Integration technologies in Java

PA165
autumn
2012

Jiří Kolář
Faculty of Informatics
Masaryk University
kolar@fi.muni.cz

Speaker introduction

Jiří Kolář

- Phd student of Doc. Tomas Pitner
- Member of LaSArIS Lab
- Research in Business Process Management
- Industry Partnership coordination
  - Red Hat, IBM etc..
- Process analyst
- Background in System Integration
- More on:
  - http://www.fi.muni.cz/~xkolar2

Lecture summary

- Motivation
- 2 Levels of integration
  - Application level, system level
- Application level: Component containers
  - JBI , OSGi

BREAK 10mins;

- System Integration
  - Messaging – approaches, standards, technologies
  - Integration patterns
  - Middleware architecture approaches
  - Integration scenarios, examples

What should you learn today

- Basic idea of integration and its importance
- Rough idea why and how to integrate:
  - System components together
  - Systems together
- Overview of key technologies in area of integration
- Overview of architectonic approaches to integration
- Understand integration schemas

Integration on application level

- Integration of components within one system
  - Autonomous components communicating together inside one application container
  - Example: In one application server (J2EE)

- Operating system level:
  - Standardized communication of OS kernel with applications

- Communication among apps through OS
  - Example: D-BUS Linux

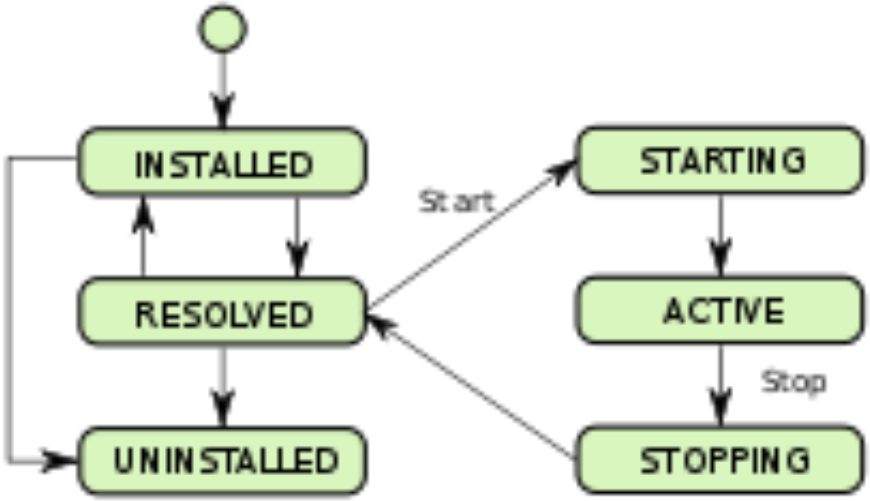
Motivation for integration on Application level

- Component based approach to SW design
  - Independent isolated components
  - Universal reusable components
  - Component encapsulation
  - Minimal public interfaces
  - Provision of services
  - „Components marketplace“

- Platform/environments interconnection
  - Programming languages
  - Runtime environments
  - Operating systems

Component containers in Java

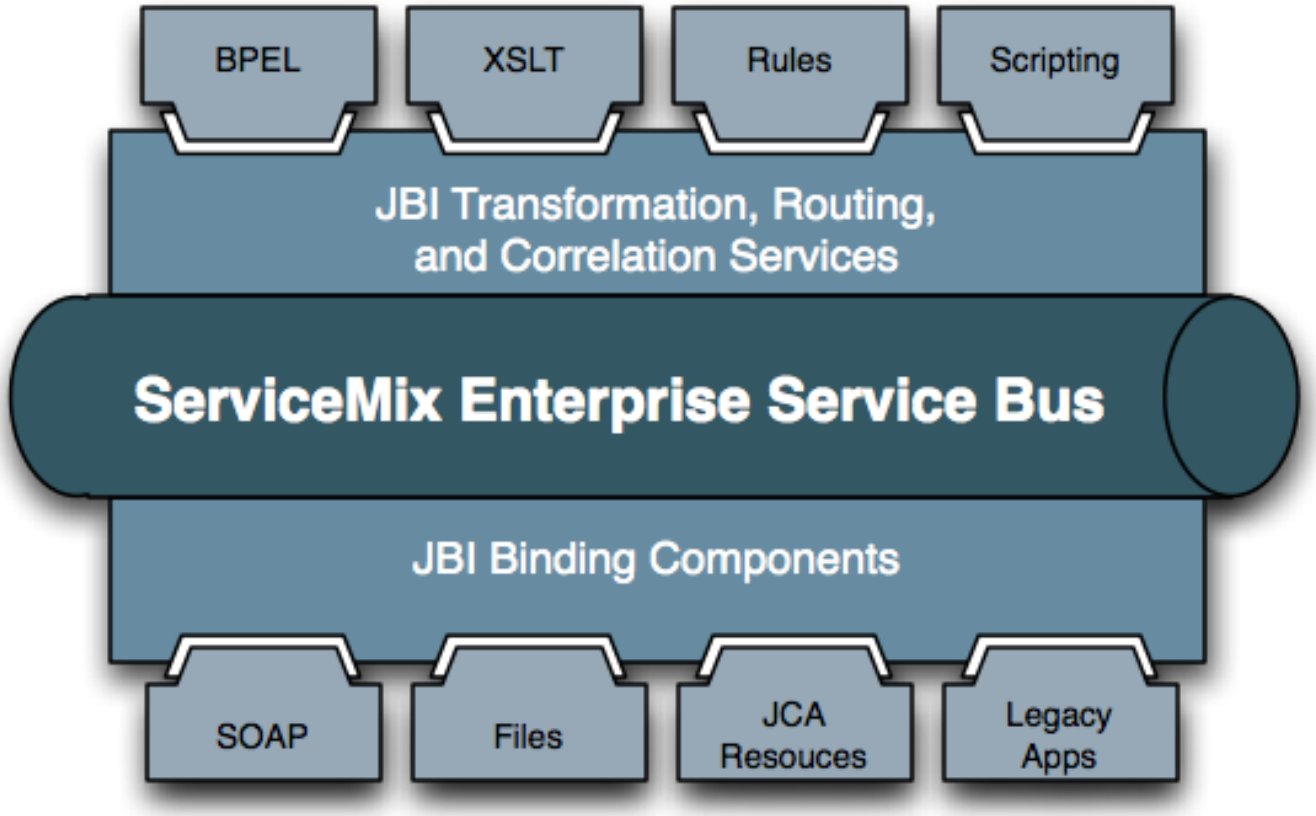
- Runtime environment for components
- Whole container runs as a service inside application server
- Container serve in component life-cycle usually:
  - Deploy
  - Init
  - Start
  - Stop
  - Undeploy



