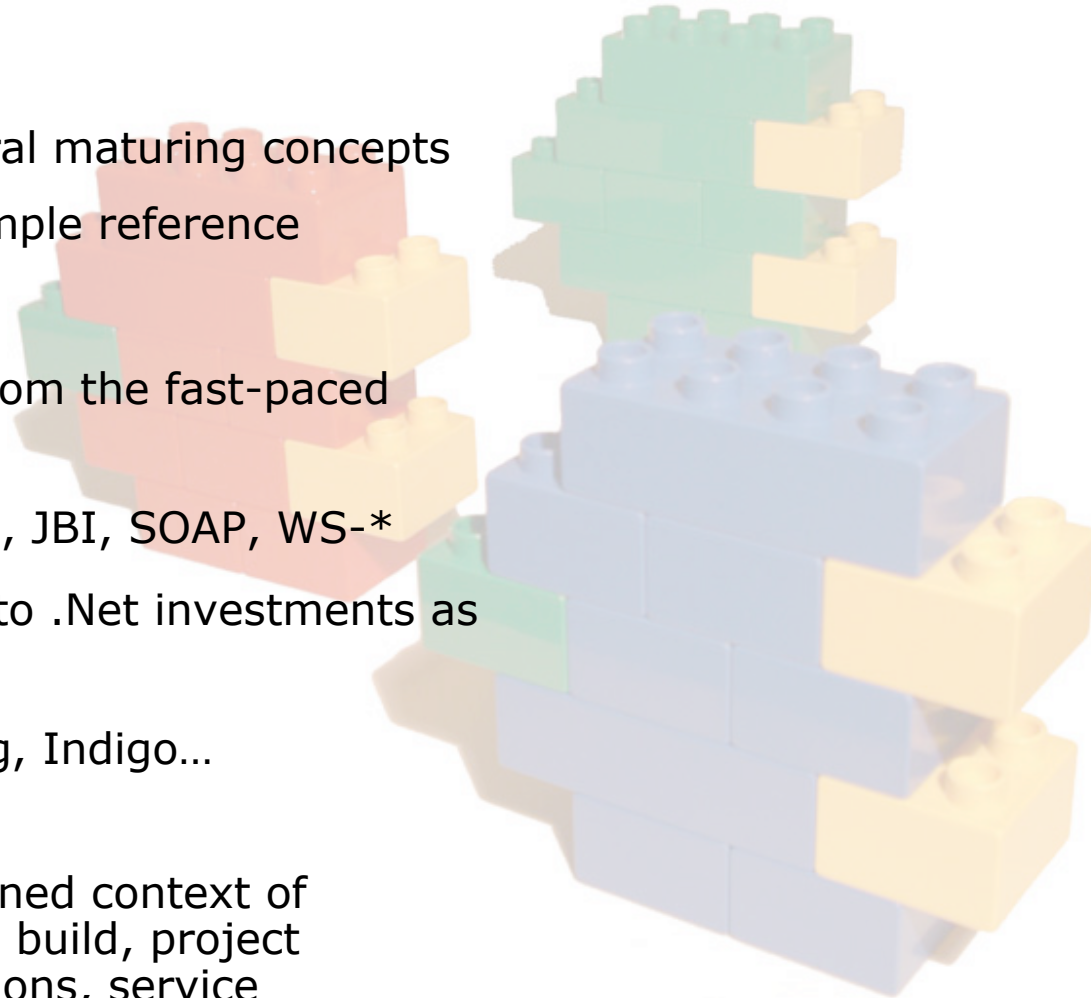
- Declaratively assign as Policies [Proposal]
 - Security
 - Transactions
 - Reliable messaging – property of the transport

> Conclusions

- The evolution of SOA infrastructure
 - *from SOAP to ESB*
- Service Component Architecture (SCA)
 - *Takes SOA from infrastructure to service modelling*
- SCA in practice
 - *Large-scale SOA within manufacturing*
- Tooling
 - *Demo of high-end SCA tooling (WebSphere Integration Developer)*
- Standardization
 - *Status of SCA standardization efforts*
- **Conclusions**
 - ***Lego re-use and composition has finally arrived!***

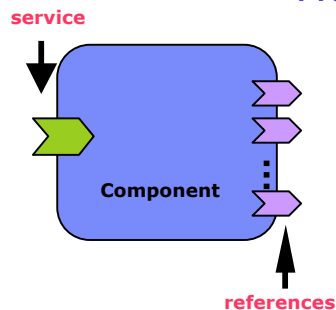
Conclusions

- Lego - At last!
- SCA is the result of several maturing concepts merging into a sound, simple reference architecture
- Shields our investment from the fast-paced infrastructure evolution
 - Corba, RMI, JMS, EJB, JBI, SOAP, WS-*
- SCA could provide value to .Net investments as well
 - DCOM, .Net Remoting, Indigo...
- Disclaimer:
 - A usual, it needs a tuned context of modelling, CCM, test, build, project management, operations, service management...and...SOA infrastructure...



Core values

- ❑ Isolates service implementations from the infrastructure that binds them together
 - Services, Components, References - not java calls, JMS, EJB, WS etc
- ❑ Realizes the Lego vision of composing business services from reusable business modules
 - Recursive composition (Component m:m Module m:m Subsystem)
- ❑ Unifies adapter, integration glue and core service development within a single concept
 - Language bindings (BPEL, Java, C++, Interface Map, Rule Engine, Human Task) for component implementations.



As a side-affect...

*SCA is bootstrapped from Dependency Injection
and a set of structuring principles*

...SO...

The core architecture specified by
SCA actually makes SCA itself
pluggable

(if you resist from using some shortcuts provided by the SCA
Java language binding)

SCA get-started kit

- Would you like to build on our experience?
- Packaged SCA best-practice
 - Introductory workshop
 - Service modelling
 - Business Object / Schema / WSDL management and Versioning strategy
 - Test automation / TDD integration
 - SCA Build System set-up (binary version / dependency management)
 - Change Control structure
 - Reference application
- Open Source- or High-End tooling best-practice
 - Guidelines for applying Spring in an SCA context
 - WebSphere Process Server and WID