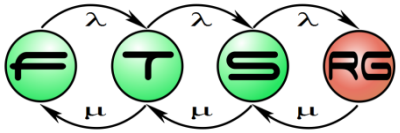
Nagyvállalati szoftvertechnológiák

Szolgáltatás-orientált környezetben

Ráth István

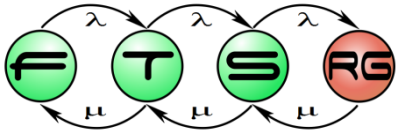


Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Tartalom

- Java Enterprise bevezető
- JEE 6 újdonságai
- JBoss Application Server 7 áttekintés
- OSGi
  - OSGi alapok
  - Szolgáltatások
- Objektum-relációs leképezés
  - ORM alapok
  - JPA
- jBPM5 áttekintés
- Integrált architektúra a házi feladathoz

JEE Overview



JEE Overview

- What is JEE ?
- The JEE Architecture
- Overview of JEE technologies
- JEE Application Servers
- The JEE Development Environment
- Exercise
- Summary

What is JEE ?

- A set of technologies for developing enterprise applications in Java
- Specified by Sun and the Java Community Process (JCP).
- Implemented by JEE vendors.
- Implementations of JEE technologies are provided within Application Servers.
- Previously named J2EE (until version 1.4) current version is JEE 5.

What is JEE ?

- The Java Platform

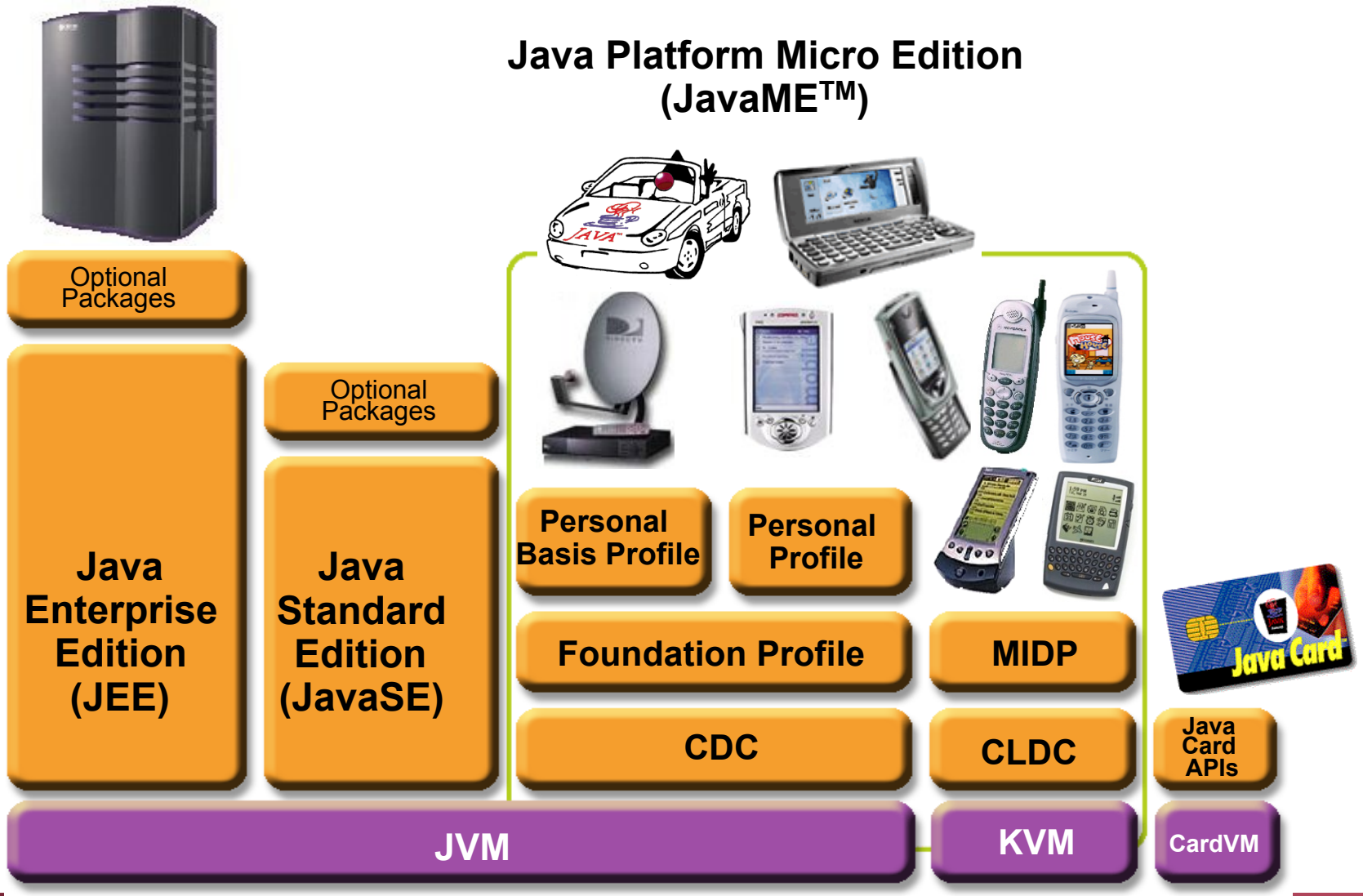
Java Technology Enabled Devices

Java Technology Enabled Desktop

Workgroup Server

High-End Server

The Java Platform



* Under development in JCP

Why do we need JEE ?

- Distribution
- Transactions
- Security
- Scalability
- Persistence

Value to developers.

* Can use ***any JEE implementation*** for development and deployment
  * Use production-quality standard implementation which is free for development/deployment
  * Use high-end commercial JEE products for scalability and fault-tolerance
* Vast amount of JEE ***community resources***
  * Many JEE related books, articles, tutorials, quality code you can use, best practice guidelines, design patterns etc.
* Can use off-the-shelf 3rd-party business components

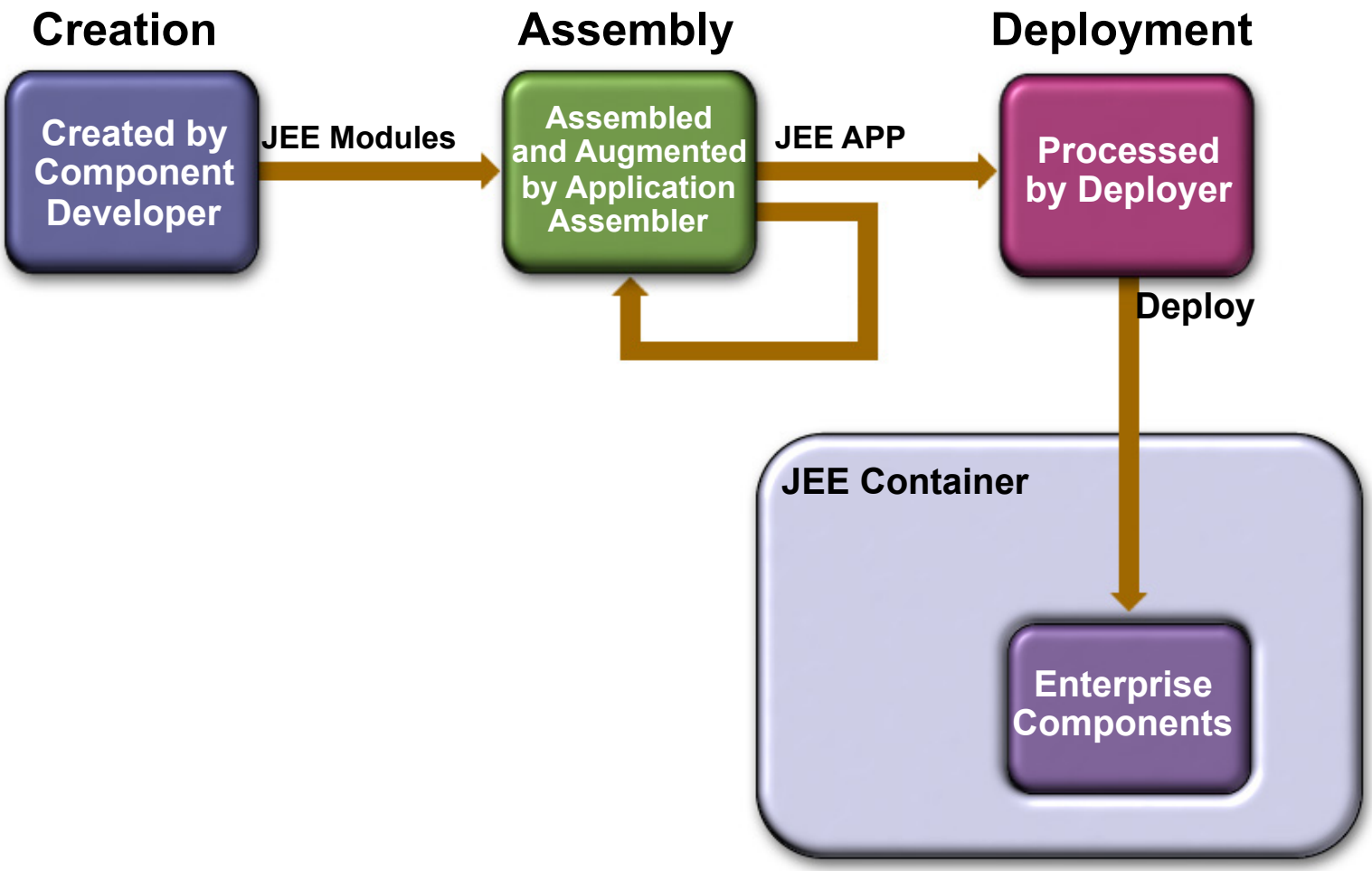
Value to vendors

- Vendors work together on specifications and then compete in implementations
  - In the areas of Scalability, Performance, Reliability, Availability, Management and development tools, and so on
- Freedom to innovate while maintaining the portability of applications
- ***Do not have create/maintain their own proprietary APIs***

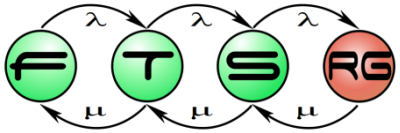
JEE Development Roles

* Component provider
  * Bean provider
* Application assembler
* Deployer
* Platform provider
  * Container provider
* Tools provider
* System administrator

The JEE Life Cycle



The JEE Architecture



The JEE Architecture

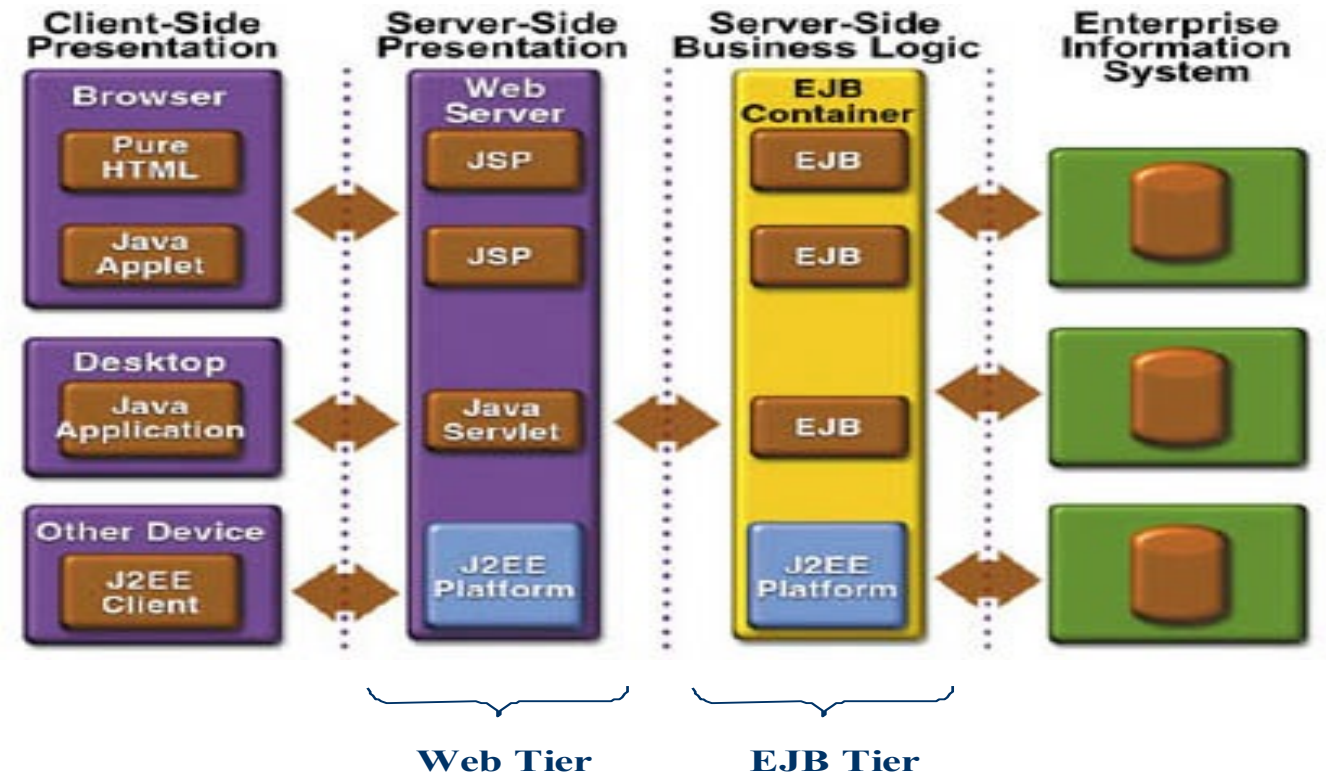
- N-tier architecture
- Comprised of technologies for the business tier the presentation tier and other system services.
- Runs within the application server and within specific containers (web container, EJB container) within the Application server.

The JEE Architecture

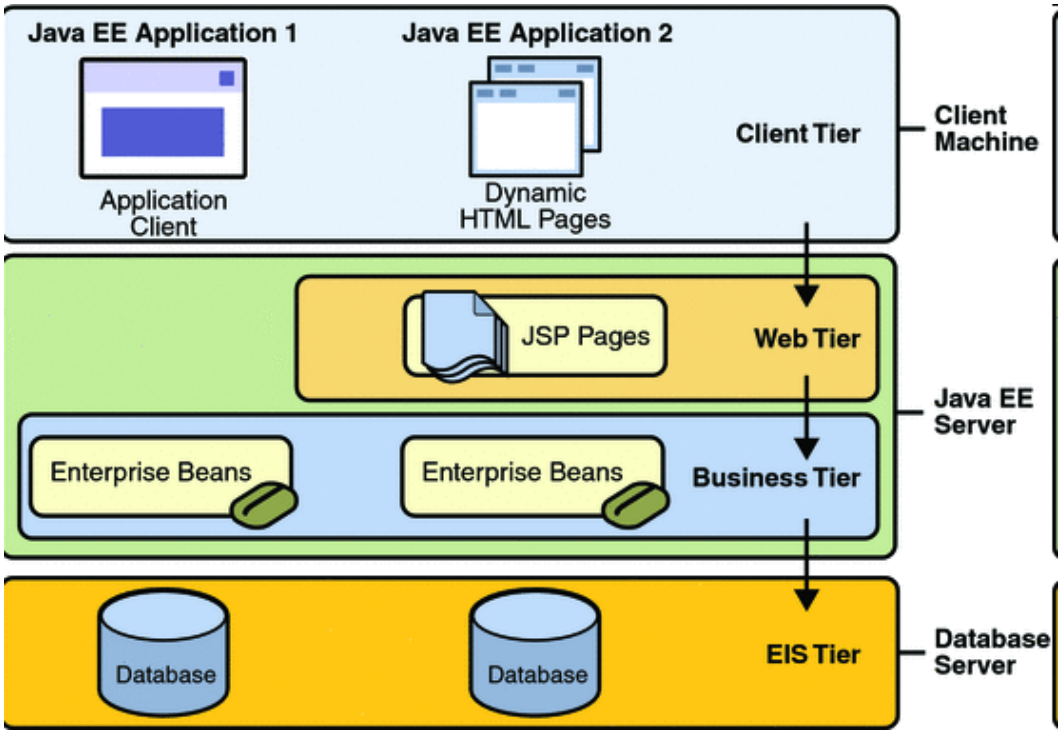
- Uses the "component and container" model in which container provides system services in a well-defined and as industry standard

- JEE is a standard that also provides portability of code because it is based on Java technology and standard-based Java programming APIs

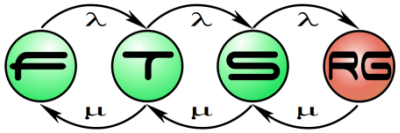
Three-Tier architecture



JEE Tier Architecture



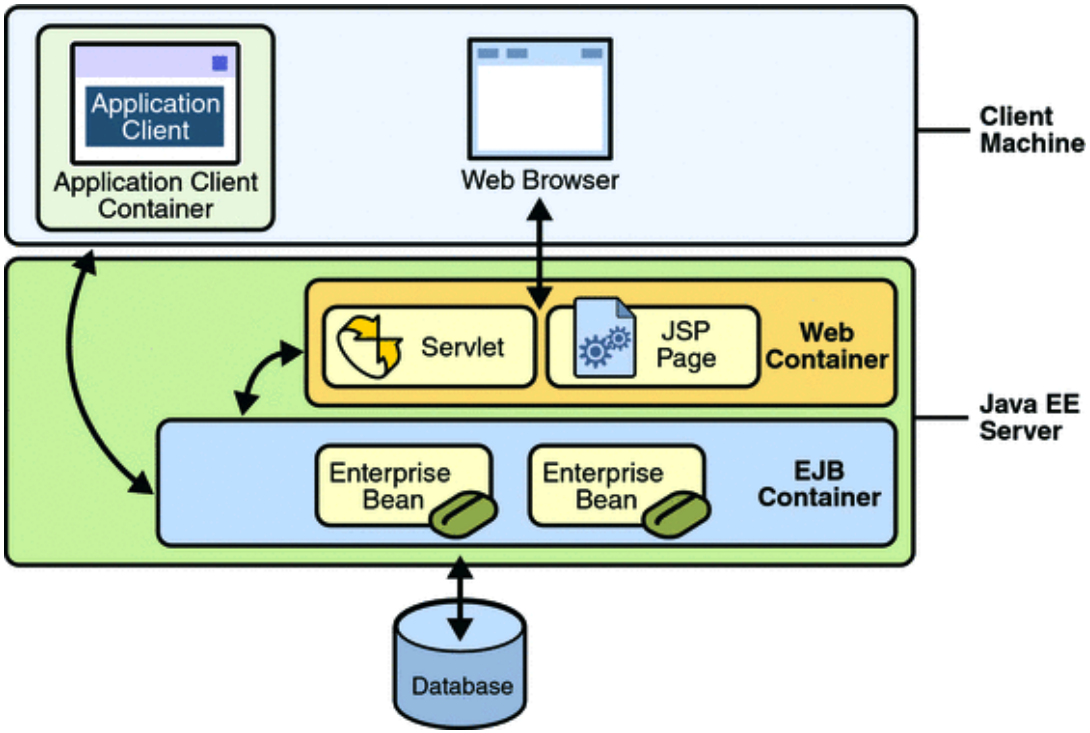
JEE Application Servers



JEE Application Servers

- JEE vendors provide their implementation of JEE technologies within an Application Server.
- Each application server has its own implementation of JEE standards as well as some proprietary features.
- Comprised of a Web Container,EJB Container and other server services.

The App server and JEE containers.