JBI

- First attempt to standardize a meta-container
- Proposed by SUN Microsystems
  - Implemented in OpenESB a Apache Service Mix
  - Lack of wide acceptance across other vendors
- Define architecture of plug-in components
  - Standard does not define components
  - Components communicating through messaging
- Components can serve as containers (nesting)
  - Examples:
    - BPEL engine
    - XSLT transformation engine

JBI Architecture- ESB

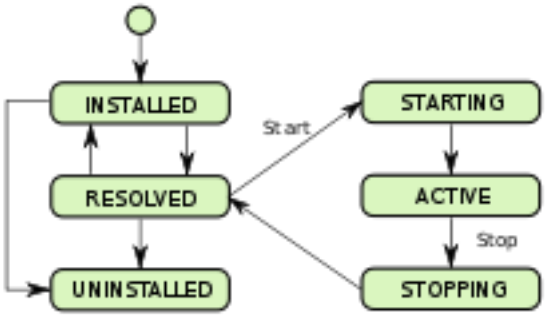


OSGi framework

- Modern standard used in micro-containers
  - Strong support from Eclipse Foundation
  - BTW: Initial purpose was smart homes :)
  - Specification defined for:
    - J2EE
    - Java SE
    - Java ME
- Used in major JavaEE Application servers
  - Lower level than JBI
  - More versatile and simplified
  - Make POJO sexy again

OSGi framework

- Dynamic component model
  - Support of component lifecycle
  - Possibility of manipulation with components in runtime
  - Dynamic classloading
- Components provide functionality as a service
  - Service management
  - Service registry



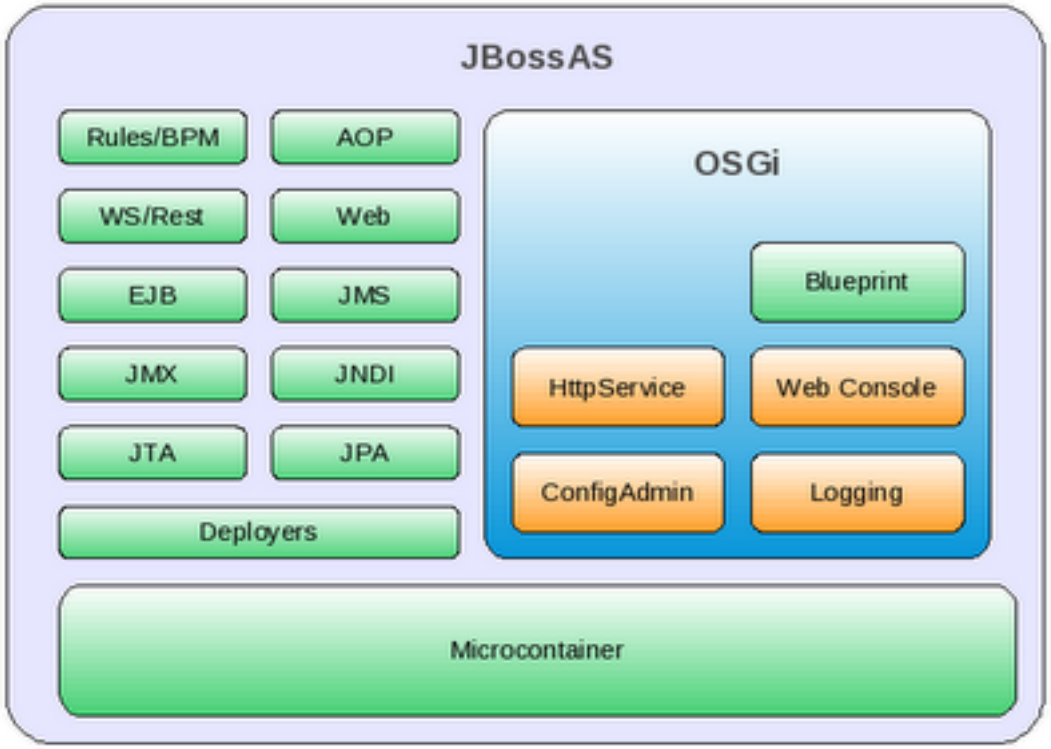
OSGi framework components

- Components (bundles) are simple *.jar
  - Component description stored in MANIFEST.MF
    - Component dependencies
    - Versioning
    - Metadata

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: HelloWorld Plug-in
Bundle-SymbolicName: com.javaworld.sample.HelloWorld
Bundle-Version: 1.0.0
Bundle-Activator: com.javaworld.sample.helloworld.Activator
Bundle-Vendor: JAVAWORLD
Bundle-Localization: plugin
Import-Package: org.osgi.framework;version="1.3.0"
```

```
package com.javaworld.sample.helloworld;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
public class Activator implements BundleActivator {
    public void start(BundleContext context) throws Exception {
        System.out.println("Hello world");
    }
    public void stop(BundleContext context) throws Exception {
        System.out.println("Goodbye World");
    }
}
```

OSGi container example: JBoss Microcontainer



OSGi conclusions

- Big hype few years ago
- Consensus of Big players
  - Glassfish (SUN), SpringSource, JBoss (Red Hat),
  - WebSphere (IBM), Weblogic (Oracle)..
- Strict component approach
- Idea: ”From scratch and more simple”

- However: 10 years of standardization and 2 years of massive use? Hopefully not.

OSGi based products:

- Equinox– referenční implementace OSGi (EclipseRT)
- Virgo – OSGi container (WS/AS) (EclipseRT)
- JbossAS
- Apache ServiceMix 4.X.X (with JBI on top)

OSGi vs JBI

- JBI
  - Put emphasis on integration and data transformation
  - Probably "obsolete" today
  - Define NMR and inter-component communication
- OSGi
  - Universal, modular framework
  - Specification for many programming languages, not just Java
  - Lightweight, low level

Break 10 mins

System integration

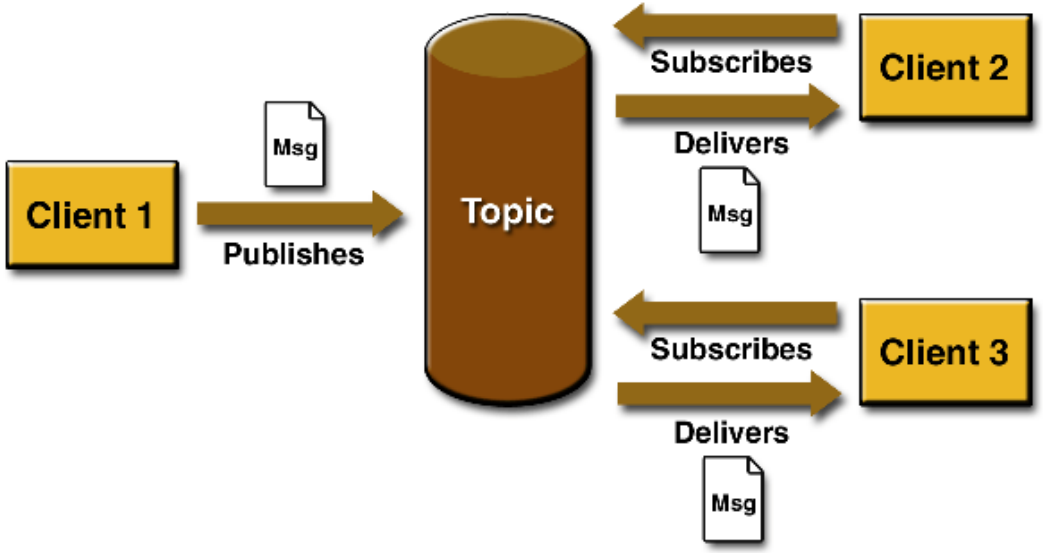
- Interconnection of autonomous systems and autonomous environments
  - (EAI – Enterprise Application Integration)
  - Very heterogenous environment
  - Various communication protocols
  - Various interfaces and APIs
  - Components within one system
    - Finance, E-commerce, ERP, Internal systems
- Provision of services between systems
  - Business partners, Customers, Government ..
  - Web Services, public APIs
  - Necessity of orchestration = **BPM**

Motivation (System Integration)

- Legacy systems
  - Old but good and reliable software
  - Wrapped to provide modern interfaces (WS etc..)
- Interconnection of autonomous Enterprise Information Systems
  - Various EIS within one organisation/enterprise
  - Provision of functionally to "outside world":
    - Customers
    - Business partners
- Acquisitions and splitting of Enterprises
- Outsourcing of services

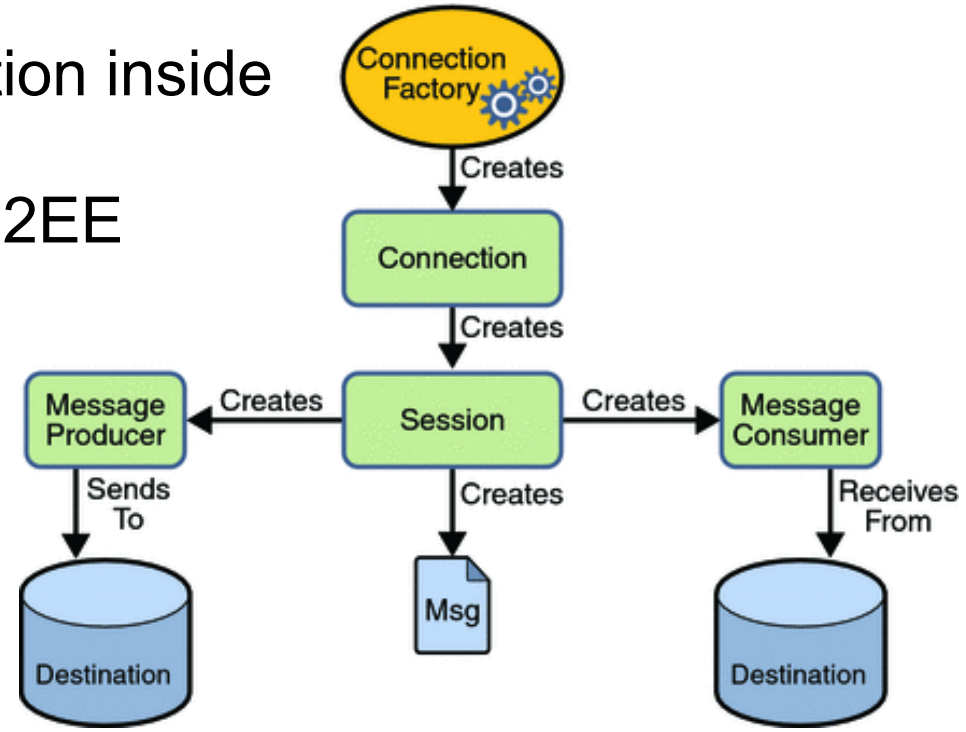
Messaging systems

- Messaging models
  - Point to Point
  - Publisher-Subscriber