References

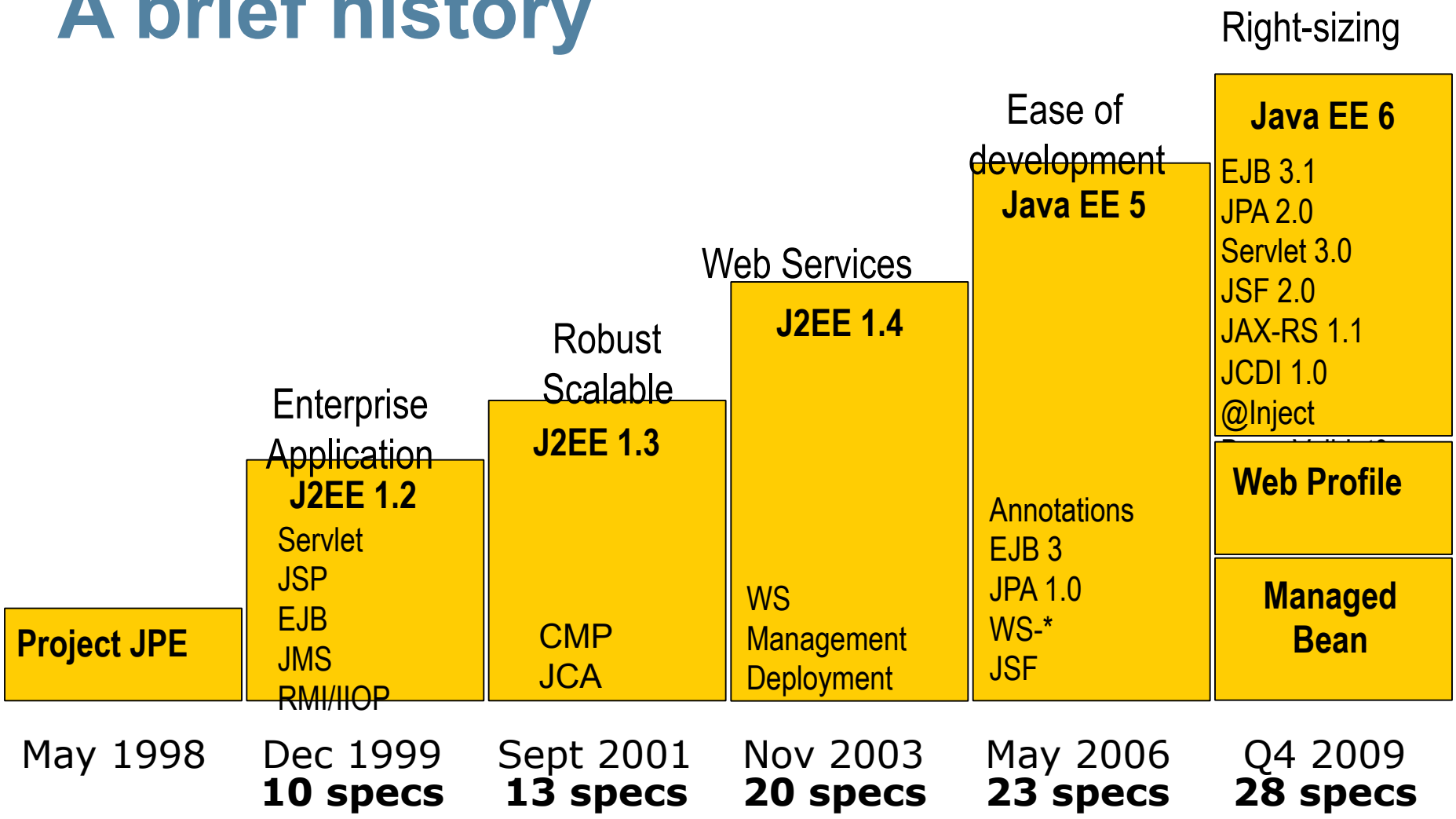
- Oded Nissan: JEE Overview
  - http://www.slideshare.net/odedns/jee-course-jee-overview
- Imre Gábor: Enterprise Java Beans
  - (UML bázisú modellezés és analízis tárgy anyagai, 2008-2010)

Java EE 6: The Next Generation Enterprise Application Platform

Topics

- Java EE 6 themes
  - Right-sizing, Extensibility, Ease of development
- Java EE 6 Technologies
  - Managed Beans & Interceptors
  - Bean validation
  - DI (JSR-330) and CDI (JSR-299)
  - JPA 2.0
  - Servlet 3.0
  - EJB 3.1
  - JSF 2.0
  - JAX-RS 1.1

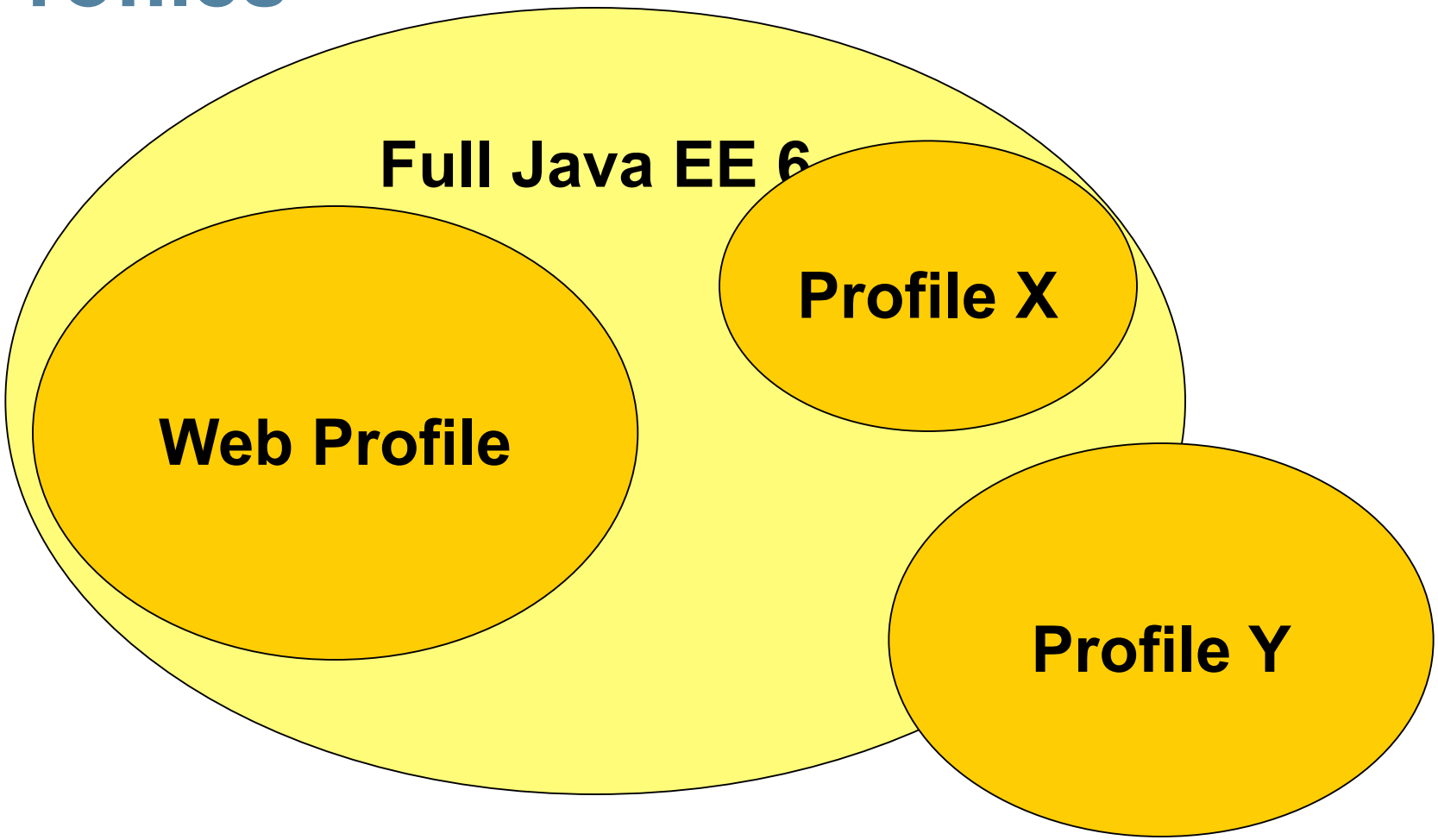
A brief history



Major Themes of Java EE 6

- Right-sizing
  - Profiles & Pruning
  - Use only what you need
  - Web profile, EJB Lite
- Extensibility (Pluggability)
  - Use 3rd-party frameworks or libraries without extra configuration
- Ease of development
  - Default over configuration
  - Dependency Injection (DI)

Profiles



Pruning

- Marks some specifications “deprecated” in next version
  - Might disappear from Java EE 7
- Pruned in Java EE 6
  - Entity CMP 2.x
  - JAX-RPC
  - JAX-R
  - JSR 88 (Java EE Application Deployment)

Managed Bean 1.0: What is it?

- Managed Beans are container-managed POJOs
  - Lightweight component model
- Support a small set of basic services
  - Injection of a resource (@Resource...)
  - Life-cycle management (@PostConstruct, @PreDestroy)
  - Interceptor (@Interceptors, @AroundInvoke)

Managed Beans 1.0: Example

```
@ManagedBean
public class MyPojo {

   @Resource   // Resource injection
   private Datasource ds;

   @PostConstruct  // Life-cycle
   private void init() {
      ....
   }

   public void myMethod() {...}
}
```

Managed Bean vs. EJB and REST

- You could see everything as a Managed Bean with extra services
- An EJB is a Managed Bean with :
  - Transaction support
  - Security
  - Thread safety
- A REST service is a Managed Bean with
  - HTTP support

DI 1.0 (JSR 330): What & Why?

- Stands for “Dependency Injection for Java”
- Java EE 5 has resource injection
  - *@Resource, @PersistenceContext, @EJB*
- But there is no application level injection in Java EE 5
- DI 1.0 introduces *@Inject* annotation (and others) for application level injection
- Spec. leads are Bob Lee (from Google) and Rod Johnson (from SpringSource)

CDI 1.0 (JSR-299)

- Used to be called “WebBeans”
- Uses annotations defined in DI 1.0 (JSR-330)

@Inject ShoppingCart cart;

- Bean discovery and wiring
  - Container discovers beans and wires everything together automatically
- Let you use EJBs directly as JSF backing beans
- Every object managed by CDI has a well-defined scope
  - @Dependent, @ConversationalScoped, @RequestScoped, @SessionScoped, @ApplicationScoped,

EJB 3.1

- Packaging in a war file
- Optional Local Interfaces
- Singleton
- Asynchronous calls
- Cron-based Timer Service
- Embeddable Container
- EJB Lite

JSF 2.0

- Annotation based
  - *faces-config.xml* is optional
- Easy navigation
- Page description language
  - Facelets (preferred)
  - JSP (still supported)
- Templating
- Composite components
- Ajax support

Bean Validation 1.0: What is it?

- Enable declarative validation in your applications
- Constrain Once, Validate Anywhere
  - restriction on a bean, field or property
  - not null, size between 1 and 7, valid email...
- Standard way to validate constraints
- Integration with JPA 2.0 & JSF 2.0

JAX-RS 1.1

- JAX-RS 1.0 has been released in 2008
  - RESTful Services
  - POJO and Annotations Based
  - Maps HTTP verbs (Get, Post, Put, Delete...)
- JAX-RS 1.1
  - Part of Java EE 6
  - EJB support

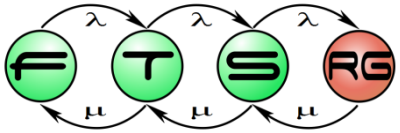
Summary

- Java EE 6 is a standard based next generation enterprise application development and deployment platform
- Java EE 6 leverages proven best practice ideas

References

- Eugene Bogart: JEE6 Overview
  - http://www.slideshare.net/eugenebogaart/java-enterprise-edition-6-overview
- Roberto Chinicci: JEE6 Platform overview and highlights
  - http://www.slideshare.net/alexismp/java-ee-6

JBoss AS7



Motivations for AS7

- Improve usability
  - with AS4 and AS5 the JBoss user used to be an expert
- Increase manageability
  - with AS4 and AS5 each server is based on specific configuration
- Simplify configurations
  - with AS4 and AS5 each service has its own configuration approach
- Increase performance
  - much faster than previous versions

7 reasons to love AS7

- Blazingly fast (<3s startup)
- Lightweight
- Modular / service-oriented
- Hot parallel deployment
- Elegant administration
- Domain management
- Easy testing

Key new features

- Fast and lightweight
- Supports domain (multi node) management
- Multiple consistent management interfaces
  CLI, Java API, HTTP API, Web Console
- Unified user-focused configuration
- Modular
  - Pluggable components of the application server, referenced from profiles, based on `module.xml`
  - tha basis of classloading in AS7

JBoss modules

JBoss Modules is a standalone implementation of a modular (non-hierarchical) class loading and execution environment for Java. In other words, rather than a single class loader which loads all JARs into a flat class path, *each library becomes a module which only links against the exact modules it depends on, and nothing more*. It implements a thread-safe, fast, and highly concurrent delegating class loader model, coupled to an extensible module resolution system, which combine to form a unique, simple and powerful system for application execution and distribution.

module.xml

```
<module xmlns="urn:jboss:module:1.0" name="org.jboss.weld.api">

   <resources>
      <resource-root path="weld-api-1.1.Final.jar"/>
   </resources>

   <dependencies>
      <module name="javax.api"/>
      <module name="javax.enterprise.api"/>
      <module name="javax.inject.api"/>
      <module name="javax.persistence.api"/>
      <module name="javax.interceptor.api"/>
      <module name="javax.servlet.api"/>
   </dependencies>
</module>
```

Jar that provides the module classes

Other modules that this module can see

User deployments

- User deployments are modules too
- Sets up dependencies on some modules automatically
  (e.g. JPA, Hibernate, WebServices)
- The user can also set up their own dependencies on app server modules

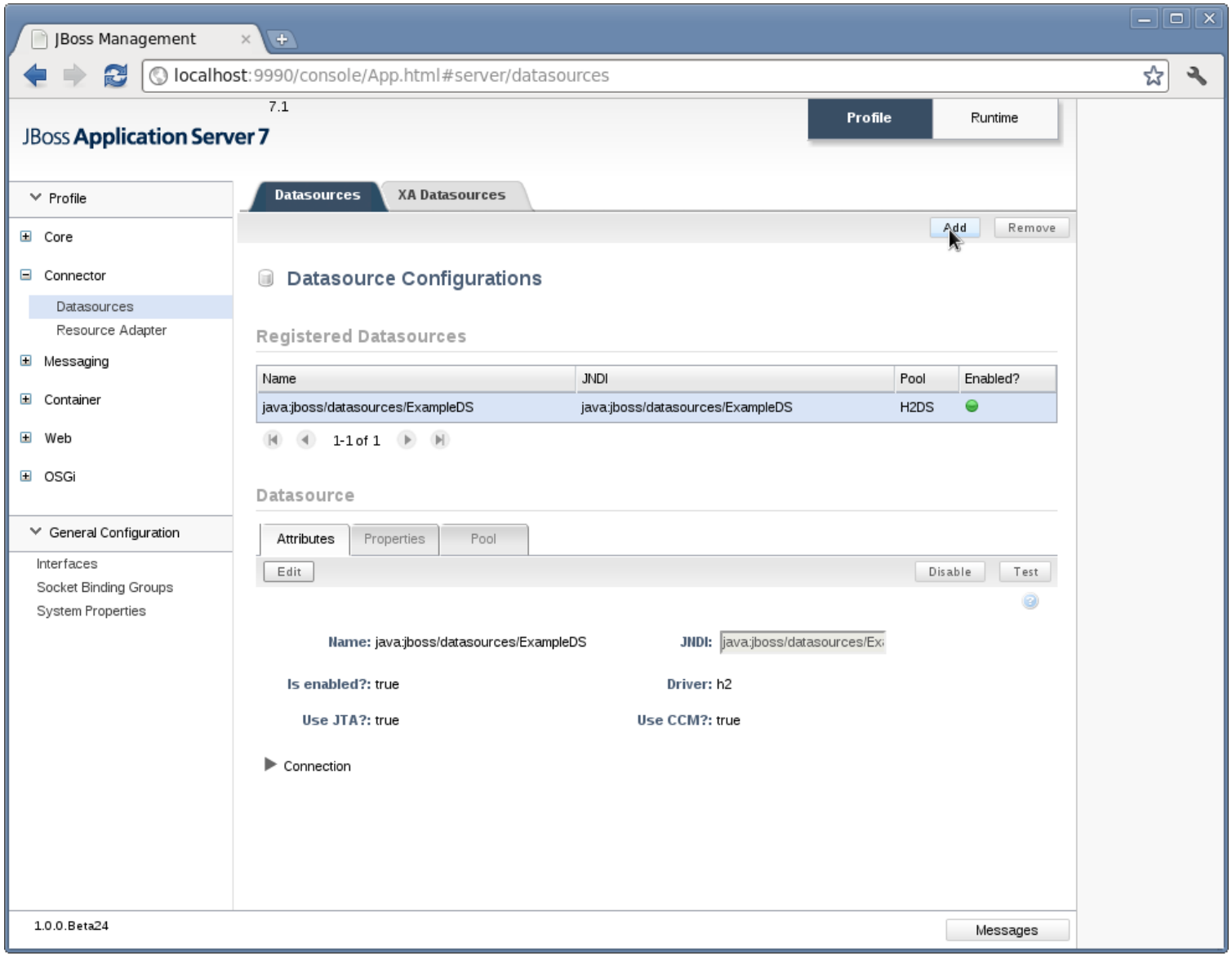
Services

- In AS7 almost everything is a service
- Services are objects that can be started and stopped
- Services can have dependencies on other services
- When all services dependencies are satisfied it will attempt to start
- If a dependency going to be stopped then MSC will stop all dependent services first
- Services can inject dependent services

Two operational modes

- Standalone
  This is the traditional JBoss single JVM server
  This will have management facilities IN-VM
- Domain
  Multi JVM multi server model
  Management coordinated by *Domain Controller Process*
  Multiple server instances (JVMs) per host
  Full lifecycle managed by *Process Controller*

Management console



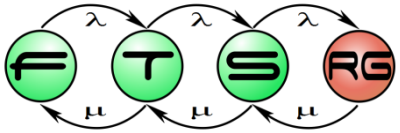
Testing with arquillian

- AS7 supports easy testing with Arquillian
- Arquillian is used both in the internal test suite and by end users to test their applications
- Combined with the fast startup speed of AS7 testing in the container is just as easy as running normal JUnit tests.

References

- Ray Ploski: Jboss Application Server 7
  - http://www.slideshare.net/rayploski/jboss-application-server-7

OSGi

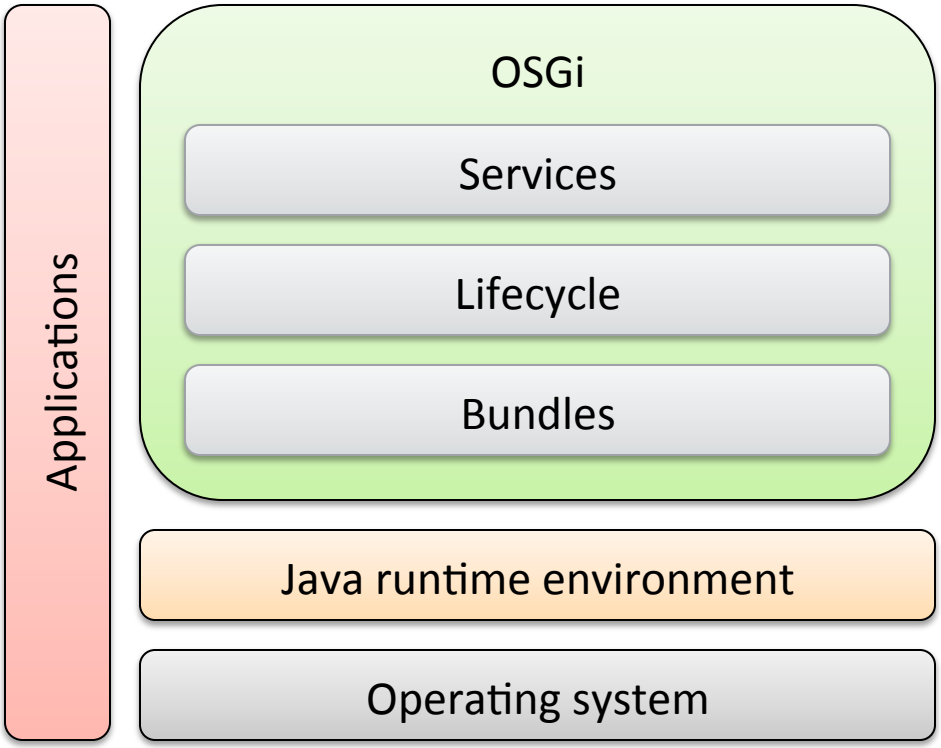


OSGi

- „A dynamic module system for Java”
- OSGI szövetség (www.osgi.org)→ ~30 teljes tag (Nokia, IBM, NTT, Motorola, stb.)