- Distributed messaging (clustering)
  - Messaging in cloud
- Persistence
- Monitoring
- Management



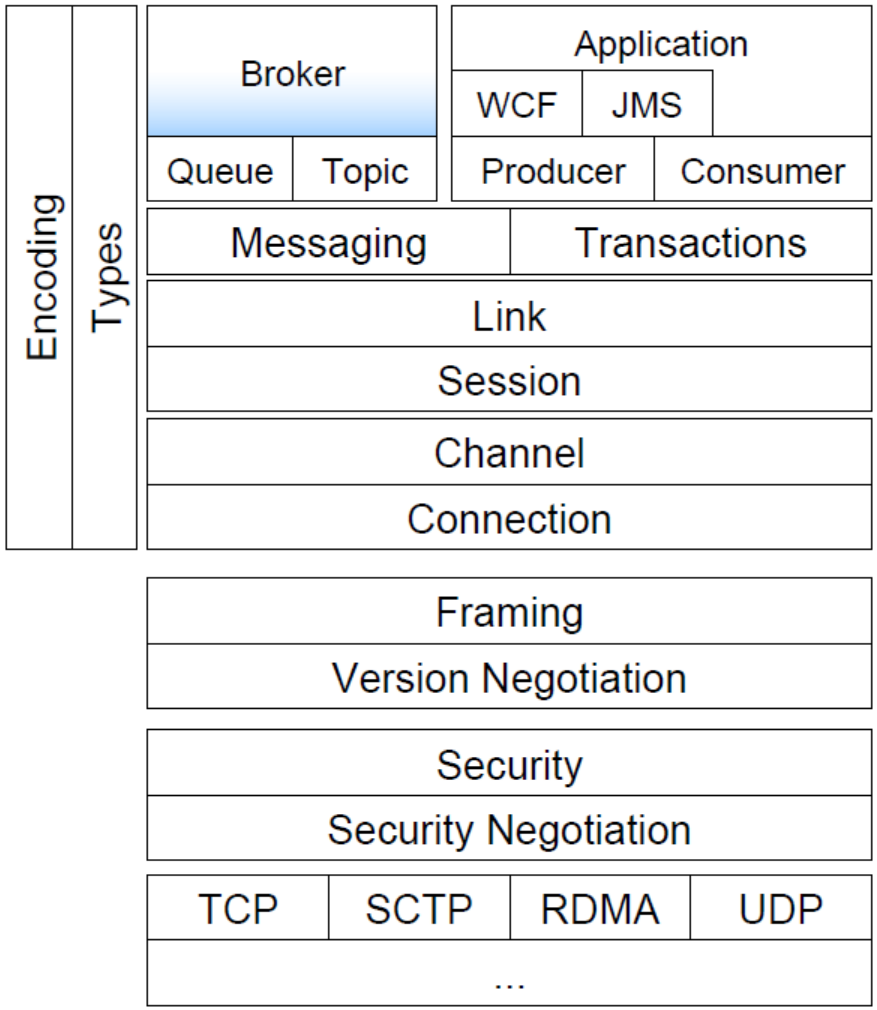
JMS – Messaging API in JavaEE

- Standardized messaging API
- High level, hide implementation details
- Implemented in major messaging systems
- Used both for communication inside and across systems
- Application integration in J2EE



AMQP - messaging protocol standard

- AMQP Consortium standard
  - Wire level protocol
  - Define also API
- Implementations
  - Qpid – Apache
  - RabbitMQ – Vmware
  - Red Hat Enterprise MRG (Qpid-based)



Web services

- Standardized interface, widely used
- Fits well in heterogenous environment
- Most common technology in system integration nowadays
- Need for”service registry”
- Need for orchestration (BPM)
- Significant overhead, not usable in high throughput scenarios
- Different protocols and approaches
  - SOAP
  - REST

BPM: orchestration vs choreography (More in PV207 - BPM)

- Choreography
  - Services communication "logic" hardcoded in components
  - components "know about each other"
  - Communication scenarios being held by services/applications
- Orchestration
  - Services are orchestrated by "conductor"
    - Business Process Execution engines
    - Business Rules
  - Communication "logic" being held conductor

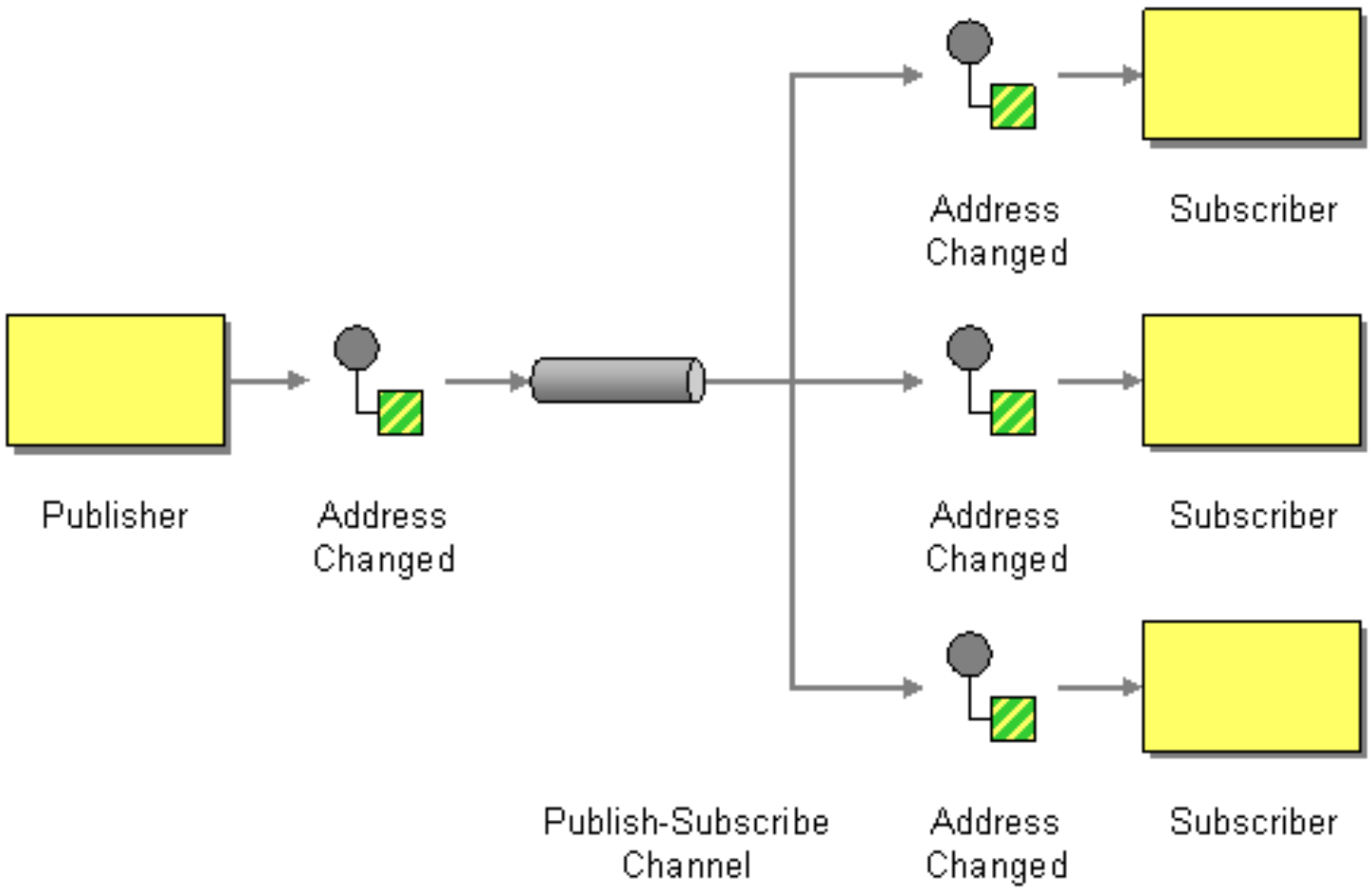
Integration patterns

- Design patterns for integration
- Like GoF, but started by Apache Foundation

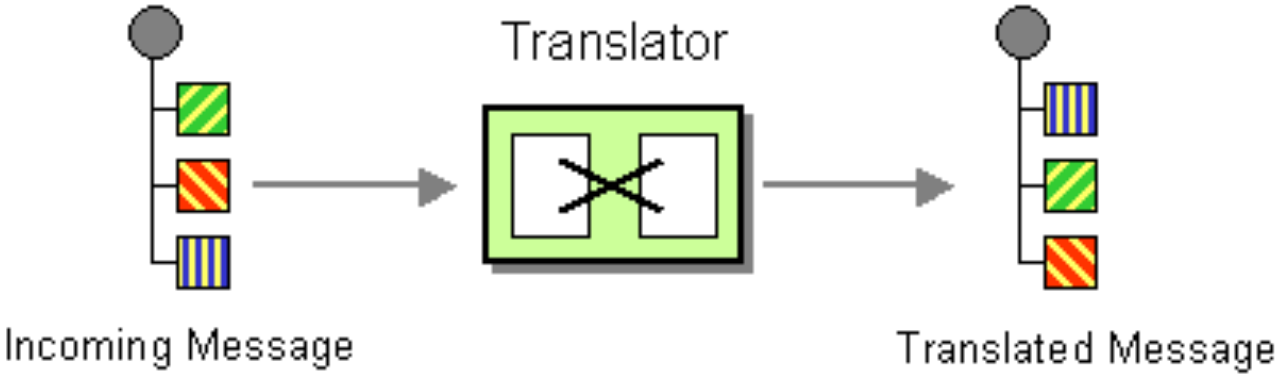
- Standardization effort
- Describe ”best practices” in integration

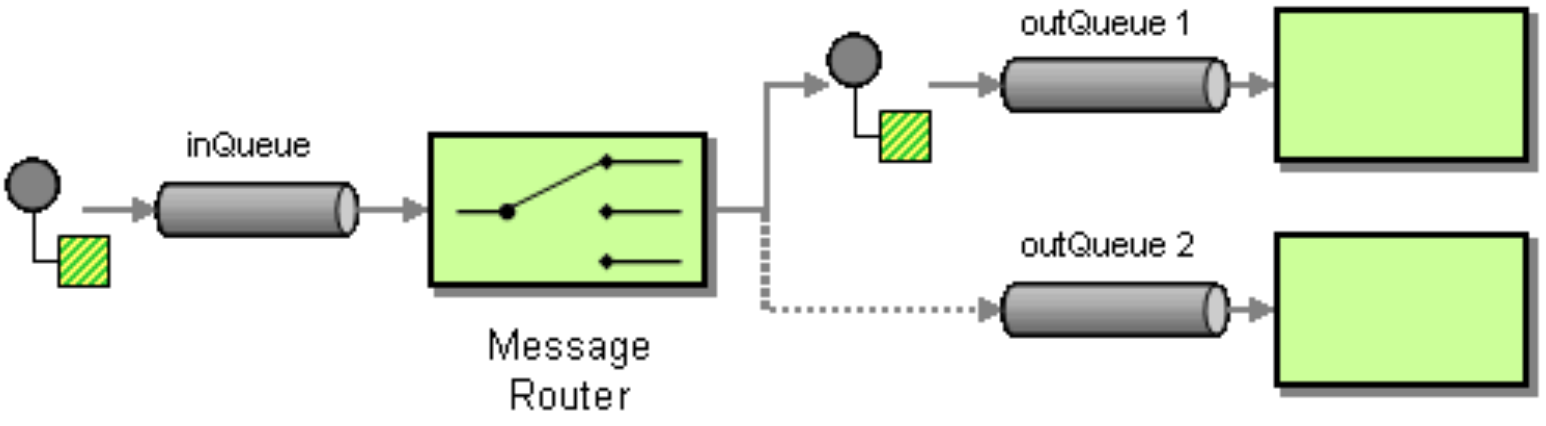
- Supported by various integration platforms
- Supported by design environments and tools

Integration patterns: Publisher-Subscriber



Integration patterns: Translator and message router



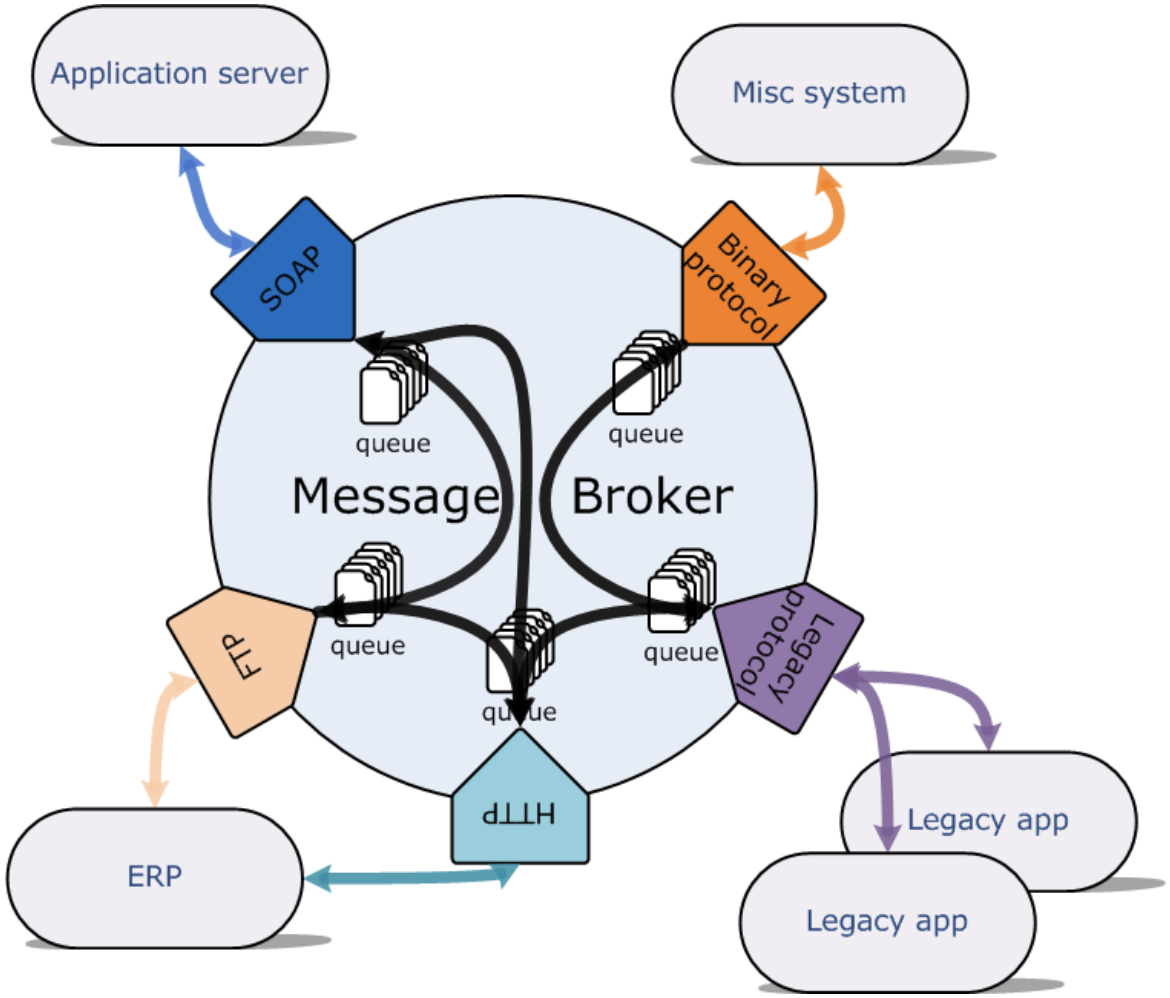


Middleware architecture concepts

- Architectonic concepts
- Describe "logic" of integration
- Different concepts can be based on the same physical representation (implementation) and vice versa

- Implementation of complex integration concept can be sometimes relatively simple

Integration architecture: Service hub

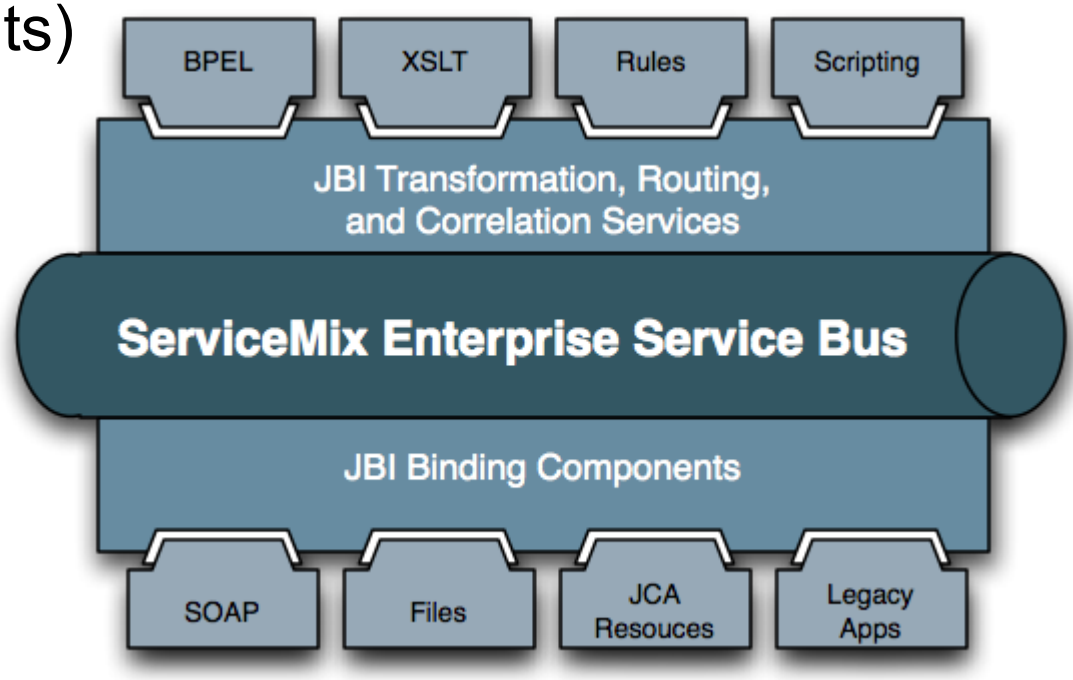


Integration architecture: Service hub

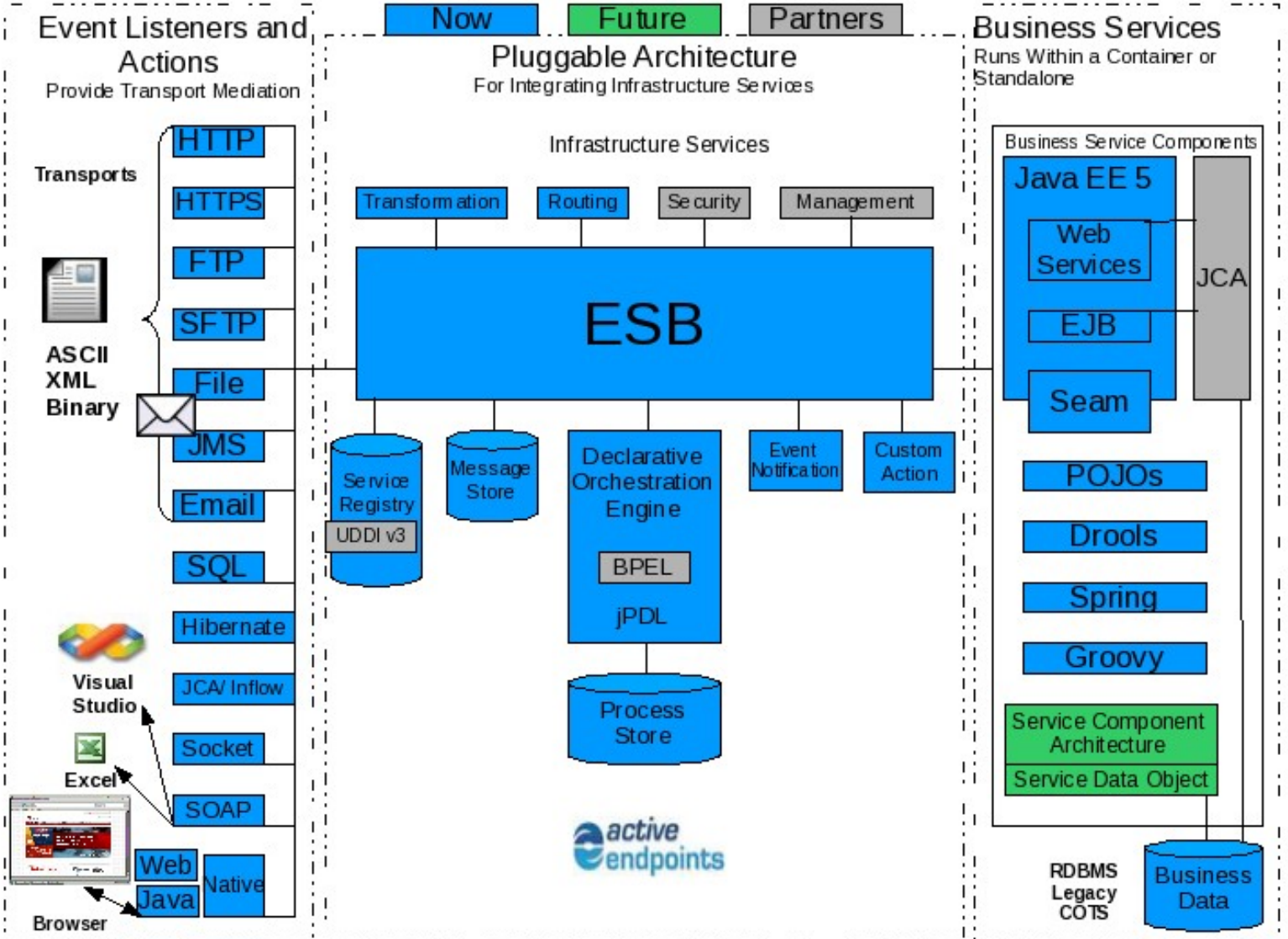