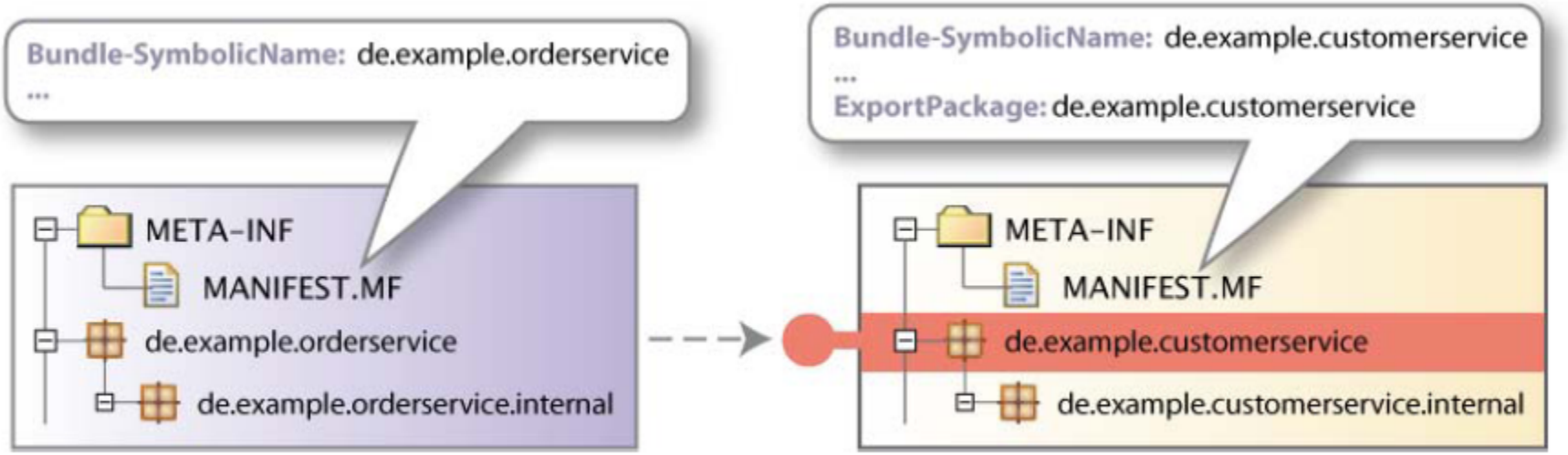
- Közös problémák (integráció, verziózás, élet ciklus)
- Megoldás → OSGI szabvány (specifikáció)
- Komponens alapú
- Közös integrációs „primítívekkel” (interfészek, leírók, stb.)
- Jelenleg: 4.3 –es verzió (2011 április)

OSGi


Applications
OSGi
Services
Lifecycle
Bundles
Java runtime environment
Operating system

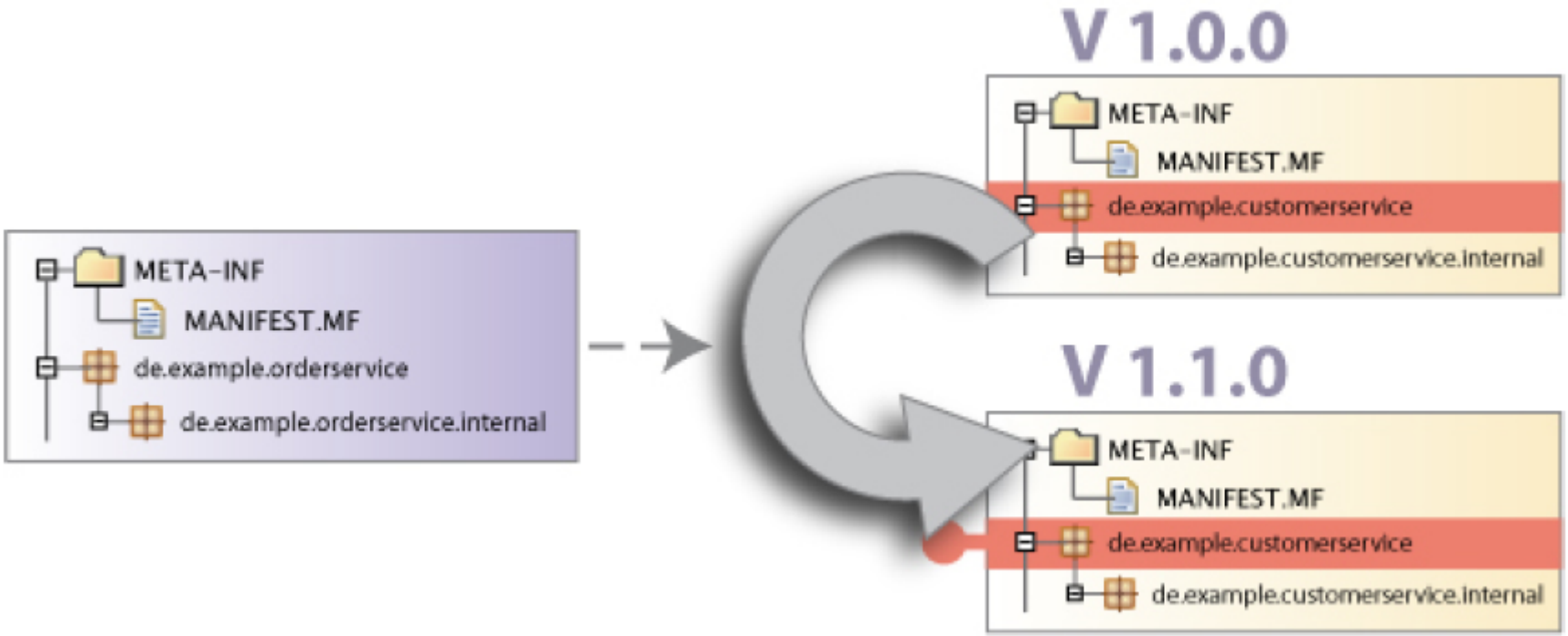
OSGi alapok: modulok

- Modulok (bundles)
  - Public és private API láthatósága
  - Függőség kezelés
  - Verziózás

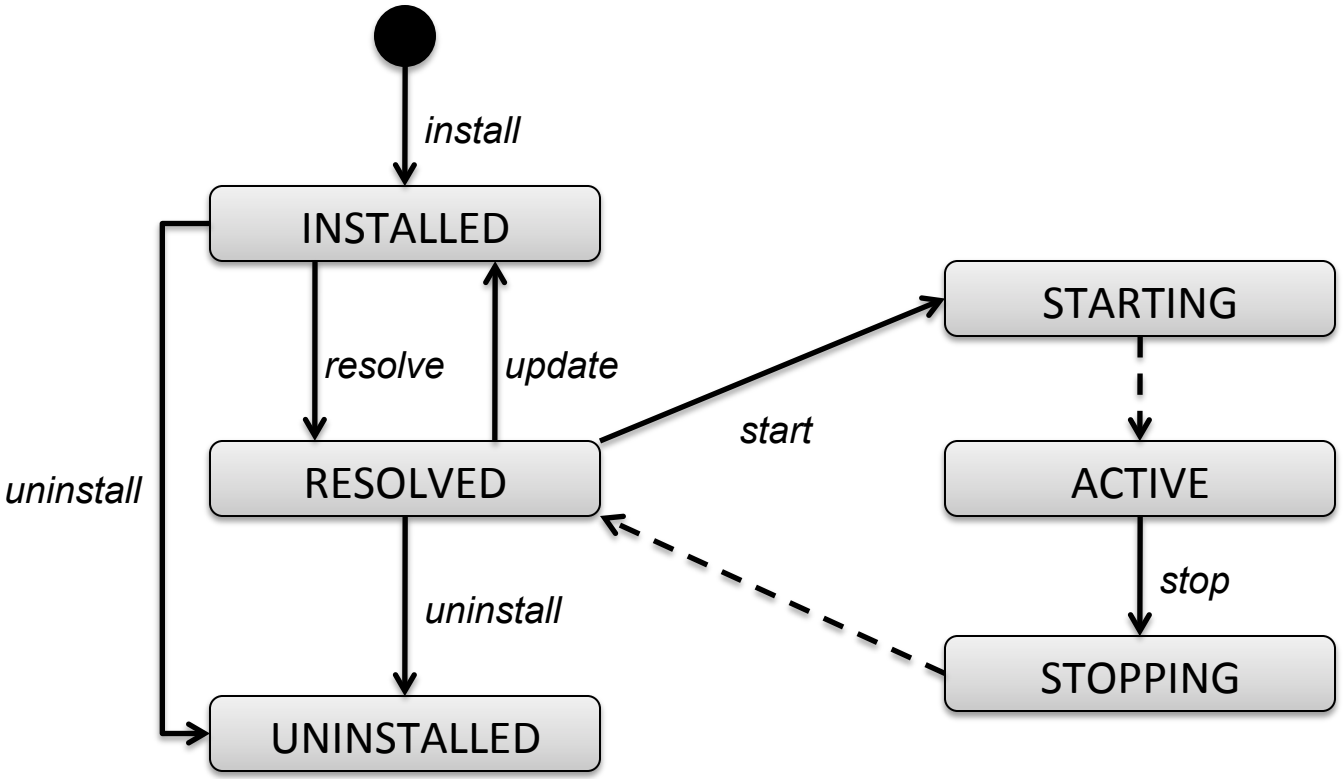


OSGi alapok: életciklus

- Életciklus (Life cycle)
- Dinamikus Bundle:
  - Betöltés (install)
  - Start/stop

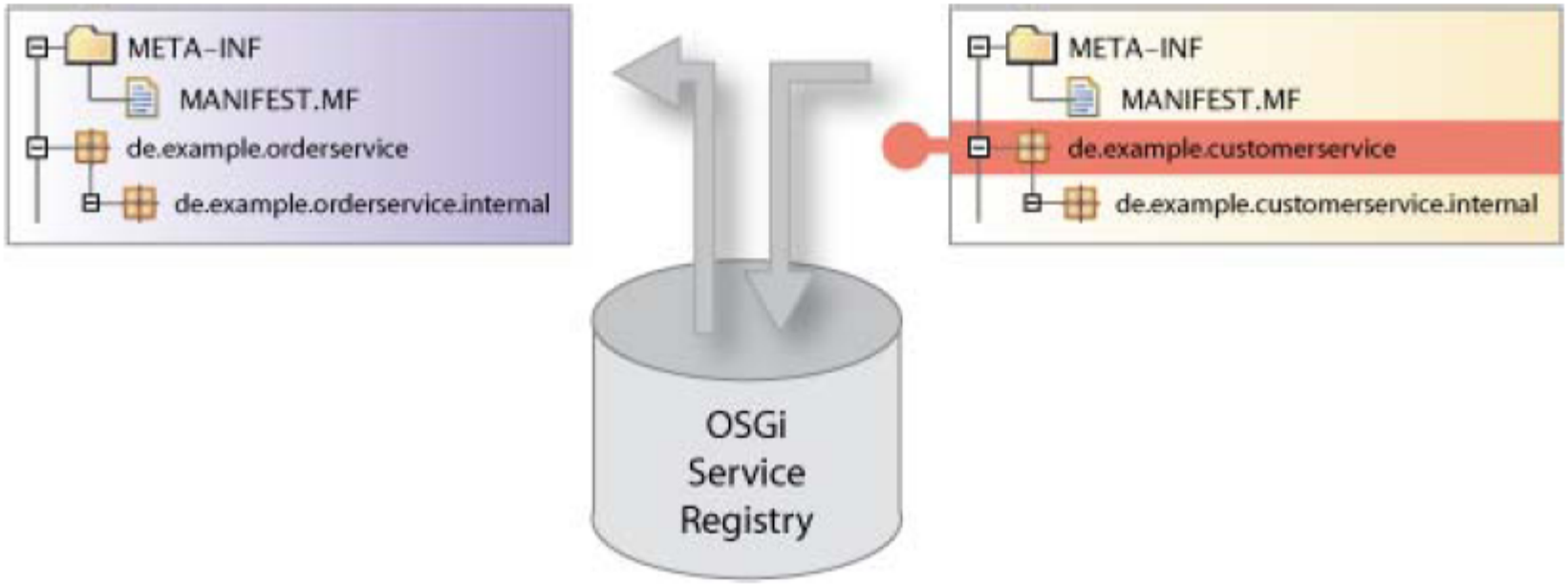


OSGi alapok: Bundle életciklus modell



OSGi alapok: szolgáltatások

- Szolgáltatás orientált (Services):
  - Bundle-ok kiajánlhatóak
  - Kereshetőek (service registry)
  - Futás idő közben megjelenhetnek(/eltűnhetnek)!



OSGi Bundles

OSGi belülről (Manifest.MF)

- Identifikáció:
  - Bundle-SymbolicName: org.eclipse.equinox.registry
  - Bundle-Version: 3.2.100.v20060918
  - Bundle-Name: Eclipse Extension Registry
  - Bundle-Vendor: Eclipse.org
- ClassPath:
  - Bundle-ClassPath: ., someOtherJar.jar
- Életciklus:
  - Bundle-Activator: org.eclipse.core.internal.registry.osgi.Activator
- Függőségek:
  - Import-Package: javax.xml.parsers,
  - org.xml.sax,
  - org.osgi.framework;version=1.3
  - Require-Bundle: org.eclipse.equinox.common;bundle-version="[3.2.0,4.0.0)"
  - Bundle-RequiredExecutionEnvironment: CDC-1.0/Foundation-1.0,J2SE-1.3
- Kiajánlás (export)
  - Export-Package: org.eclipse.equinox.registry

Modul réteg

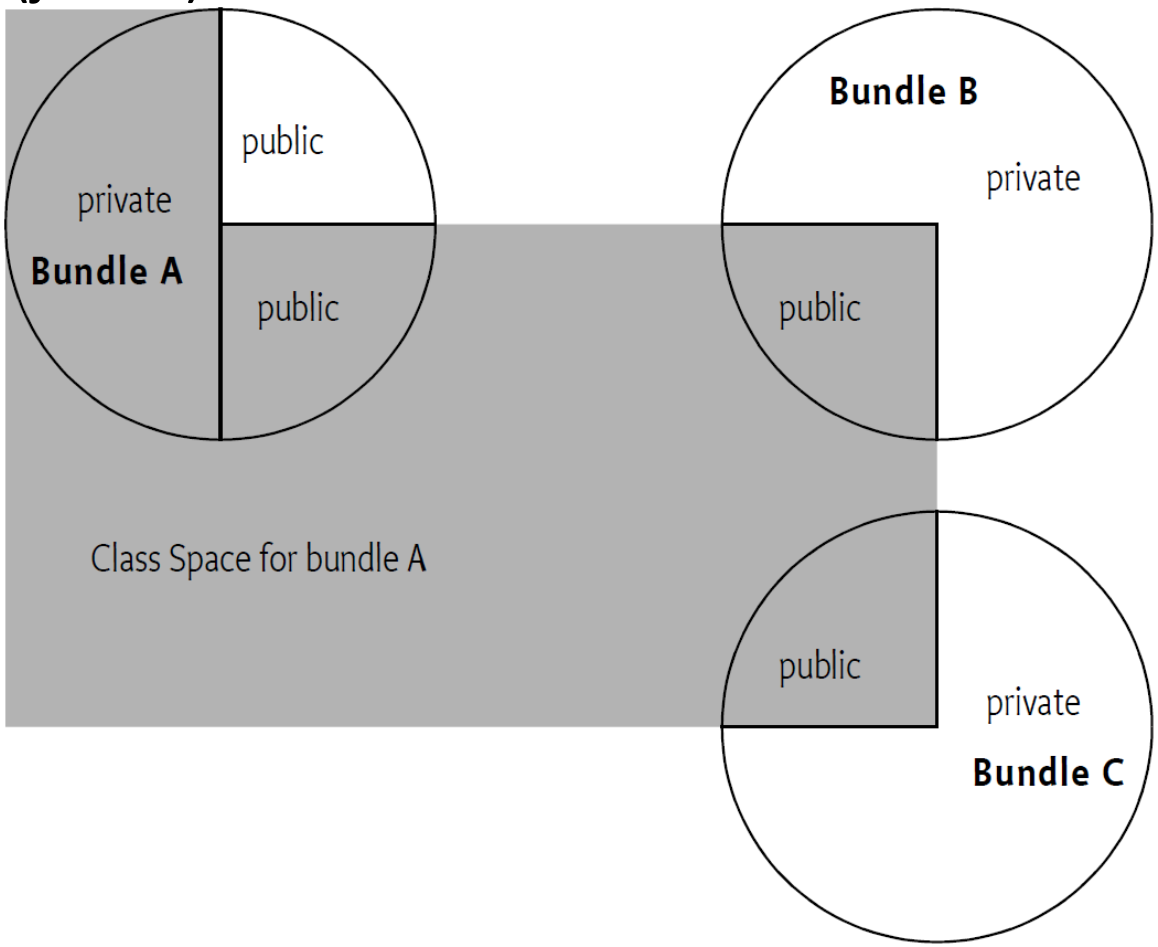
- A modulok indíthatók, leállíthatók
- A futó bundle-k szolgáltatásai kiajánlásra kerülnek
- Fontos manifeszt adatok
  - Bundle-activator: az életciklus menedzselését végző osztály neve
  - Bundle-classpath: a bundle-n belüli classpath-ok listája. A default értéke: . (a bundle root)
  - Bundle-name: olvasható név
  - Bundle-SymbolicName: azonosító (egyedi)
  - Bundle-UpdateLocation: URL, ahonnan a firssítések letölthetőek
  - Export-Package: a kiajánlott java csomagok listája
  - Import-Package: importált csomagok listája
  - Require-Bundle: szükséges modulok listája (import + függőségek)

Class loading

- Minden bundle egy vm-en belül fut
- Minden bundle-nak saját class loadere van
  - 3 helyről tölthet be osztályokat/erőforrásokat
    - Boot class path: java.* csomagok és implementációik
    - Framework class path: a framework-nek saját class loadere van, amitől elkérhetőek az interfészek és implementáló osztályok
    - Bundle space: a bundle jar fájljai, valamint a hozzá kötődő egyéb jar-ok

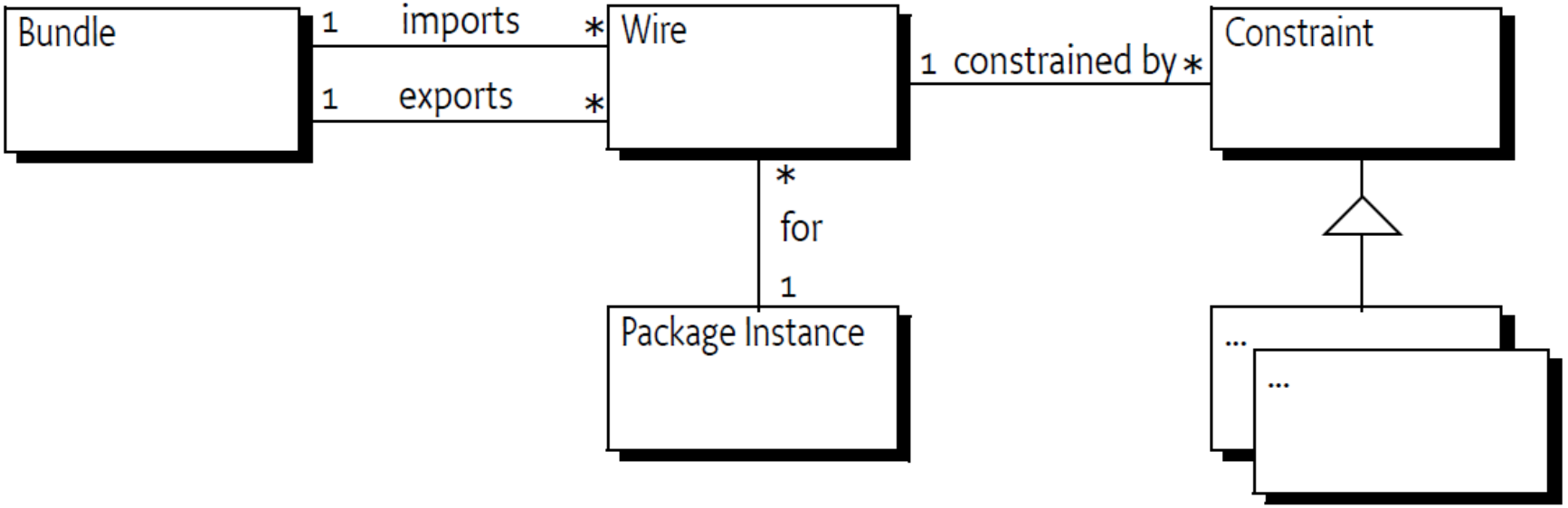
Class loading – Class space

- Egy bundle class space-e
  - A szülő class loader-e (java.*)
  - Importált csomagok
  - Függőségek
  - A bundle privát classpath-a
  - Csatolt fragmensek



Bundle feloldás

- Feloldás: az importerek és exporterek összekötése
  - Kényszereknek megfelelően
- Vezeték (wire): összeköttetés importer és exporter között



Metaadat feloldás

- Bundle-SymbolicName
  - kötelező, egyedi azonosító
  - Ha két egyező nevű és verziójú van, a második telepítése sikertelen
  - Paraméterek
    - Singleton: csak egyetlen verziója lehet betöltve
    - Fragment-attached: definiálja, hogyan lehet fragmenseket hozzákapcsolni
      - Always: bármikor kapcsolódhat
      - Never: nem lehetséges
      - Resolve-time: csak a resolve fázisban
  - Példa: `Bundle-SymbolicName: com.acme.foo;singleton:=true`
- Bundle-Version
  - Meghatározott formátum: major.minor.micro.qualifier
  - Összehasonlítás hierarchikus
    - Numerikusan, illetve a qualifier esetén String.compareTo
    - Két verzió akkor azonos, ha minden szegmensük egyezik
  - Példa: `Bundle-Version: 22.3.58.build-345678`

Metaadat feloldás

- Imported-packages
  - Importált csomagok listája
  - Resolution – a csomagot fel kell oldani kötelező import esetében, ha ez sikertelen a bundle sem töltődhet be
  - Version – verzió intervallum a csomagot exportáló csomagra zárt [], nyitott (), pl. [1.0.0,2.0.0)
  - Bundle-version: az exportáló bundle verziója
  - Bundle-symbolic-name: az exportáló bundle neve
  - Példa:

```
Import-Package: com.acme.foo;com.acme.bar;
        version="[1.23,1.24]";
        resolution:=mandatory
```

- Exported-packages
  - Exportált csomagok listája
  - Hasonlóan az Imported-packages-hez
  - Példa:

```
Export-Package: com.acme.foo;com.acme.bar;version=1.23
```

OSGi Services

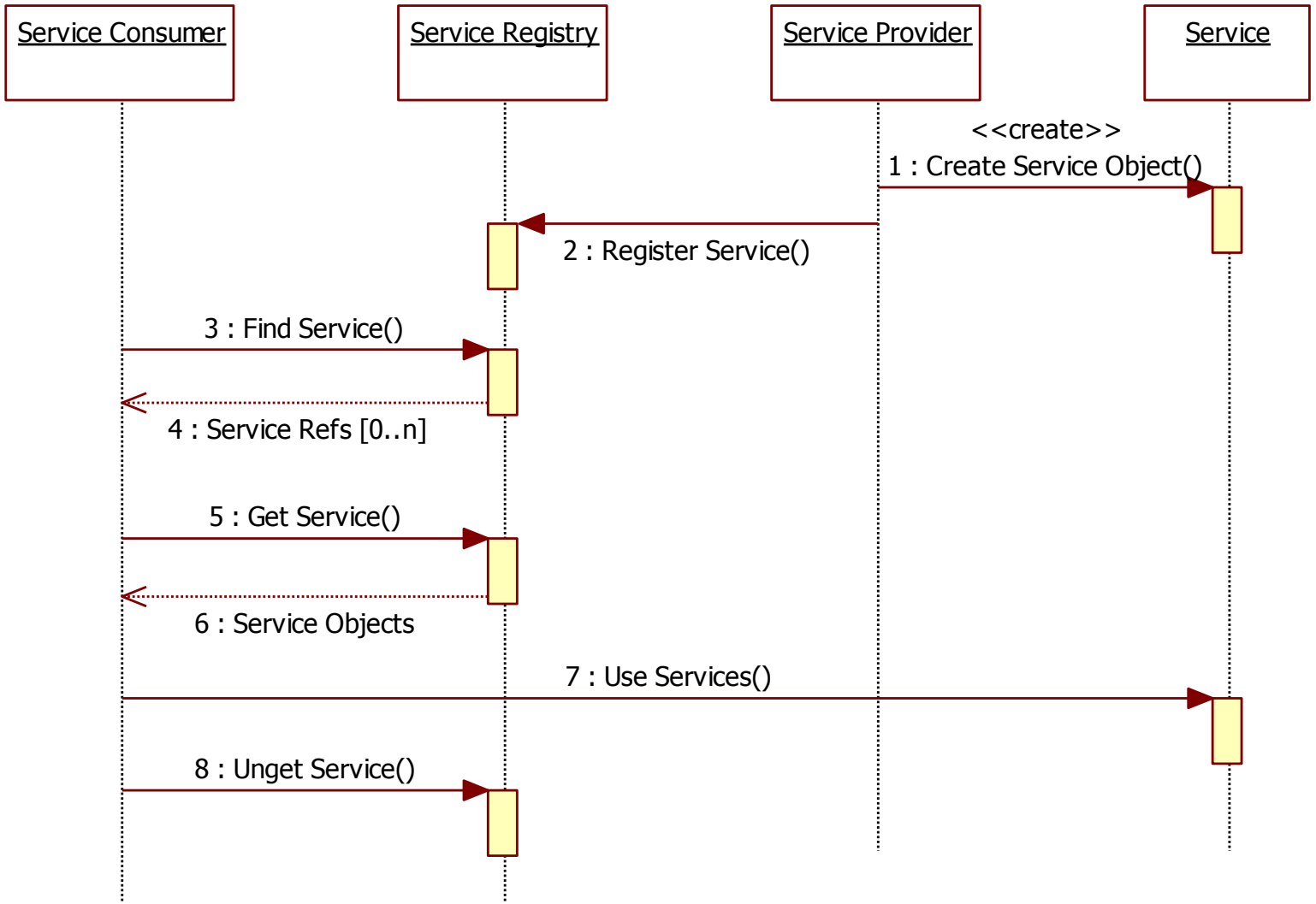
Szolgáltatási réteg

- Definiálja az együttműködési modellt
  - „Publish, find, and bind”
  - A szolgáltatás egy normál java objektum
  - Regisztrálódik egy vagy több java interfész alatt
- A bundle-k
  - Regisztrálhatnak
  - Kereshetnek
  - Használhatnak szolgáltatásokat
  - Illetve, ezekkel kapcsolatban eseményeket kezelhetnek

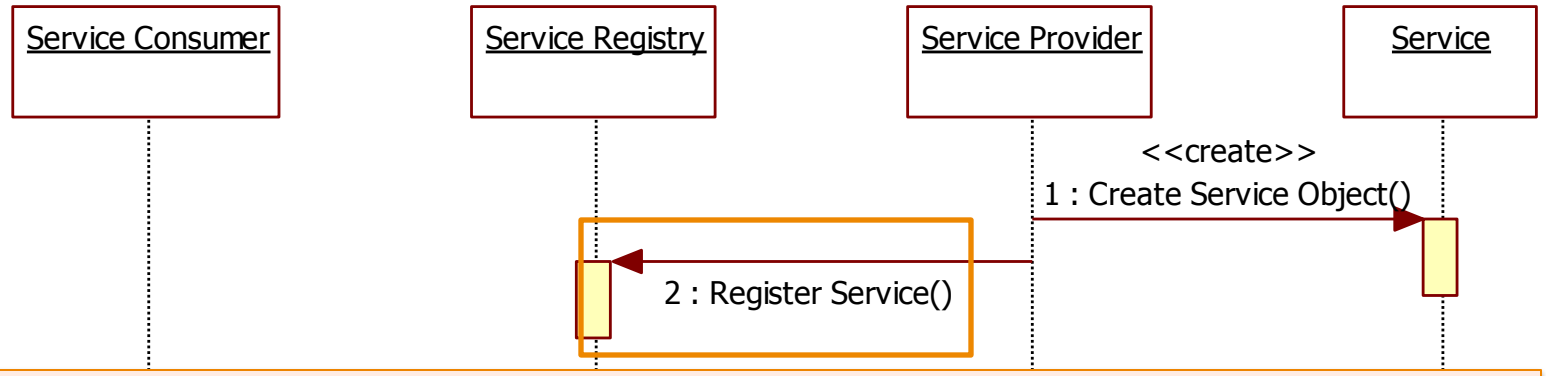
Szolgáltatási réteg - alaptulajdonságok

- Kollaboratív: bundle-k közötti együttműködés megvalósítása
- Dinamikus: futásidejű változások
  - Új szolgáltatások megjelenése
  - Régi szolgáltatások megszűnése
- Biztonságos: hozzáférés korlátozható
- Reflektív: teljes hozzáférés a réteg belső állapotához
- Verziókezelés: a szolgáltatások frissülhetnek
- Perzisztens id: framework indítások között is lehet a szolgáltatásokat követni

Szolgáltatások „életciklusa”



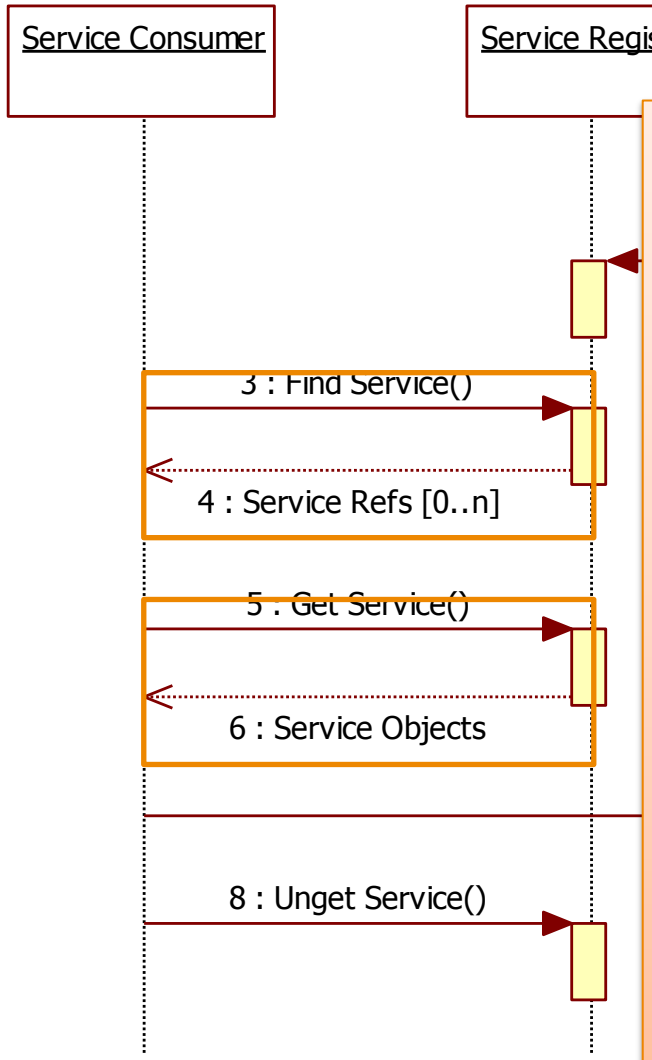
Szolgáltatások regisztrációja



- BundleContext
  - registerService(String, Object, Dictionary)
    - Egy adott interfész név alá regisztálja a szolgáltatást
  - registerService(String[], Object, Dictionary)
    - Több interfész név alá regisztálja a szolgáltatást
- Példa

```
service = new HelloServiceImpl();
// register the service
context.registerService(HelloService.class.getName(),
                service, new Hashtable());
```

Szolgáltatások regisztrációja



- BundleContext
  - getServiceReference(String)
    - Ha több van, a ranking dönt
  - getServiceReferences(String,String)
    - Az összes referencia az adott interfészhez és filterhez
  - getService(ServiceReference)
    - A szolgáltatás objektumot adja vissza
  - ungetService(ServiceReference)
    - Szolgáltatás „elengedése”
- Példa

```
ServiceReference reference = context.getServiceReference(
          HelloService.class.getName());
HelloService service = (HelloService) context.
          getService(reference);
```

Service Tracker

- Problémák a szolgáltatások közvetlen lekérésével
  - Nincs értesítés, ha egy szolgáltatás megszűnik/megjelenik
  - Alacsony szintű API
  - Minden alkalommal le kéne kérni a szolgáltatást, amikor használni akarjuk
  - Lehetőség még a ServiceListener használata, de könnyű hibázni
  - Körülményes, sok Java kód
- Megoldás: a ServiceTracker használata
  - Szolgáltatáshoz lehet regisztrálni
  - Jelez, ha
    - Megjelenik egy, az adott típusú szolgáltatás
    - Eltűnik egy, az adott típusú szolgáltatás
    - Módosul egy, az adott típusú szolgáltatás

Declarative Services

- DS előzménye: Service Binder
  - Humbertó Cervantes és Richard Hall fejlesztették ki
  - Szolgáltások függőségeinek automatikus menedzselése
  - Fejlesztő a szolgáltatások írására koncentrálhat (POJO)
- DS a Compendium services része R4 óta
  - XML leírók (komponens leíró)
  - „Publish, find, and bind” deklaratív módon
  - Válasszuk szét a felelőségeket:
    - Szolgáltatás implementáció ← továbbra is bundle felelőssége
    - Szolgáltatás regisztráció ← Service Component Runtime (SCR)
  - Dinamikus, mint a szervizek
  - „On demand” betöltés, mint az Eclipse Extension-ök.

DS – komponens leíró

- XML alapú
- Deklaratív módja a szervizek regisztrálásának, kötésének
- OSGI-INF könyvtárban
- Több komponens egy bundle-ban
- MANIFEST.MF-ban összeset fel kell venni
  - Service-Component

Példa komponensleíróra

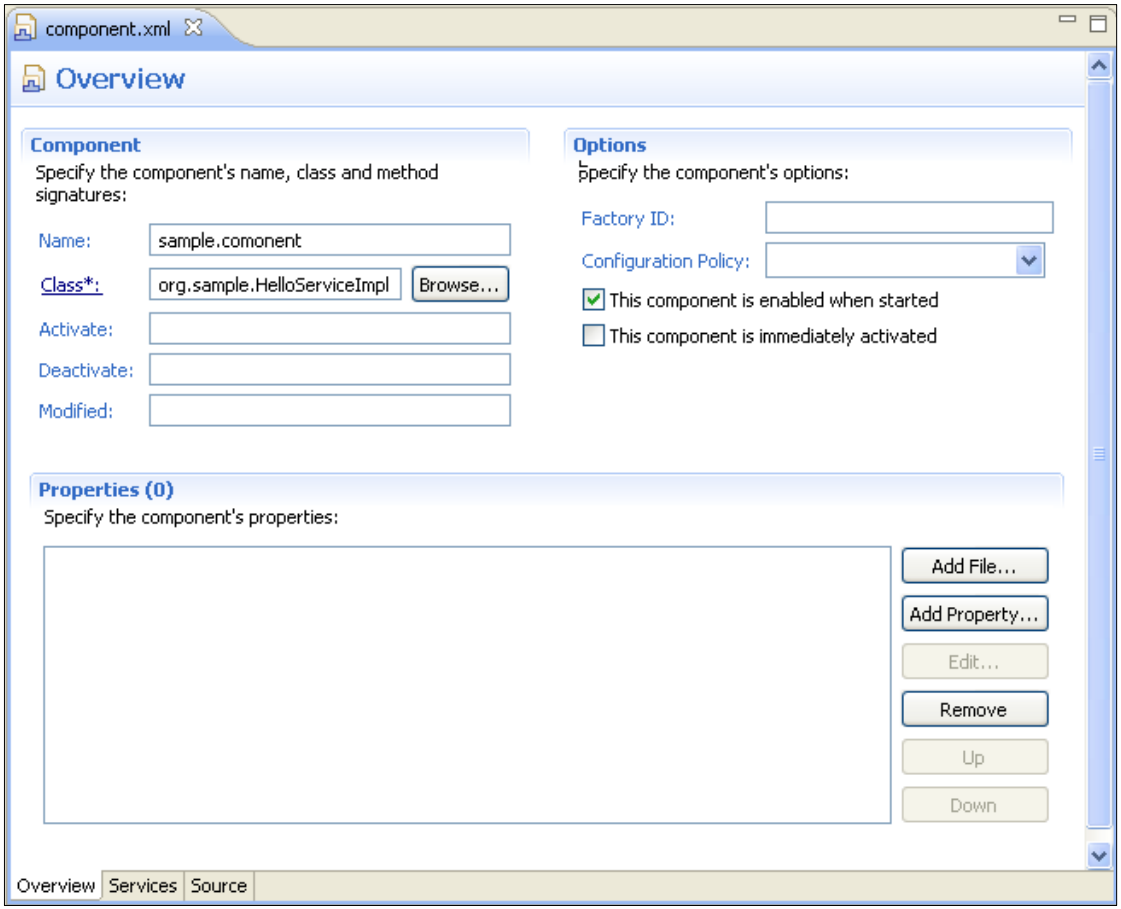
```
<scr:component xmlns:scr="http://www.osgi.org/
   xmlns/scr/v1.1.0"
    name="sample.component">
    <implementation
   class="org.sample.HelloServiceImpl"/>
    <service>
      <provide interface="org.sample.HelloService"/>
    </service>
    <reference
       bind="setService"
       unbind="unsetService"
       cardinality="0..1"
       interface="org.sample.ServiceForHello"
   name=„SERVICEFORHELLO"
    policy="dynamic"/>
</scr:component>
```

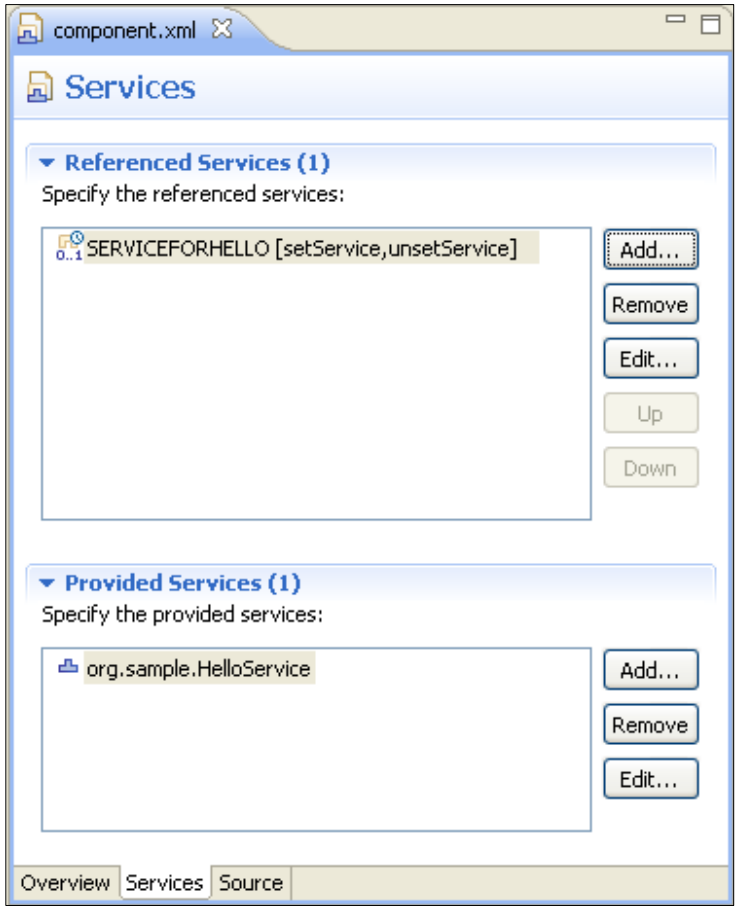
Kiajánlott szolgáltatás definíciója

Felhasznált szolgáltatás

DS – komponens leíró

- Eclipse támogatás: Declarative Service Tooling
  - Component Definition Editor





DS – komponensek összekapcsolása

- 3 dimenzió
  - Opcionális / Kötelező
  - Egy értékű / Több értekű
  - Statikus / Dinamikus
- Első két dimenziót adja a számosság (cardinality):
  - 0..1 → opcionális, egy értékű
  - 1..1 → kötelező, egy értékű
  - 1..n → kötelező, több értékű
  - 0..n → opcionális, több értékű
- Harmadik dimenziót pedig a policy
  - dynamic: a szolgáltatás menet közben „kicserélhető”
  - static: garantáltan egy szolgáltatás lesz végig (pl. állapottal rendelkező szolgáltatás esetében)

```
<reference
      bind="setService"
      unbind="unsetService"
      cardinality="0..1"
      interface="org.sample.ServiceForHello" name=„SERVICEFORHELLO"
  policy="dynamic"/>
```

DS – komponensek összekapcsolása

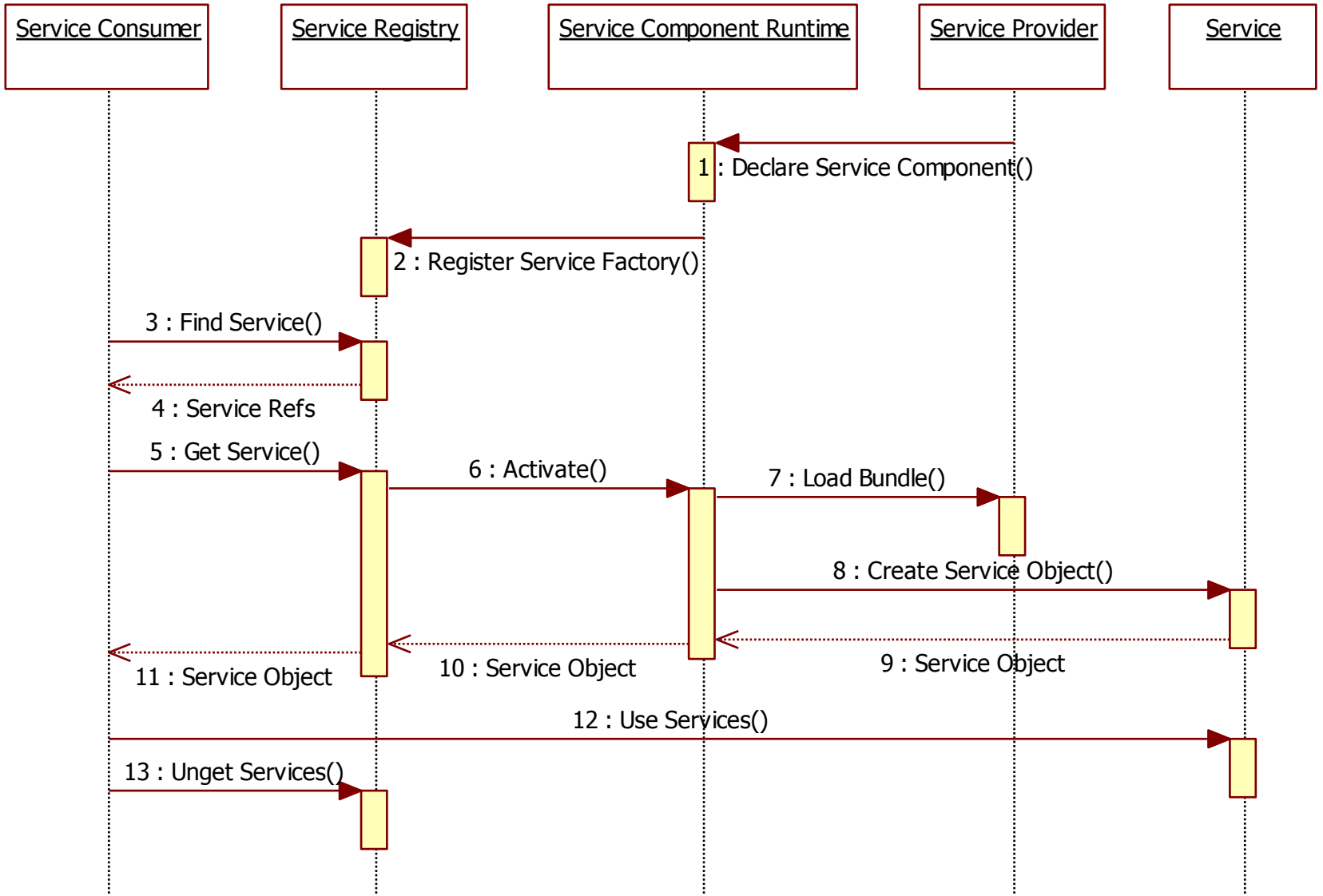
- 3 dimenzió
  - Opcionális / Kötelező
  - Egy értékű / Több értekű
  - Statikus / Dinamikus
- Első két dimenziót adja a számosság ***(cardinality)***:
  - 0..1 → opcionális, egy értékű
  - 1..1 → kötelező, egy értékű
  - 1..n → kötelező, több értékű
  - 0..n → opcionális, több értékű
- Harmadik dimenziót pedig a ***policy***
  - dynamic: a szolgáltatás menet közben „kicse
  - static: garantáltan egy szolgáltatás les gig (pl. állapottal rendelkező szolgáltatás esetében)

```
<reference
      bind="setService"
      unbind="unsetService"
      cardinality="0..1"
      interface="org.sample.ServiceForHello" name=„SERVICEFORHELLO"
   policy="dynamic"/>
```