- Older approach, but still widely used
- + Smaller overhead
- + simple management
- - Less flexibility in message transformation
- - Not useful for more complex topologies
  - Ex.: cascaded / distributed integration

Integration architecture: Enterprise Service Bus

- Messages are:
  - Translated to "normalised format" (typically JMS)
  - Processed by any components if required
  - Translated to output format
  - Translation is performed by adapters (binding components)



Integration architecture: Enterprise Service Bus



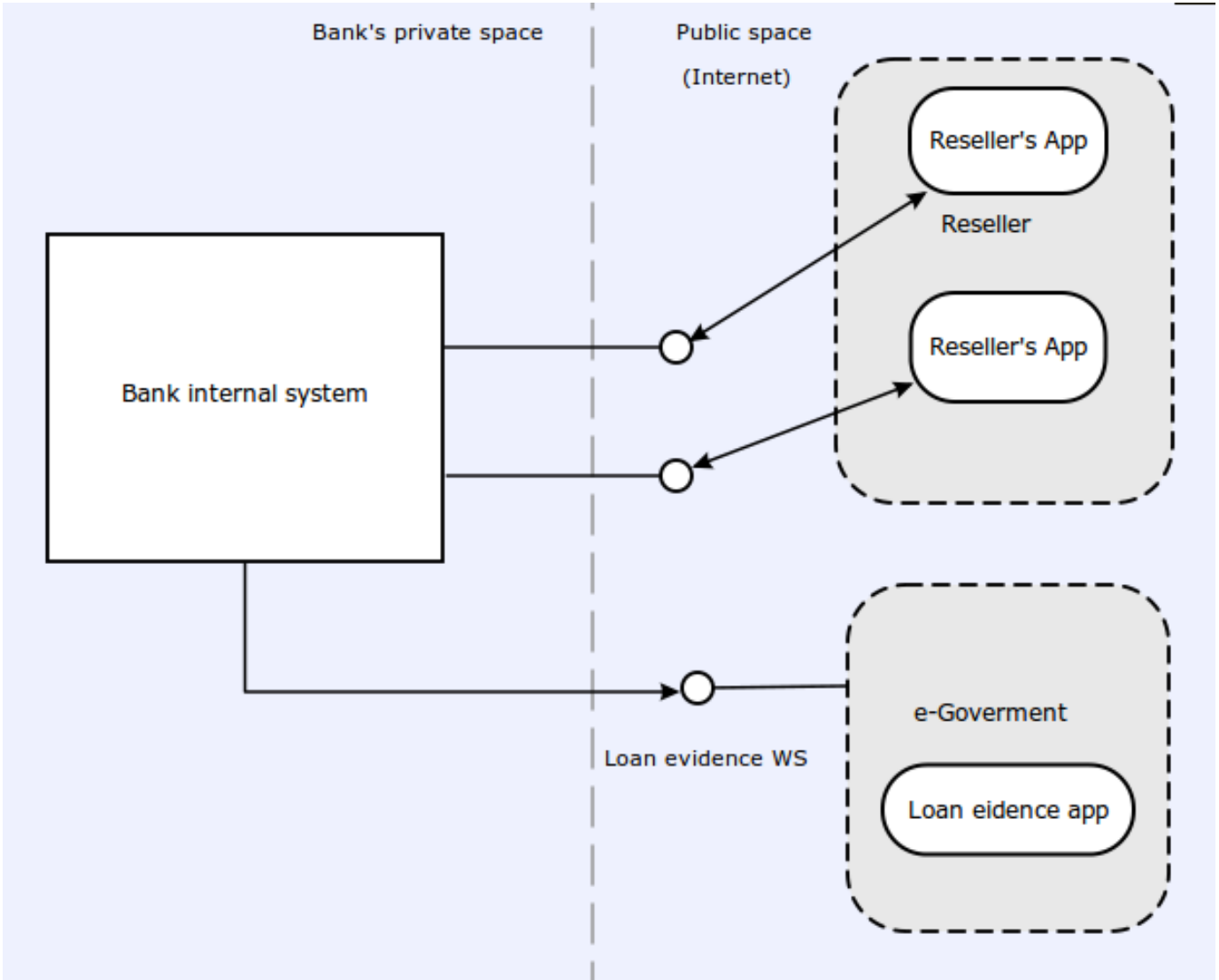
Service Bus (ESB)

- Primary in context of Java and Web Services
- + More flexible– better support of integration patterns
- + Better for more complex topologies
  - Clustering, cascading, distributed ESB etc...
- - Higher overhead (lower throughput)
- + More complicated management of flows

Scenario: Loans

- A bank run internal system for Loans management
  - There is a Cobol module computing Loan Risk value
  - Bank want their resellers to be able to use this module
  - Resellers want to integrate this functionality into their own systems