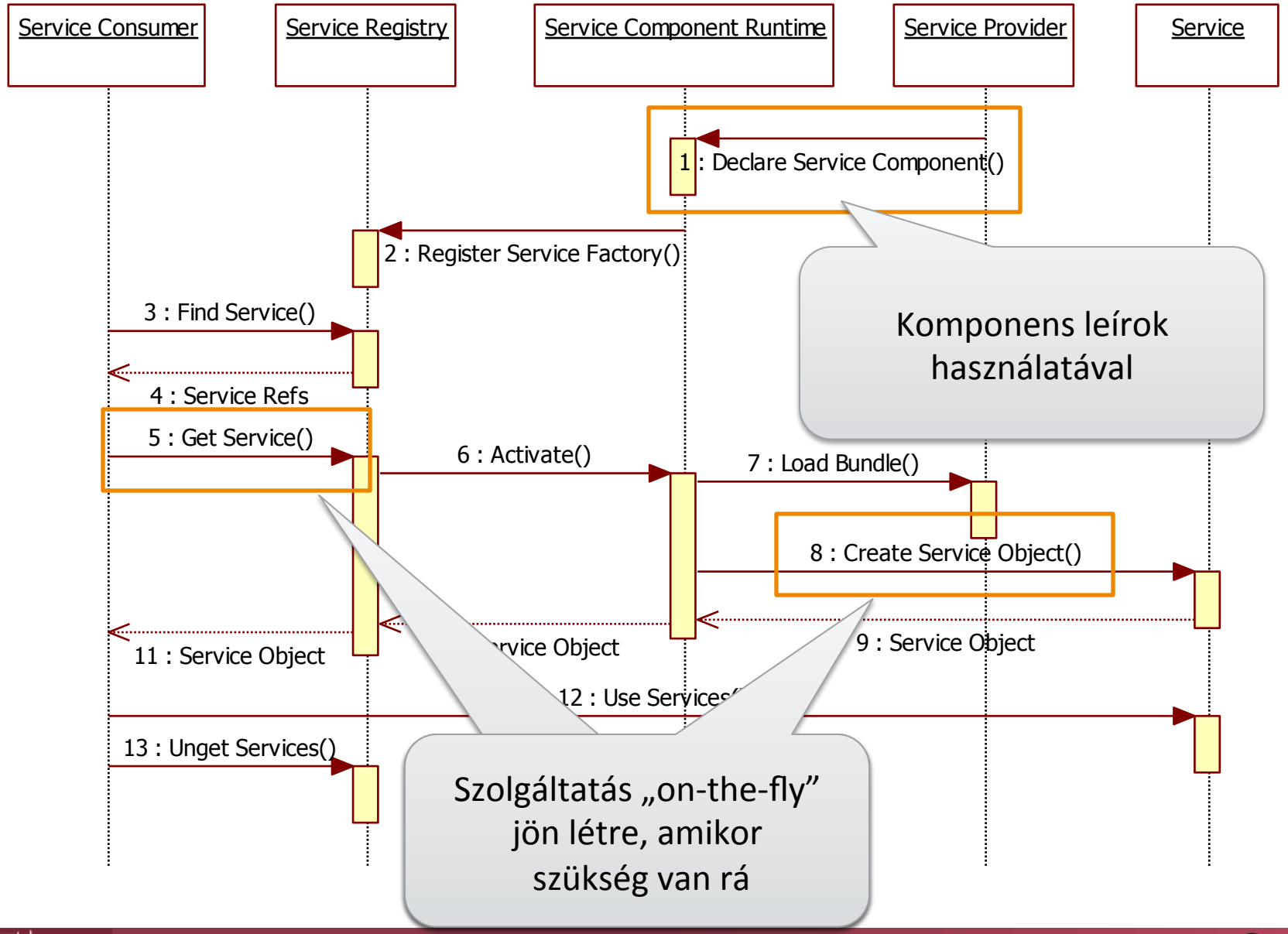
Szolgáltatás megjelenése esetén hívandó metódus

Szolgáltatás megszűnése esetén hívandó metódus

DS – Életciklus menedzselés



DS – Életciklus menedzselés



Http services

- Egyike a legrégebbi, legelterjedtebb szolgáltatásnak
- OSGi alapú webes komponensek fejlesztése
- Jelenleg támogatott komponensek
  - Servlet-ek regisztrálsa
    - Servlet-ek regisztrálása on-the-fly
    - OSGi servlet-be csomagolva
  - Erőforrások regisztrálása (HTML fájlok, képek, stb…)
- Alkalmazások
  - Pl. Apache Felix Web Console: OSGi konténerek monitorozása

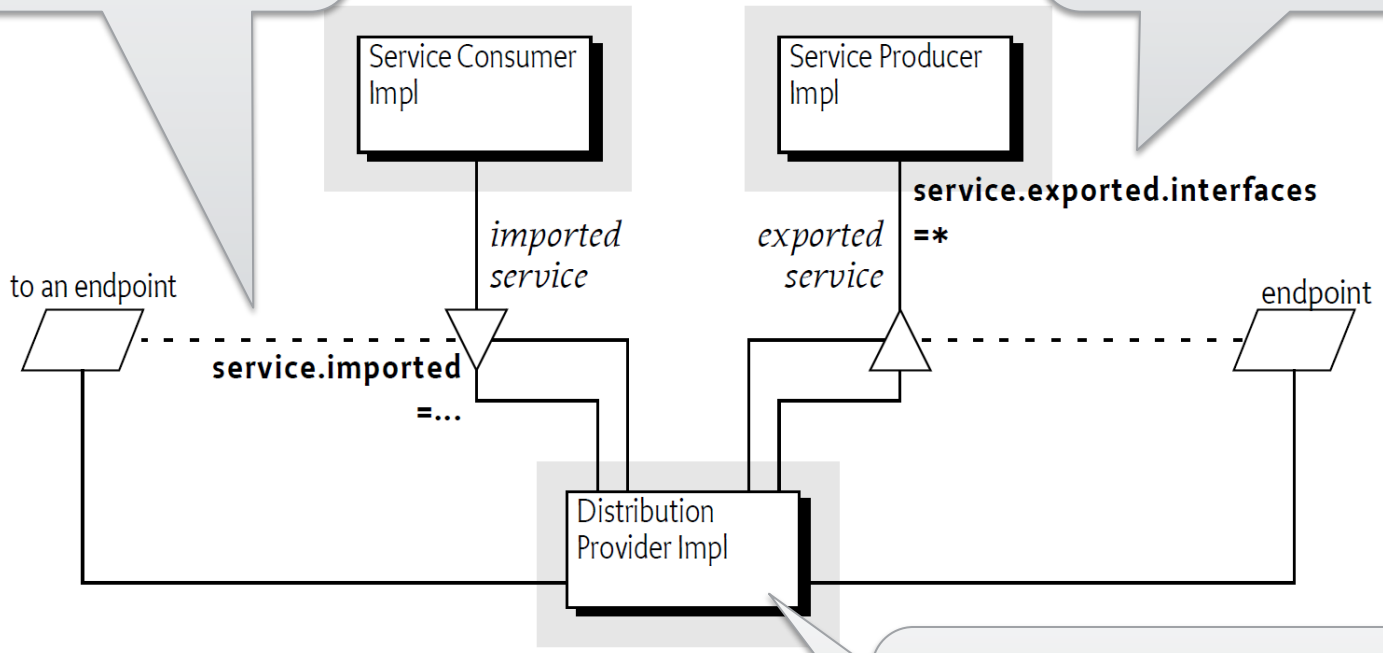
Remote Services

- Követelmények
  - Átlátszóság: nincs különbség helyi és távoli szolgáltatások között
  - Általános: Ne állítson korlátokat az elosztottsággal
  - Konzisztens viselkedés: helyi és távoli szerver ugyanúgy viselkedjen
- Elosztott alkalmazások készítése

Remote Services

Property-ben a importált szolgáltatás

Property-ben a kiexportált szolgáltatás



- Átlátszó módon
- Szolgáltatások proxy-ként

Eclipse Equinox

OSGi implementációk

- Open Source
  - **Eclipse Equinox (http://www.eclipse.org/equinox/)**
  - Apache Felix (http://felix.apache.org/)
  - Knopflerfish (http://www.knopflerfish.org/)
  - ProSyst mBedded Server Equinox Edition (http://www.prosyst.com/products/osgi_se_equi_ed.html)
- Fizetős:
  - ProSyst (http://www.prosyst.com/)
  - Knopflerfish Pro (http://www.gatespacetelematics.com/)

OSGi, Eclipse és Equinox viszonya

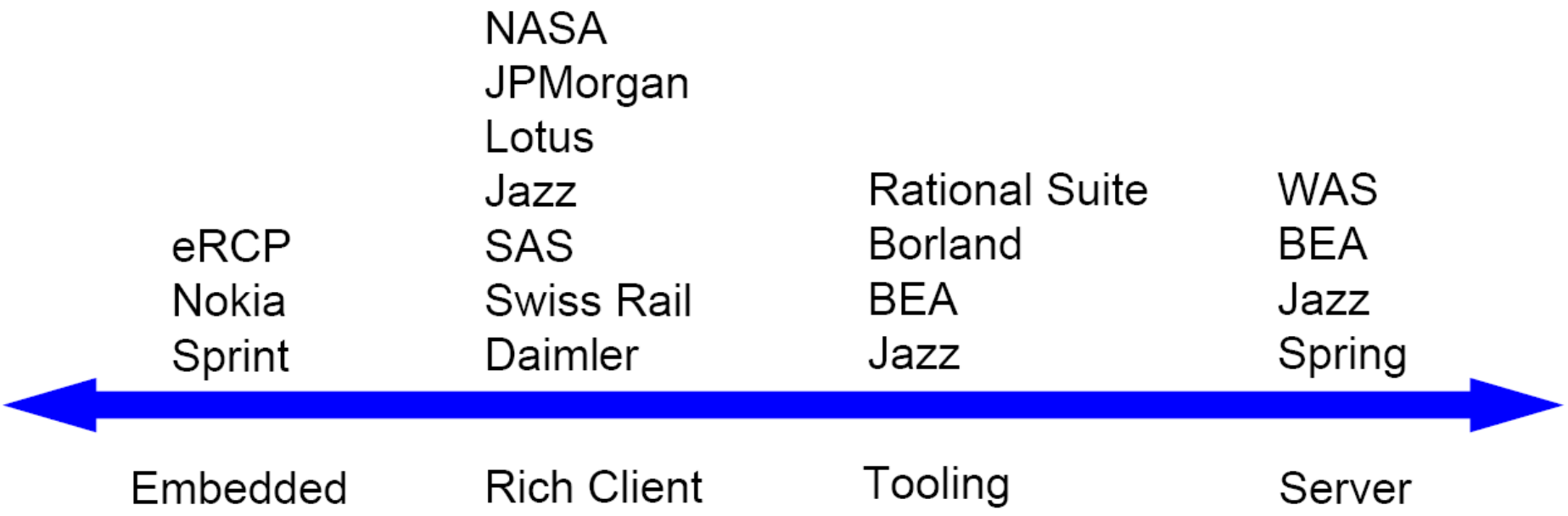
- OSGi szövetség
  - Nyílt szabvány
  - Komponens alapú leírások
  - Egyre szélesebb alkalmazási kör (mobil, szerver, desktop, vállalati, beágyazott)
- Eclipse
  - RCP használata nagyon megnőtt
  - Eclipse runtime lecserélése nyílt szabványra →
  - Eclipse 3.0 óta OSGi-ra épül
- Equinox
  - Eclipse OSGi implementációja (3.3 óta)
  - „Szerver oldali eclipse” → több ennél
  - OSGi 4.0 és 4.1 referencia implementáció

Equinox „Runtime”

- OSGi 4.3 ref implementáció + Eclipse Extension mechanizmusa!
- Több mint szerver oldali Eclipse! (eRCP, RCP, server side,…)
  - Extension és extension point definíciók
  - Erős support SDK oldalon
  - Add-on:
    - Admin
    - Security
    - Application container
    - …

Hol használják

- Eclipse IDE és



Equinox mint szerver

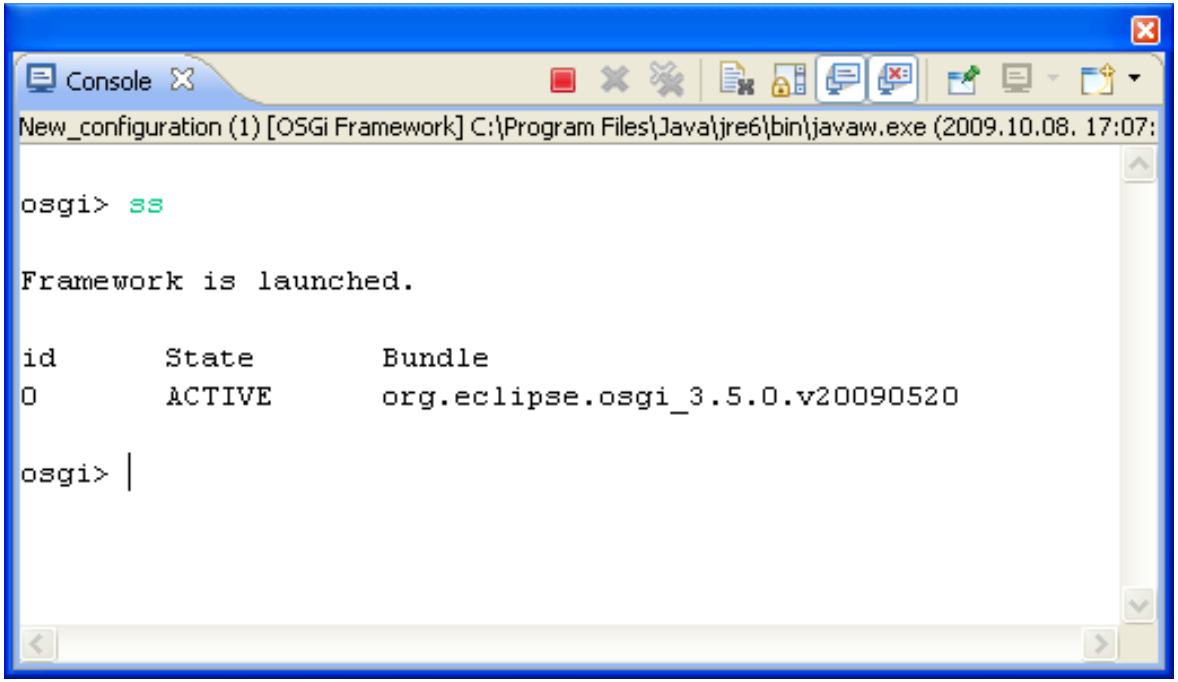
- Equinox runtime + szerver oldali add-ons
  - HTTP service/registry
  - Jetty: beépített pehelysúlyú webszerver
  - Integrációs bundle-ok (ServletBridge=Servlet/JSP, stb.)

Előnyei

- Az egyes komponenseknek inkrementális frissítése
- Többszörös instance futtatás párhuzamosan → HA / nagy sebesség
- Management Külön-Külön és akár együttesen is (szinteken)
- Más-más igényekhez másként paraméterezett instancek
- Class loading teljesítmény növekedés
- Komponens megosztás server és kliens között
  - „Hálózat nélküli” mód támogatása

OSGi konzol

- OSGi prompt ~ Hasonló egy DOS v. Bash prompt-hoz
- Eclipse támogatás
  - Console view
  - Highlight



OSGi konzol - parancsok

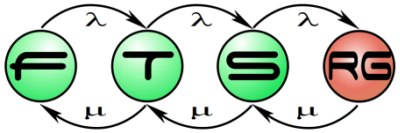
- Hasznos parancsok
  - ss: kilistázza az telepített bundle-okat.
  - start <id>: elindítja a megadott azonosítójú bundle-t
  - stop <id>: leállítja a megadott azonosítójú bundle-t
  - install file:<path>: telepíti a megadott „bundle”-t
  - uninstall <id>: eltávolítja a megadott bundle-t
  - update <id>: frissíti a megadott bundle-t
  - services <filter>: kilistázza a futó szolgáltatásokat
    pl.: osgi> services (objectClass=*HelloService)
  - shutdown: a futó osgi framework leállítása
  - close: shutdown és exit
  - exit: ~ System.exit

References

- Ráth István, Ujhelyi Zoltán, Horváth Ákos és mások: Eclipse alapú technológiák és Eclipse alapú fejlesztés és integráció c. tárgyak anyagai (2010-2012)
  - http://eclipse.inf.mit.bme.hu

Objektum-relációs leképezés

JPA



Object-Relational Mapping

- Cél: objektumok tárolása adatbázisban
- Probléma: az objektum-orientált és a relációs adatmodell közötti különbségek
- Megoldás: automatizált leképezés

Object-Relational Mapping

- Leképezés:

Osztályhierarchia	Séma
Osztály	Tábla
Mező	Oszlop
Objektum	Sor

Object-Relational Mapping

- Példa:

```
class Book {
      int id;
      String title;
      String
author;
}
```

```
CREATE TABLE Book (
      INTEGER id,
      VARCHAR(255) title,
      VARCHAR(255)
author
);
```

Problémás esetek

- Hivatkozáshoz objektumoknál van referencia, ehelyett kell elsődleges kulcs
- Öröklés
  - Többféle stratégia, ld. később
- Eltérő adattípusok
- Mikor szinkronizáljunk a memóriában levő modell és az adatbázis között?

ORM keretrendszerek

- Sok különböző
  - ActiveObjects
    - Öröklés + annotációk
  - Torque
    - Kódgenerálás XML leíró fájlokból
  - **JPA**
    - Annotációk és/vagy XML

Java Persistence API

- EJB 3 specifikáció része
- Elfedi az adatbázisszerver-specifikus részleteket
- Futásidőben átlátszóan setterekkel tudjuk módosítani az objektumokat

JPA providerek

- A JPA csak egy API specifikáció
- Megvalósítását providerek biztosítják
  - Hibernate
  - OpenJPA
  - Toplink
  - **EclipseLink**

A JPA használata

- Java osztályok (POJO) annotációkkal
  - Alternatív módon: leképezés adatok XML-ben
    - Felülírja az annotációkat
    - Nem használjuk
- Alapelem: Entity = perzisztens osztály
- Perzisztens modul minden olyan jar, amelynek META-INF könyvtárában van `persistence.xml` fájl, amiben pl. a provider kerül definiálásra
- `javax.persistence` csomag tartalmazza

Entitás megadása

- Java osztály `@Entity`-vel (`javax.persistence.Entity`) annotálva, default konstruktorral
- Általában szerializálható (`implements Serializable`)
- Kötelező elsődleges kulcs attribútum: `@Id`
  - többféle automatikus ID-generáló stratégia megadható a `strategy` paraméterben

Entitás attribútumai

- A perzisztens attribútumokat a kliensek getterek/setterek formájában érhetik el (JavaBean konvenció)
- Nem perzisztens (tranziens) attribútum: @Transient
- Attribútum típusa lehet:
  - primitív típus
  - `String, BigInteger, BigDecimal, java.util.Date, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.Timestamp, byte[], Byte[], char[], Character[]`
  - `Enum`
  - más entitás, más entitások gyűjteménye
  - beágyazott osztály

A leképezés testreszabása

- Az entitás, illetve az attribútumok nevével azonos a default tábla- és oszlopnév, ez felülírható
- `@Table(name="MyTable")`
  - `@SecondaryTable(s)` annotációval több táblába is szétszedhető
- `@Column(name="MyColumn")`
- Az oszlopoknak egyéb paraméterei is vannak
  - `nullable`
  - `unique`
  - `length`

Öröklés

- Fontos OO koncepció, melynek leképezése relációs adatbázisra nem triviális
- EJB3-tól kezdve
- Lehetséges leképezési módok:
  - egy tábla egy osztályhierarchiához
  - külön tábla gyermekosztályonként, hivatkozással
  - egy tábla egy konkrét osztályhoz

Egy tábla egy osztályhierarchiához

- egy táblában minden gyermekosztály
- diszkriminátor oszlop írja le a típust
- + hatékony (nem kell join)
- + polimorfizmust támogatja
- - mély hierarchia eseten sok, fölösleges oszlop
- - a gyermekosztályok attribútumainak megfelelő oszlopoknak nullázhatóknak kell lenniük

Egy tábla egy osztályhierarchiához

- A hierarchia legfelső szintjén:
  - @Inheritance(strategy=InheritanceType.SINGLE_TABLE)
  - @DiscriminatorColumn(name=<*oszlopnév*>)
- A hierarchia összes osztályánál:
  - @DiscriminatorValue(<*típusra utaló érték*>)

Külön tábla gyermekosztályonként

- Az ősosztályban definiált oszlopok egy táblában
- A gyermekosztályokban definiált oszlopok külön táblákban + idegen kulcs az ősre
- + nincsenek fölösleges oszlopok
- + definiálható nem nullázható oszlop
- + polimorfizmust támogatja
- - mély hierarchia esetén a sok join rontja a teljesitményt
- @Inheritance(strategy=InheritanceType.JOINED)

Önálló táblák gyermekosztályonként

- Külön tábla minden altípushoz
- Minden tábla tartalmazza az ősosztály attribútumait is
- + hatékony
- - polimorfizmus támogatása nehézkes (oszlopdefiníciók többször szerepelnek)
- az EJB 3 nem követeli meg a támogatását

Egyéb öröklési lehetőségek

- Entitás származhat nem entitásból
  - @MappedSuperClass-szal megjelölve az ősosztályt, annak nem lesz külön táblája egyik leképezési stratégia esetén sem, de az ottani attribútumok adatbázisba kerülnek
  - nem megjelölve egyáltalán nem kerülnek bele
- Nem entitás származhat entitásból
- Entitás lehet absztrakt
  - nem példányosodhat, de le lehet képezni táblába
  - le lehet kérdezni

Relációk

- Számosság szerint négyféle:
  - @OneToOne
  - @OneToMany
  - @ManyToOne
  - @ManyToMany
- Irány szerint kétféle:
  - Egyirányú
  - Kétirányú (a kapcsolat mindkét végén levő entitásnak lesz kapcsolatmenedzselő getter/setter metódusa): mappedBy paraméter
- Kétirányú OneToMany = Kétirányú ManyToOne
- A kapcsolatnak mindig egy tulajdonos oldala van

Példa reláció definiálására

- Tulajdonos oldalon (Employee):

```
@ManyToOne
@JoinColumn(name="company_id")
private Company company;
```

- Másik oldalon (Company):

```
@OneToMany(mappedBy="company_id")
private Collection<Employee> employees;
```

- + getterek, setterek
- a @JoinColumn helyett @JoinTable használandó, ha külön tábla tartja nyilván a kapcsolatot (pl. ManyToMany esetén mindenképpen)
- A @ManyToOne kötelezően tulajdonos oldal, mert nincs `mappedBy` paramétere, többi esetben szabadon választható a tulajdonos oldal

Relációk kaszkádosítása

- Mind a 4 kapcsolatdefiniáló annotációhoz megadható egy cascade elem, pl.

```
@OneToMany(cascade={
CascadeType.PERSIST,
CascadeType.MERGE})
```

- Lehetséges értékek:
  - PERSIST
  - MERGE
  - REMOVE
  - REFRESH
  - ALL
- Azt adja meg, milyen műveletek hívódjanak meg a kapcsolódó entitásokra is
- Default: nincs cascade

Fetch

- Mind a 4 kapcsolatdefiniáló annotációhoz megadható egy cascade elem, pl.
  `@OneToMany(fetch=FetchType.LAZY)`
- Azt adja meg, hogy egy entitás betöltésekor betöltődjenek-e a kapcsolódó entitások is
- **LAZY** (lusta): nem töltődnek be, csak ha hivatkozunk rájuk
  - így nem foglal memóriát, csak ha szükség van rá, de +1 lekérdezés
- **EAGER** (mohó, ez a default): kezdetben betöltődnek
  - gyorsabb, de több memóriát foglal
- Finomhangolási lehetőség:
  - legyen LAZY, de azokban a lekérdezésekben, ahol tudjuk, hogy szükség lesz a kapcsolódó entitásokra, használjunk fetch join-t az EJB-QL queryben, pl.

```
SELECT c form Customer c LEFT JOIN FETCH c.orders
```

Lazy fetch problémák

- Lecsatolt állapotban csak a korábban már hivatkozott kapcsolódó objektumokat fogja tartalmazni az entitás
- Ha egy kapcsolatait a lecsatolódás miatt részben elveszítő entitást vissza-merge-elünk, az adatbázisban is törlődnek a kapcsolatok
- A Lazy csak tanács, a persistence provider dönthet úgy, hogy mégis betölti mohó módon a kapcsolatokat

Perzisztencia kontextus

- A persistence provider által kezelt entitások egy halmaza
- Azonosítás a persistence unit nevével
- Entity manager lekérése pl.:

```
EntityManagerFactory factory =
  Persistence.createEntityManagerFactor
  y(
  PERSISTENCE_UNIT_NAME);
EntityManager entityManager =
  factory.createEntityManager();
```

Entity Manager

- Entitások kezeléséért felelős
- Használata:
  - Entitások életciklusának kezelése
  - Szinkronizáció az adatbázissal
  - Entitások keresése lekérdezésekkel

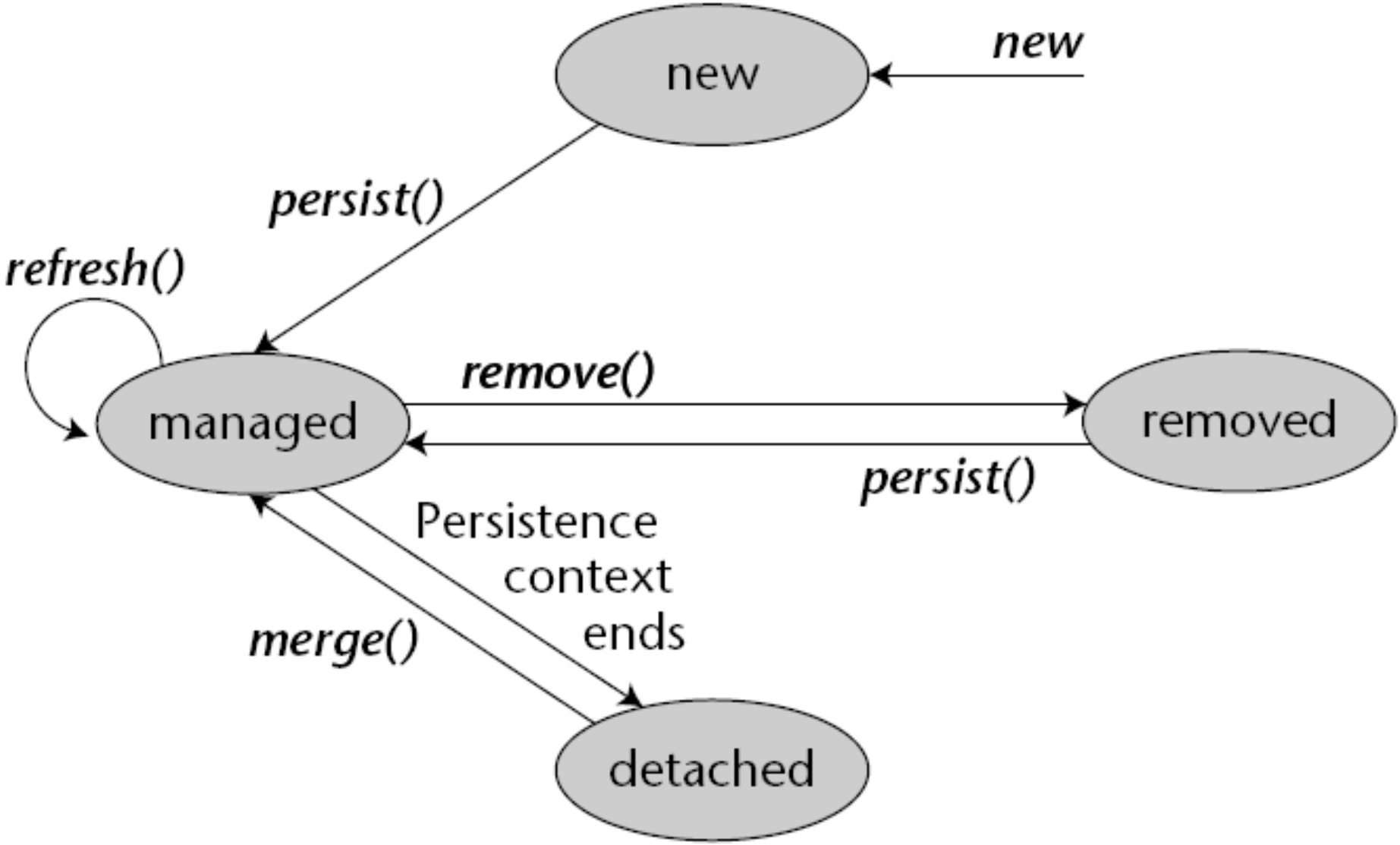
Tranzakciókezelés

- Tulajdonságok:
  - **A**tomic
  - **C**onsistent
  - **I**solated
  - **D**urable
- Hívások:
  - `entityManager.getTransaction().`
    - `begin()`
    - `commit()`
    - `rollback()`

Entitások állapotai

- **new**: new-val létrehozva kerül ide, csak a memóriában létezik, a módosítások nem mennek adatbázisba
- **managed**: létezik az adatbázisban, és hozzátartozik egy perzisztencia kontextushoz. Ez azzal jár, hogy a módosítások tranzakció commit végén, vagy explicit flush() hívással bekerülnek az adatbázisba
- **detached**: adatbázisban megvan, de nem tartozik perzisztens kontextushoz; ebben az állapotban olyan, mint egy DTO (Data Transfer Object)
- **removed**: még perzisztencia kontextushoz tartozik, de már ki van jelölve, hogy törölve lesz az adatbázisbol

Entitások életciklusa



Entitás életciklus callbackek

- Metódusok a következő annotációk valamelyikével:
  - @PrePersist
  - @PostPersist
  - @PreRemove
  - @PostRemove
  - @PreUpdate
  - @PostUpdate
  - @PostLoad
- A persistence provider hívja őket
- Ha külön osztályba akarjuk rakni, @EntityListeners-ben kötjük hozzá az entitáshoz az osztályt, és a metódusok az entitást megkapják paraméterül

Adatbázis szinkronizáció

- Általában commitkor automatikusan megtörténik
- Explicit módon is megtehetjük az EntityManageren keresztül:
  - `flush(entity)`: beírja a változtatásokat
  - `refresh(entity)`: beolvassa a változtatásokat

Lekérdezések

- Egyszerű keresés elsődleges kulcs alapján:

```
<T> T find(Class<T> entityClass, Object primaryKey)
```

- Bonyolultabb lekérdezések:
  - EJB-QL nyelven: `public Query createQuery(String ejbqlString)`
  - natív SQL: `public Query createNativeQuery(String sqlString)`

Lekérdezések

- Paraméterkezelés (biztonság):
  - `setParameter(String, Object)`/ `setParameter(int, Object)`: név vagy index alapján
- Eredmény lekérése:
  - `getSingleResult()`
  - `getResultList()`
- Módosítás/törlés
  - `executeUpdate()`
  - Lehetőség van tömeges törlésre/módosításra

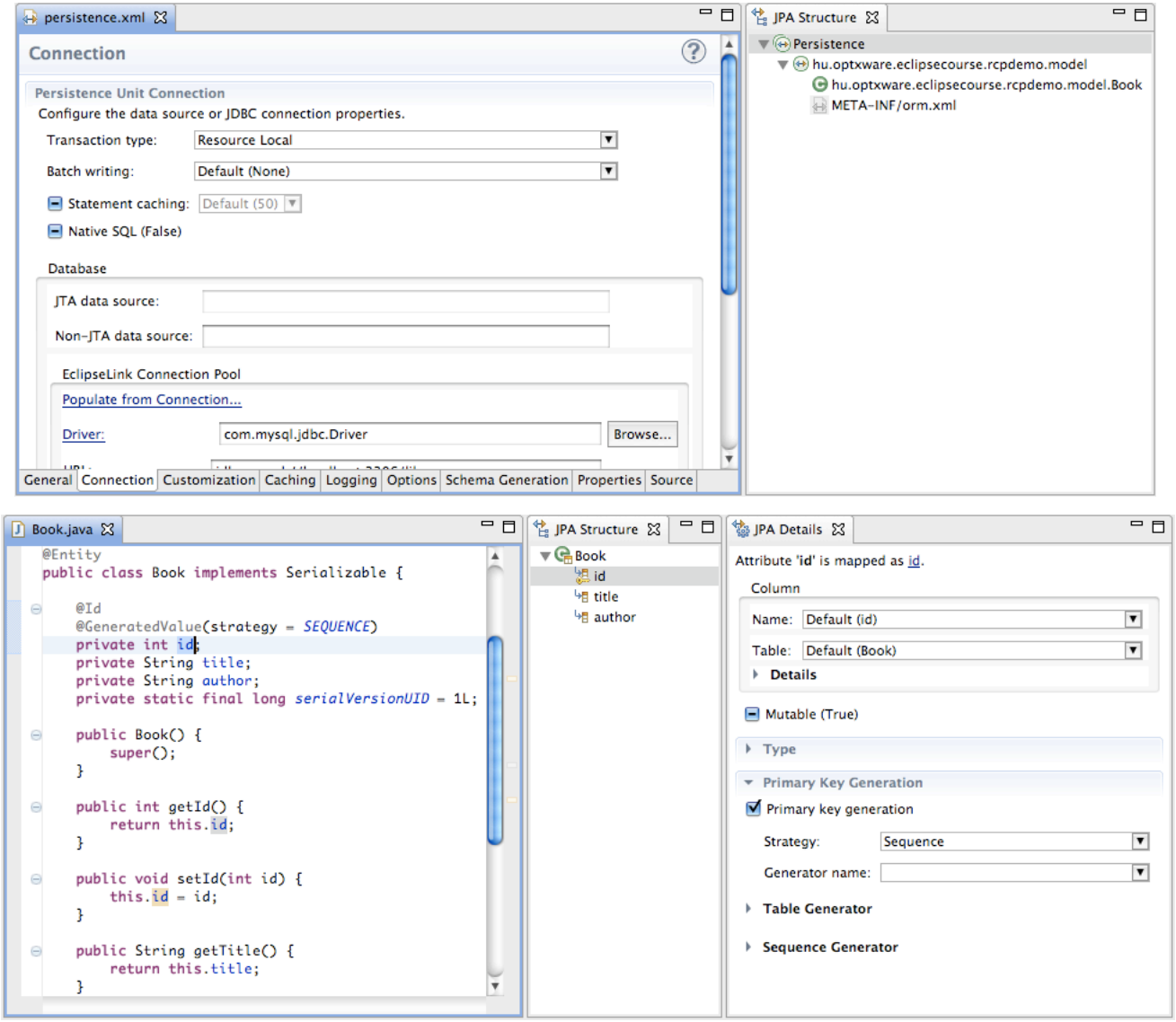
Konkurenciakezelés

- Két lehetőség
  - Optimista
    - @Version-nel meg kell jelölni egy egész szám vagy TimeStamp típusú attribútumot
    - ezt a persistence provider kezeli (update-kor növeli), kódból nem módosítjuk
    - ha beíráskor azt látja, hogy a verziószám módosult (egy konkurens kliens módosítása miatt), nem módosít, hanem OptimisticLockException-t dob
  - Explicit zárak
    - `entityManager.lock(Object entity, LockMode)`
    - LockMode: READ vagy WRITE lehet
    - csak tranzakción belül hívható

Eclipse Dali

- A Web Tools Platform része
  - Természetesen Java SE-vel is használható
- Forms alapú szerkesztőfelület
  - Persistence unit
  - Entity
- Függőségkezelés megkönnyítése
- Integráció bármelyik providerrel, de EclipseLinkkel a legkényelmesebb
- Egyéb eszközök:
  - Entitások és táblák közötti konverzió
  - Annotált & persistence.xml-ben felsorolt entitások szinkronizációja

Eclipse Dali



EclipseLink

- TopLinkre épül
- Jól használható OSGi környezetben
- Együttműködik a Dalival
- `org.eclipse.persistence.jpa` bundle

EclipseLink-specifikus propertyk

- A provider neve

  `org.eclipse.persistence.jpa.PersistenceProvider`
  - OSGI környezetben

    `org.eclipse.persistence.jpa.osgi.PersistenceProvider`
- Automatikus sémalétrehozás
  - `<property name="eclipselink.ddl-generation" value="drop-and-create-tables"/>`
    - vagy `create-tables`
  - `<property name="eclipselink.ddl-generation.output-mode" value="database"/>`

Használat

- EntityManager lekérése

```
EntityManager entityManager =
new org.eclipse.persistence.jpa.osgi.PersistenceProvider().
createEntityManagerFactory(PERSISTENCE_UNIT_NAME).
createEntityManager();
```

- Tranzakció kezdés
  - `entityManager.getTransaction().begin();`
- Tranzakció befejezés
  - `entityManager.getTransaction().commit();`
- EntityManager lezárása
  - `entityManager.close();`

JPA 2.0

JPA 2.0 Features

- Richer mappings
- Richer JPQL
- Pessimistic Locking
- Criteria API
- Cache API
- Many more

JPA 2.0: Richer Mapping

- Supports collection of basic types and embeddables
  - In JPA 1.0, only collections of entities were supported
- Supports multiple levels of embeddables
- Embeddables containing collection of embeddables and basic types
- PK can be derived entities
- More support for Maps...

JPA 2.0: Collection of basic types

```
@Entity
Public class Item {

   @ElementCollection
   private Set<String> tags;
}
```

Mapped by default in ITEM_TAGS

```
@Entity
Public class Item {

   @ElementCollection
   @CollectionTable(name="TAGS")
   private Set<String> tags;
}
```

Mapped in TAGS

JPA 2.0: Richer JPQL

- Added entity type to support non-polymorphic queries
- Allow joins in subquery FROM clause
- Added new operators
  - INDEX (for ordered lists)
  - CASE (for case expressions)
  - more
- Added new reserved words
  - ABS, BOTH, CONCAT, ELSE, END, ESCAPE, LEADING, LENGTH, LOCATE, SET, SIZE, SQRT, SUBSTRING, TRAILING

Example: JPQL CASE Expression

```
@Entity public class Employee {
    @Id Integer empId;
    String name;
    Float salary;
    Integer rating;
    // ...
}

UPDATE Employee e
SET e.salary =
        CASE WHEN e.rating = 1 THEN e.salary * 1.05
             WHEN e.rating = 2 THEN e.salary * 1.02
             ELSE e.salary * 0.95
        END
```

JPA 2.0: Locking Enhancements

- JPA 1.0 supports only optimist locking
- JPA 2.0 adds pessimistic locking
- Multiple places to specify lock
  - read and lock
  - read then lock
  - read then lock and refresh

```
public enum LockModeType {
OPTIMISTIC,
OPTIMISTIC_FORCE_INCREMENT,
PESSIMISTIC,
PESSIMISTIC_FORCE_INCREMENT,
NONE
}
```

JPA 2.0: Criteria API

- Strongly typed criteria API
- Object-based query definition objects
  - rather than string-based
- Like JPQL
- Uses a metamodel – Compile time type checking using Generics
  - Each entity X has a metamodel class X_
  - Criteria API operates on the metamodel

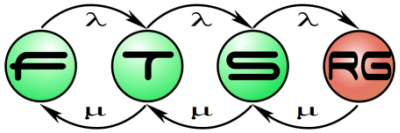
JPA 2.0: Caching

- Supports the use of a second-level cache
- Cache API
  - *contain(Class, PK)*
  - *evict(Class, PK), evict(Class)*
  - *evictAll()*
- *@Cacheable* annotation on entities

References

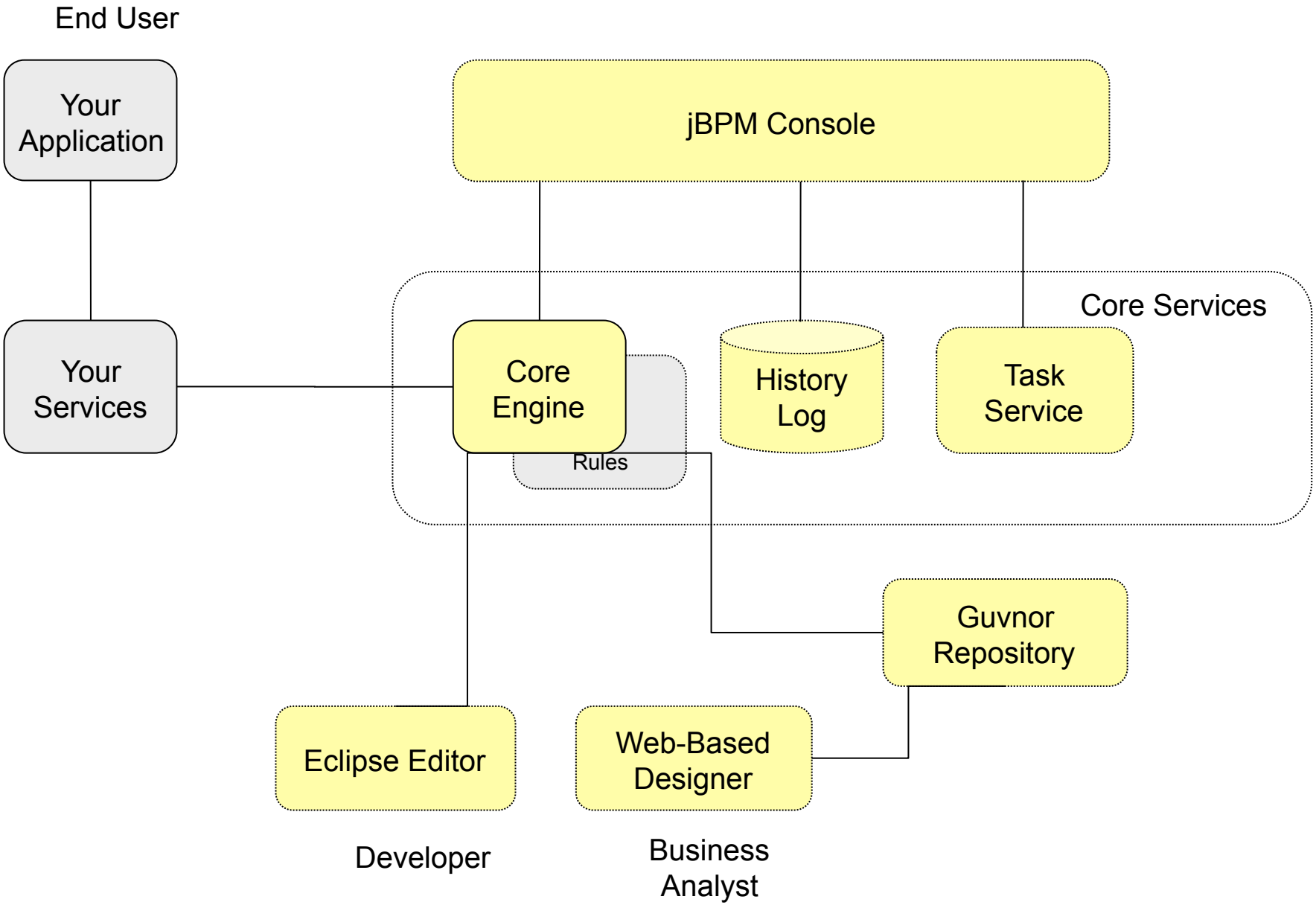
- Mike Calvo: JPA and Hibernate
  - http://www.slideshare.net/adorepump/jpa-and-hibernate-presentation
- Gordon Yorke: EclipseLink JPA
  - http://www.slideshare.net/pelegri/eclipselink-jpa-presentation
- Markus Eisele: New features of JSR-317
  - http://www.slideshare.net/myfear/new-features-of-jsr-317-jpa-20
- Varró Gergely: Objektum-relációs leképezés
  - UML bázisú modellezés és analízis c. tárgy anyagai (2006-2010)
- Ráth István, Ujhelyi Zoltán és mások, Eclipse alapú tervezés és integráció c. tárgy anyagai, (2010-2012)