- Bank want to verify profile of client in national database of debtors and dodgers

Scenario: Loans



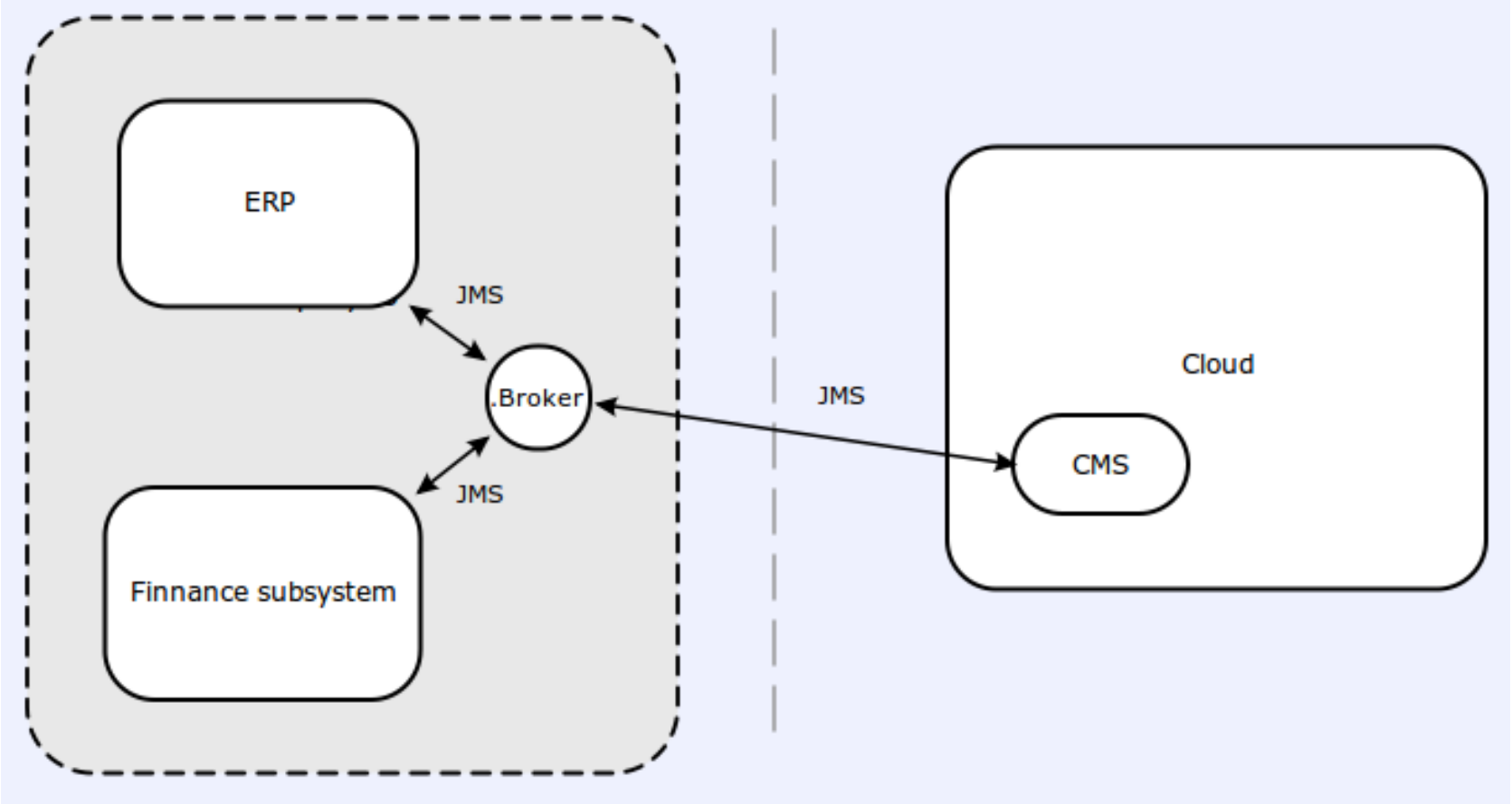
Scenario: Cloud integration

An enterprise "E" use:

- Cloud based CRM
- Internal system for accounting
- Internal e-store

Enterprise "E" wants to integrate those together

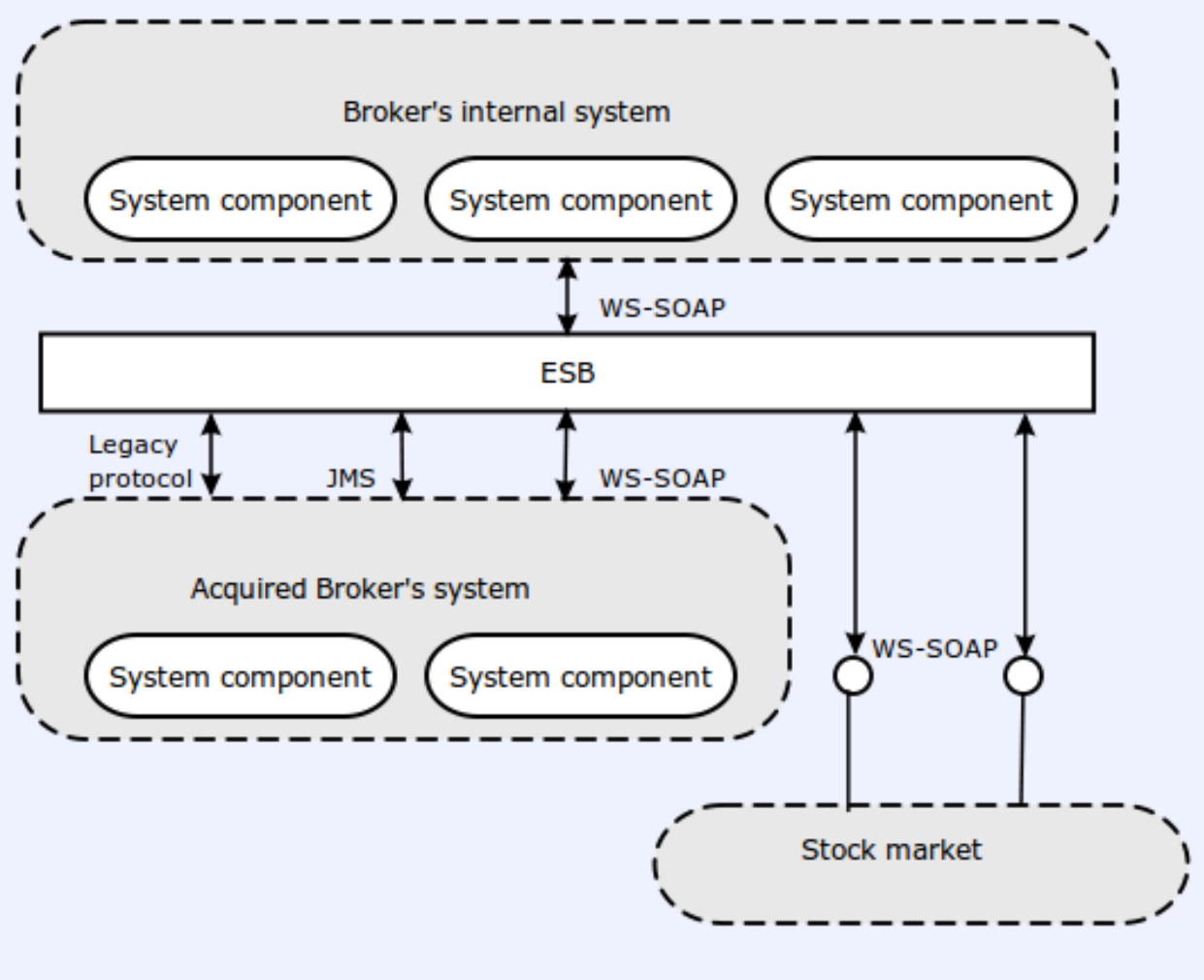
Scenario: Cloud integration



Scenario: Acquisition

- A Broker "B" perform acquisition of their smaller competitor "C"
- Competitor "C" has perfect system for stock portfolio management
- Broker "B" has also one containing all their clients , but not as good as the new one
  - Migration is planned for 3 years
  - They need to use both systems in parallel for this period
- There are strict throughput and reliability requirements

Scenario: Acquisition



Integration issues and problems

- Incompatible protocol implementations
- Changes in public APIs

- Malformed data
  - Messages validation
  - Errors in data transfers
- Security

- Performance
- Isolated cloud technologies

FIN
Questions?

PA165
autumn
2012

Jiří Kolář
Faculty of Informatics
Masaryk University
kolar@fi.muni.cz