Overview of jBPM5



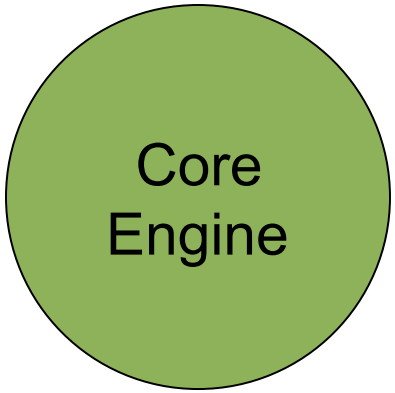
Key Characteristics of jBPM5

- Open-source business process management project offering:
  - generic process engine supporting native BPMN 2.0 execution
  - targeting developers and business users
  - collaboration, management and monitoring using web-based consoles
  - powerful rules and event integration

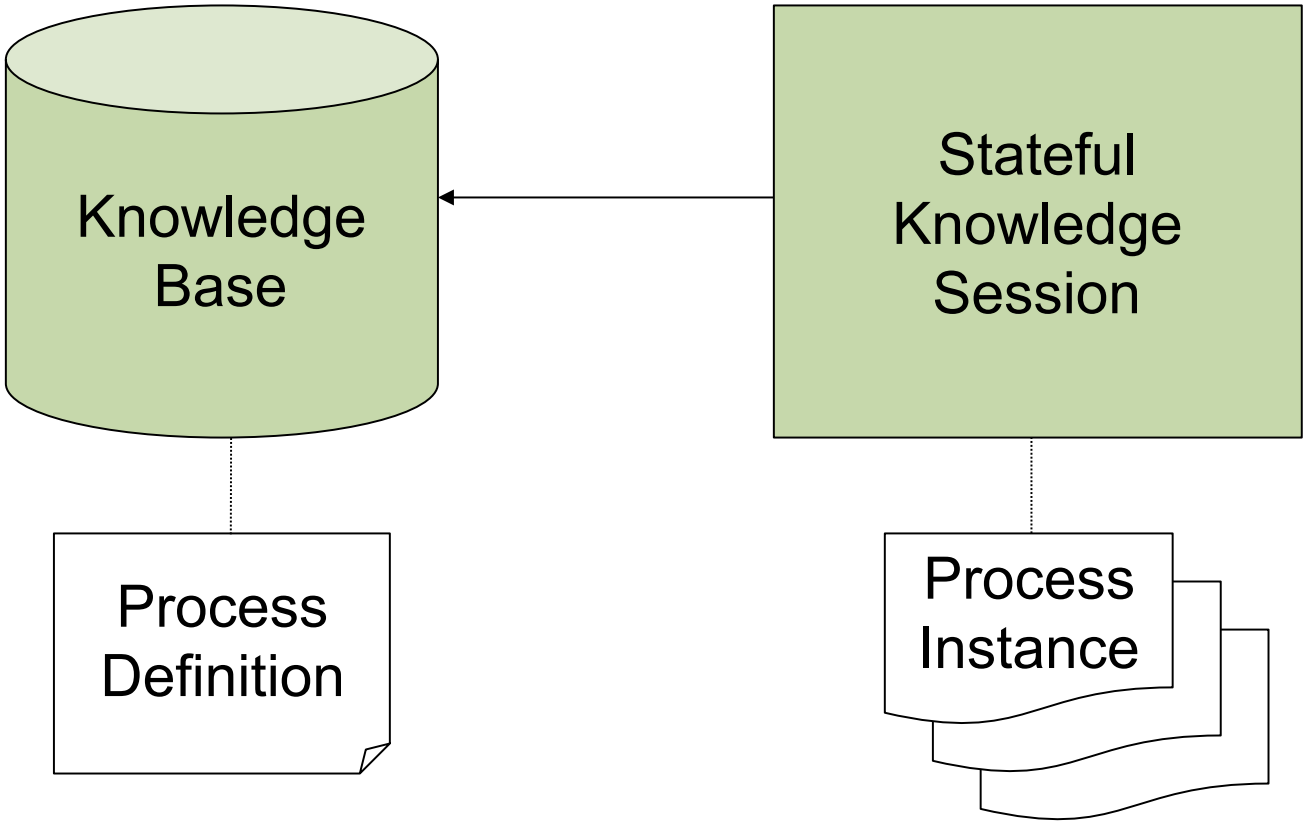

End User
Your Application
jBPM Console
Core Services
Your Services
Core Engine
History Log
Task Service
Rules
Guvnor Repository
Eclipse Editor
Web-Based Designer
Developer
Business Analyst

From Workflow to BPM

- Core engine is a workflow engine in pure Java
  - state transitions
  - lightweight
  - embeddable
  - generic, extensible



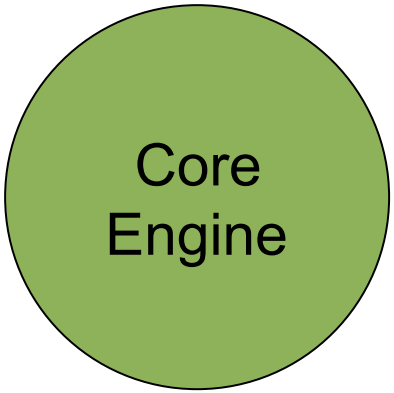
Core Engine


Knowledge Base
Stateful Knowledge Session
Process Definition
Process Instance

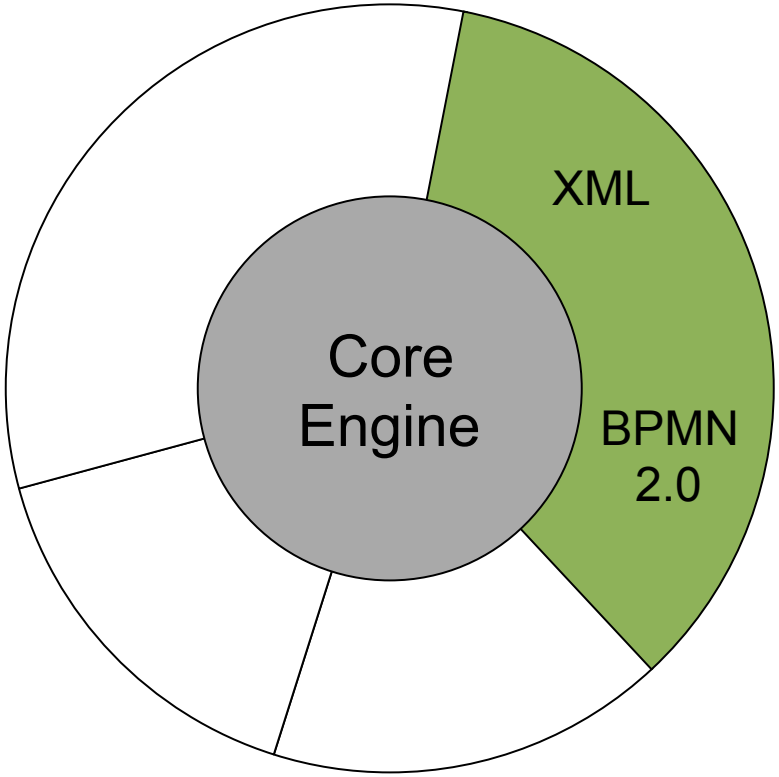
Java Example

```
// (1) Create knowledge base and add process definition
KnowledgeBuilder kbuilder = ...
kbuilder.add( ..., "sample.bpmn", ResourceType.BPMN2);
KnowledgeBase kbase = kbuilder.newKnowledgeBase();
// (2) Create new stateful knowledge session
StatefulKnowledgeSession ksession =
    kbase.newStatefulKnowledgeSession();
// (3) Start a new process instance
ksession.startProcess("com.sample.bpmn.hello");
```

From Workflow to BPM

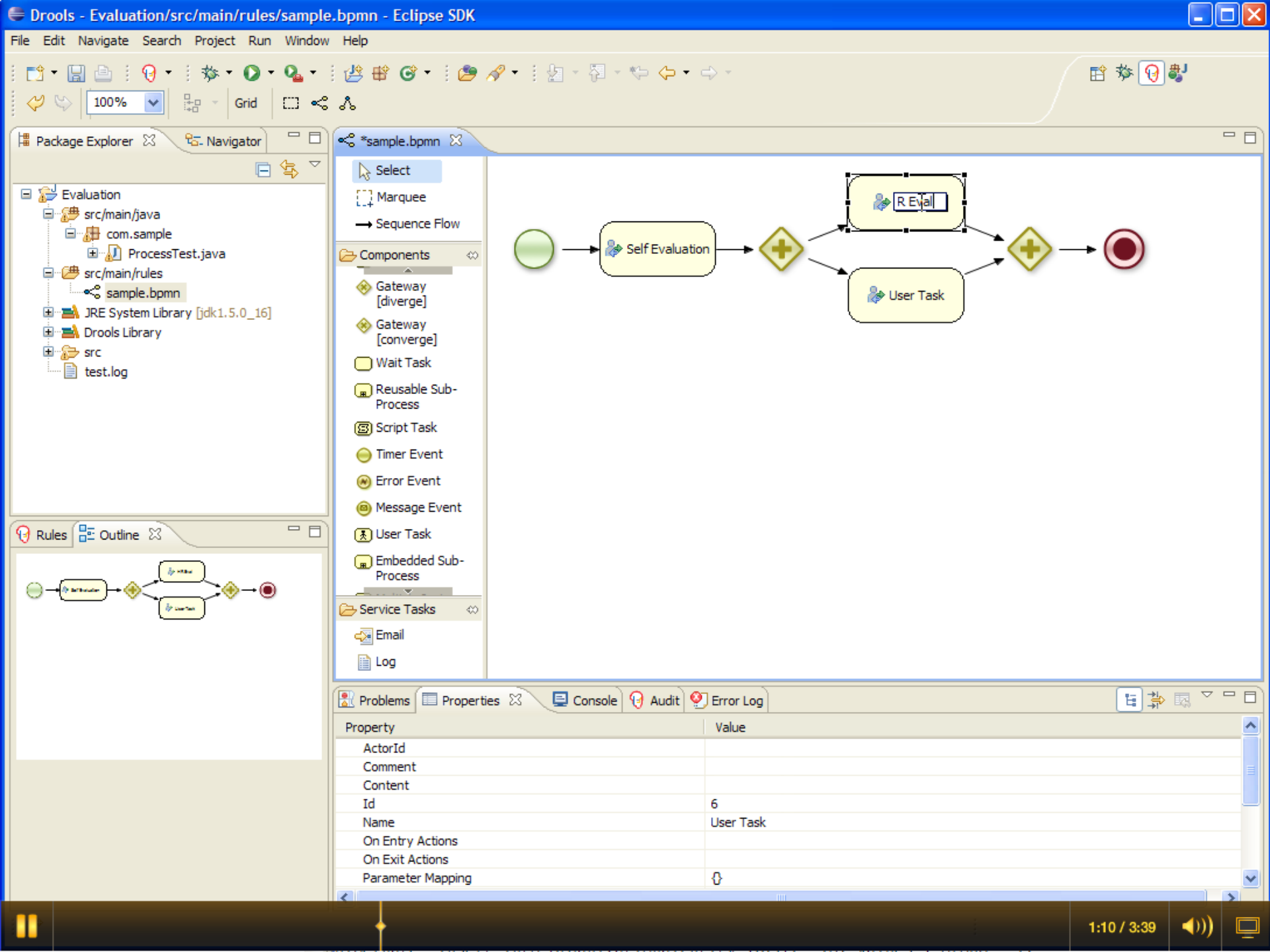


From Workflow to BPM

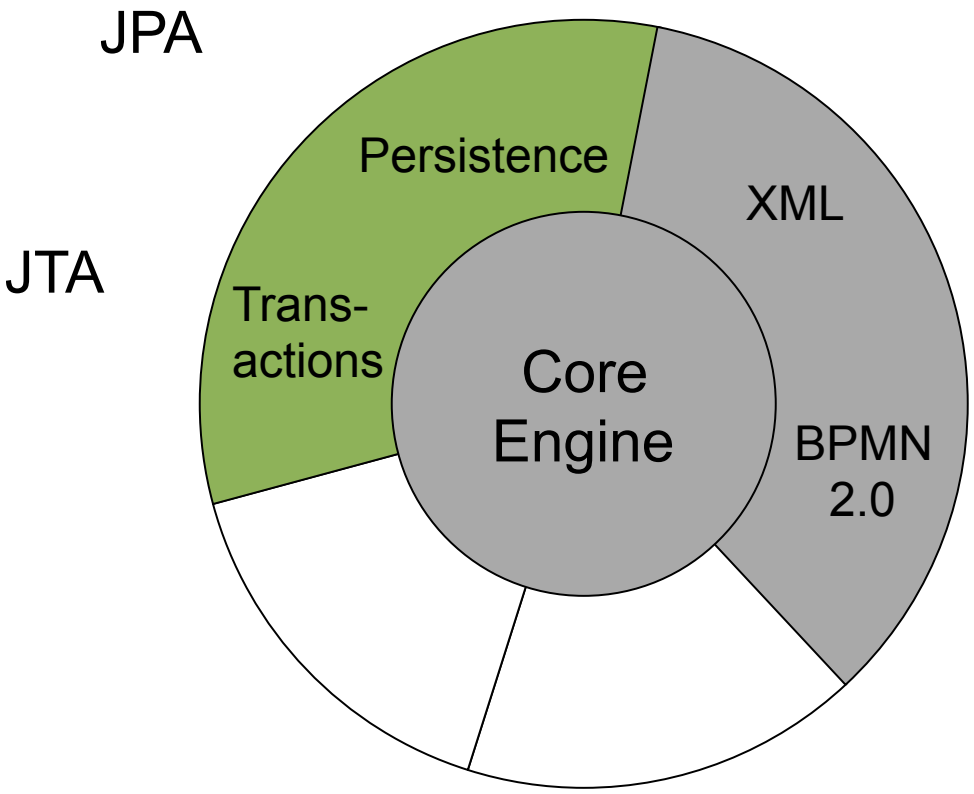


BPMN 2.0 Example

```
<definitions ... >
   <process id="com.sample.bpmn.hello" name="Hello World" >
      <startEvent id="_1" name="StartProcess" />
      <sequenceFlow sourceRef="_1" targetRef="_2" />
      <scriptTask id="_2" name="Hello" >
        <script>System.out.println("Hello World");</script>
      </scriptTask>
      <sequenceFlow sourceRef="_2" targetRef="_3" />
      <endEvent id="_3" name="EndProcess" />
    </process>
 </definitions>
```


Drools - Evaluation/src/main/rules/sample.bpmn - Eclipse SDK
File Edit Navigate Search Project Run Window Help
100%
Grid
Package Explorer
Navigator
Evaluation
src/main/java
com.sample
ProcessTest.java
src/main/rules
sample.bpmn
JRE System Library [jdk1.5.0_16]
Drools Library
src
test.log
Rules
Outline
*sample.bpmn
Select
Marquee
Sequence Flow
Components
Gateway [diverge]
Gateway [converge]
Wait Task
Reusable Sub-Process
Script Task
Timer Event
Error Event
Message Event
User Task
Embedded Sub-Process
Service Tasks
Email
Log
R Eval
Self Evaluation
User Task
Problems
Properties
Console
Audit
Error Log
Property
Value
ActorId
Comment
Content
Id
6
Name
User Task
On Entry Actions
On Exit Actions
Parameter Mapping
{}
1:10 / 3:39

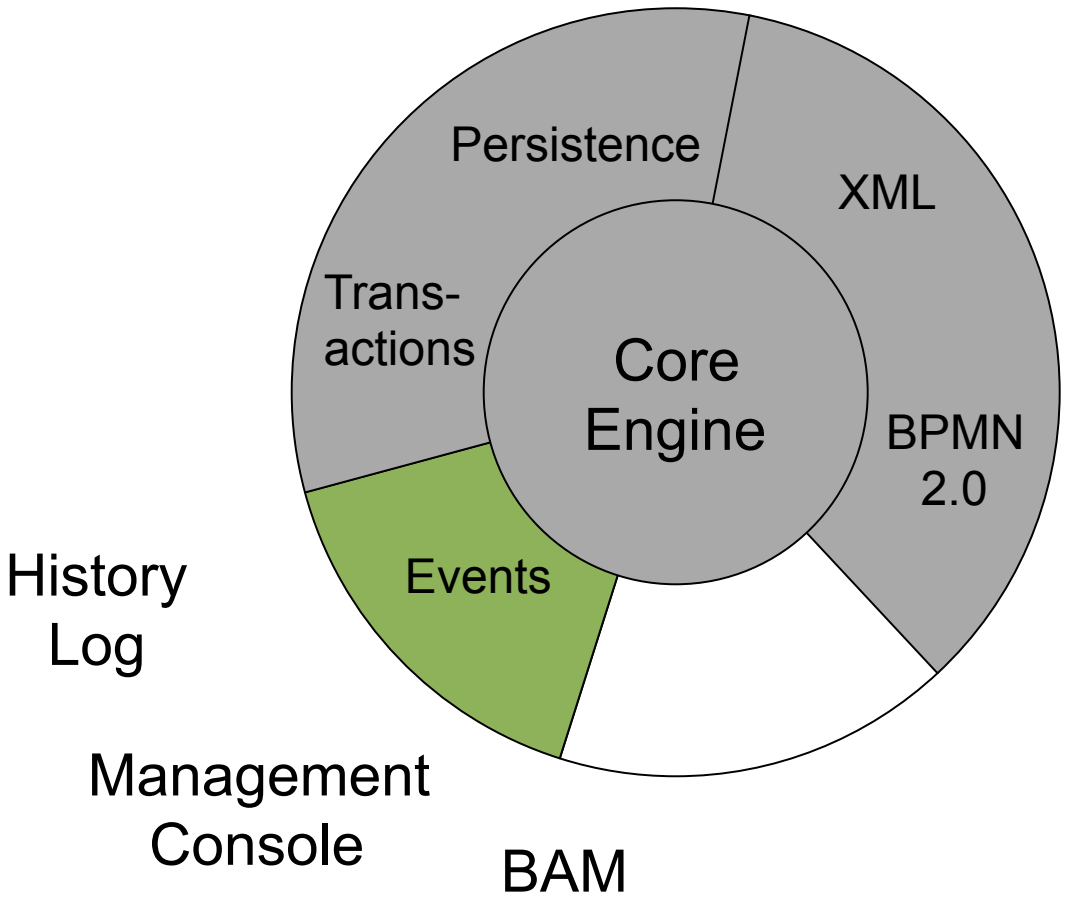
From Workflow to BPM



Persistence and Transactions

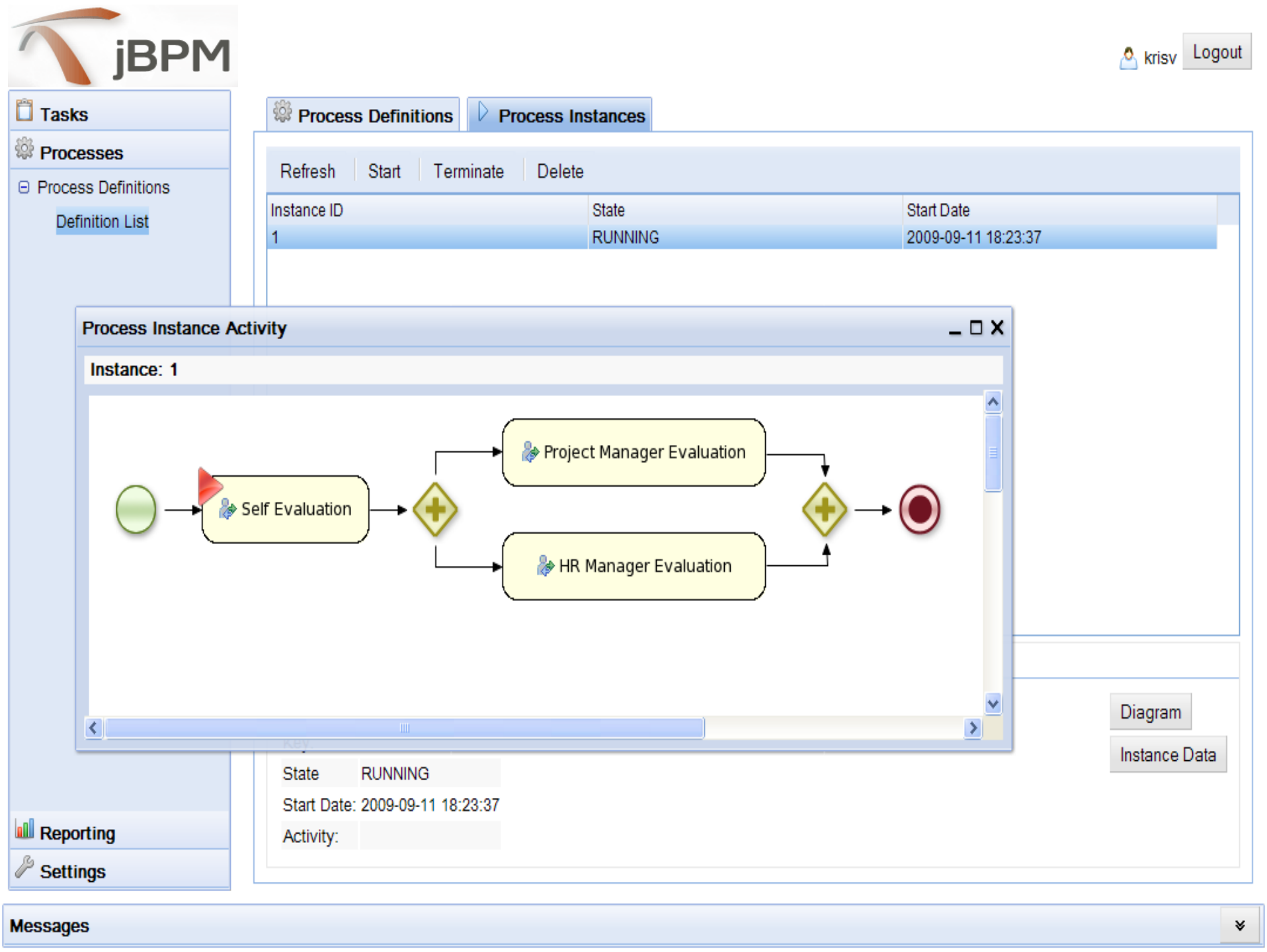
- Persistence (JPA, pluggable)
  - Runtime persistence
  - History logging
  - Services
- Transactions (JTA, pluggable)
  - Command-scoped
  - User-defined

From Workflow to BPM

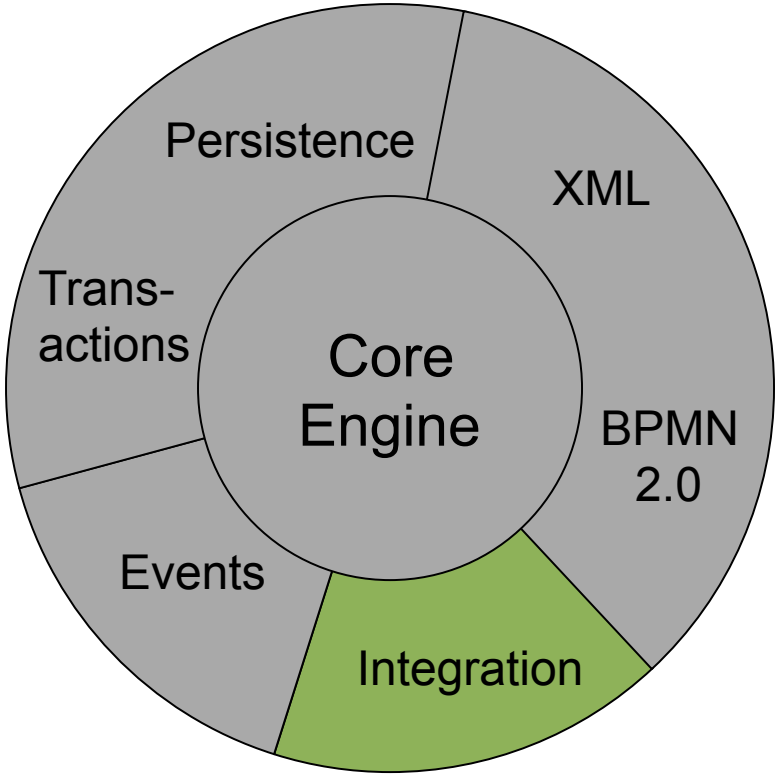


Console

- Web-based management
- Business user
- Features
  - Process instance management
  - User task lists / forms
  - Reporting


jBPM
krisv
Logout
Tasks
Processes
Process Definitions
Definition List
Reporting
Settings
Process Definitions
Process Instances
Refresh
Start
Terminate
Delete
Instance ID
State
Start Date
1
RUNNING
2009-09-11 18:23:37
Process Instance Activity
Instance: 1
Self Evaluation
Project Manager Evaluation
HR Manager Evaluation
Diagram
Instance Data
State
RUNNING
Start Date:
2009-09-11 18:23:37
Activity:
Messages

From Workflow to BPM



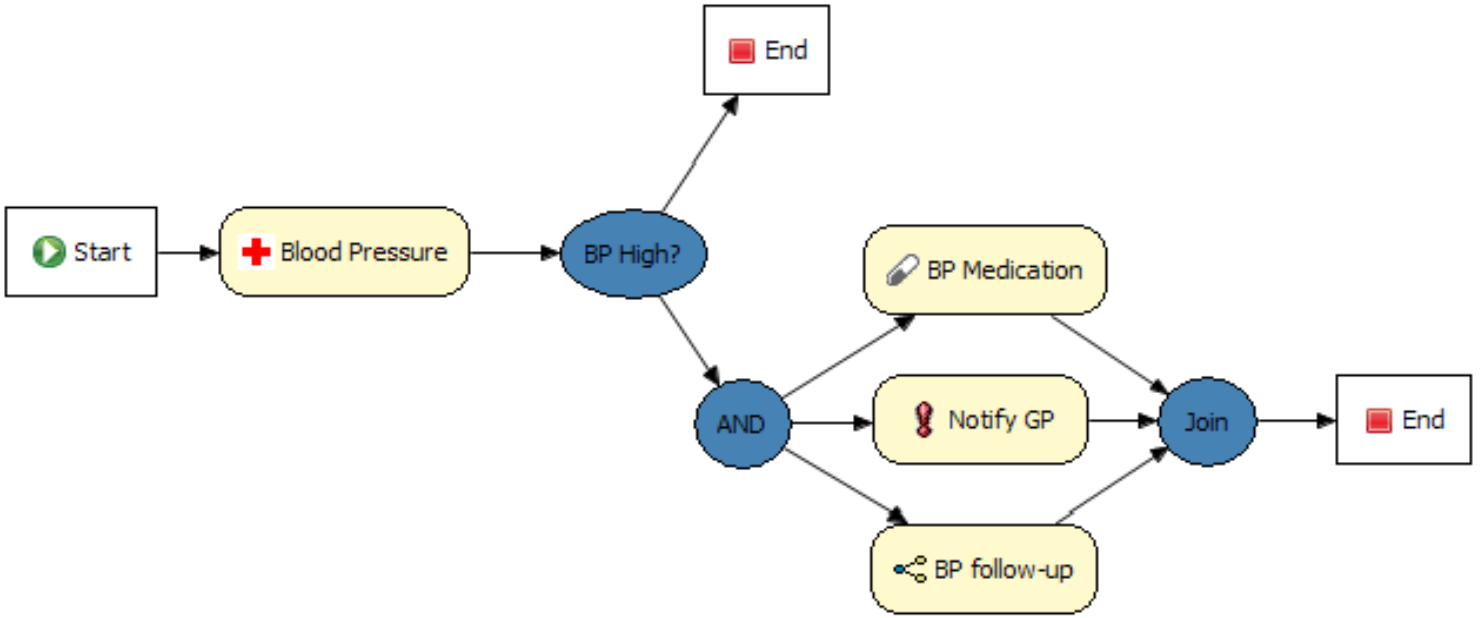
Domain-specific
Processes

ESB

Human Task
Service

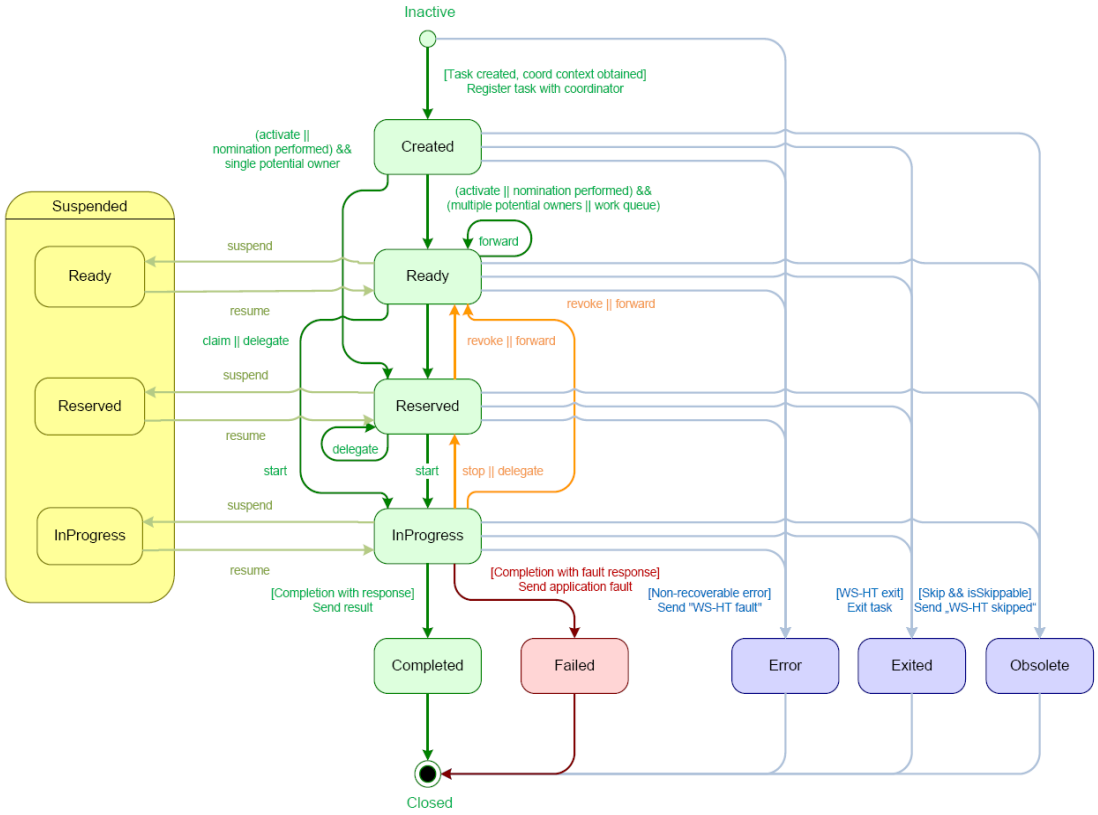
Domain-specific Processes

- Extend palette with domain-specific, declarative service nodes
  - define input / output parameters
  - runtime binding

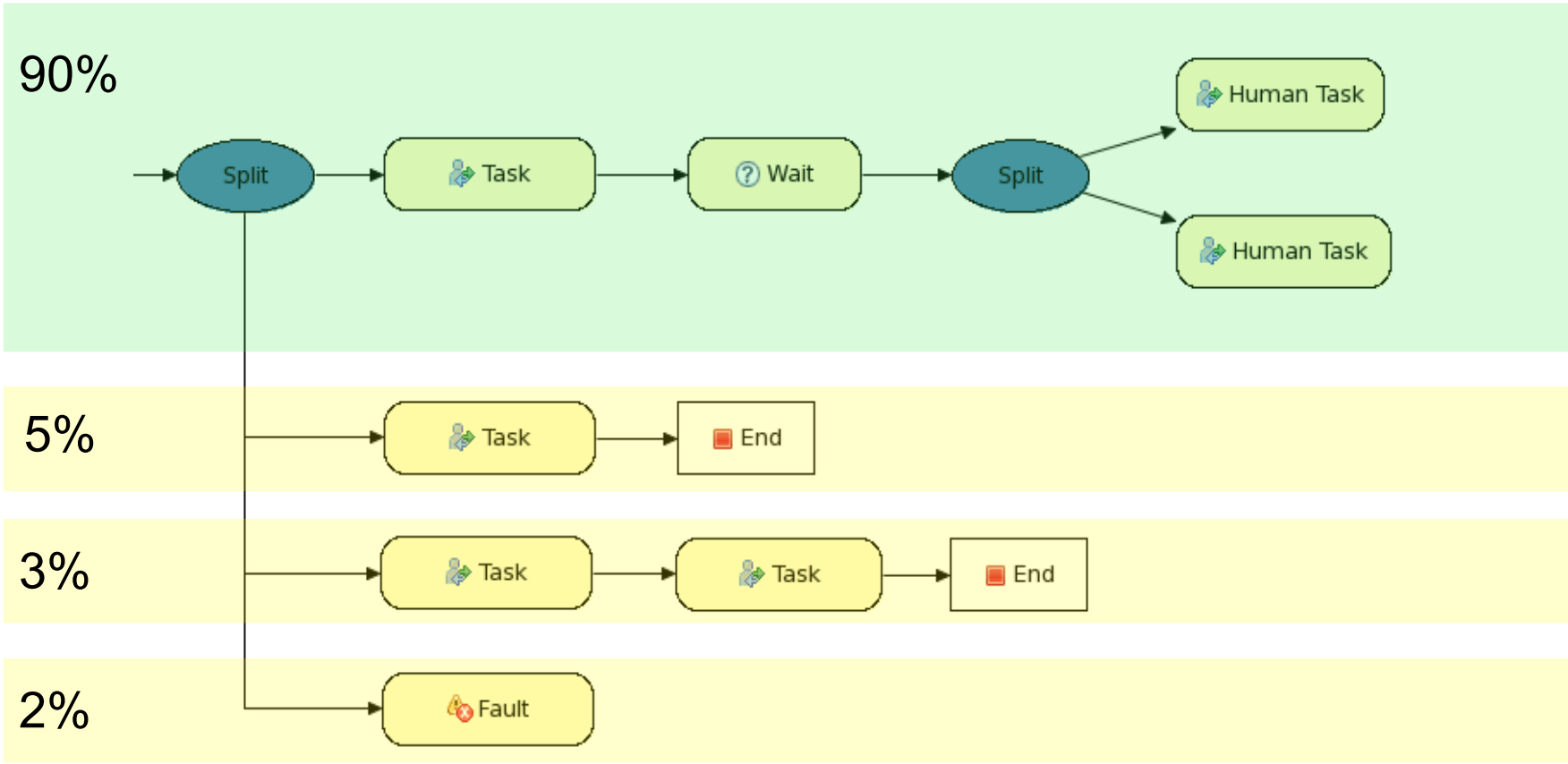


Human task service

- User task
- Human task service (WS-HT)
  - Task lists
  - Task life cycle
- Task clients
  - Task forms

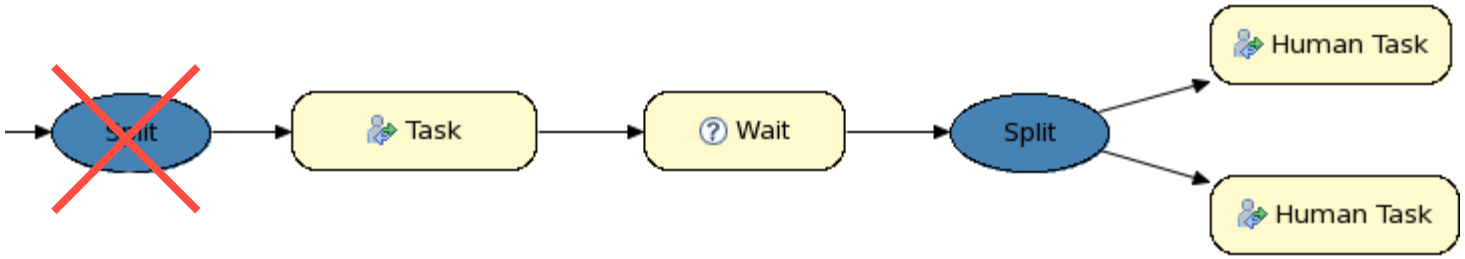


Exceptional Control Flow



Exceptional Control Flow

90%



Rule1
When
...
Then
...

5%

Rule2
When
...
Then
...

3%

Rule3
When
...
Then
...

2%

Roadmap

- jBPM 5.0: February 2011
- jBPM 5.1: NOW …
  - Improve designer to support full round-tripping
  - New Eclipse BPMN2 editor
  - Lots of small feature improvements and bug fixes
- jBPM 5.2 – jBPM 6.x
  - Simulation / testing / replay
  - Flexible, adaptive, goal-oriented processes
  - Integrated “no-code” tooling (form builder, etc.)

jBPM5: What, where?

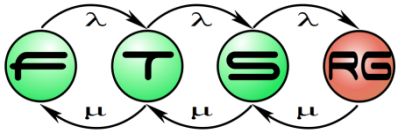
- Source http://github.com/droolsjbpm/jbpm
- Hudson http://hudson.jboss.org/hudson/job/jBPM
- Blog http://kverlaen.blogspot.com/
- #jbpm on irc.codehaus.org
- jbpm-dev@jboss.org

References

- Kris Verlaenen: BPMN2 and JBPM5
  - http://www.slideshare.net/krisverlaenen/bpmn2-and-jbpm5

JSON and REST

The New Kids on the Data Block



REST

What does it stand for?:

Representational **S**tat**e** **T**ransfer

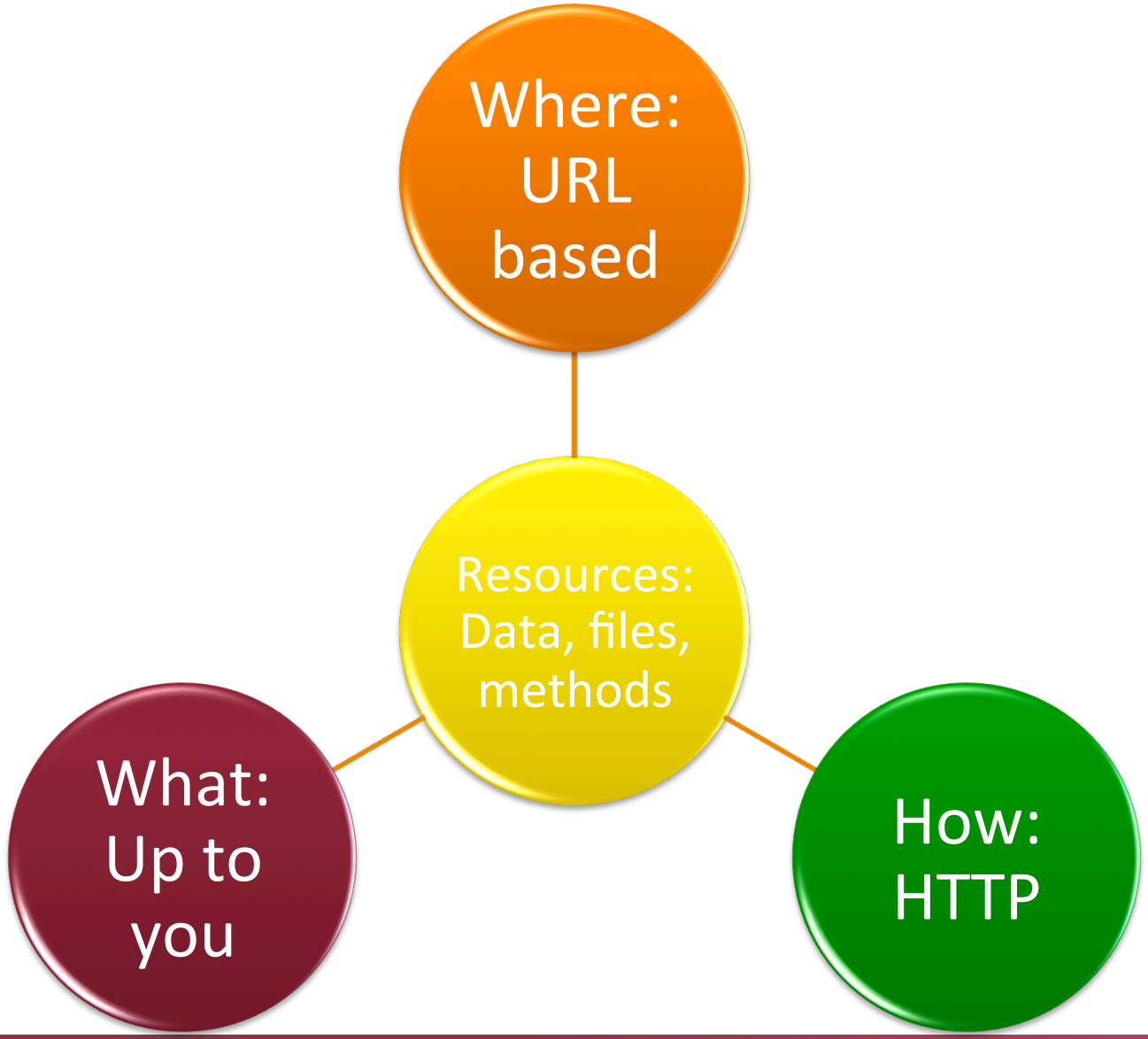
What Is it?

A **style** of software architecture for distributed systems

Who/Where/When?

Came about in 2000 doctoral dissertation of Roy Fielding – but it's been used for much longer

REST – Core Principles



REST – Where/How: Simple Example

Premise:

Data in a table could be a resource we want to read

Database server called *bbddb01*

Database called *northwind*

Table called *users*

- **http://bbddb01/northwind/users**

What, What, What?

- What type of content you return is up to you.
- Compare to SOAP where you must return XML.
- Most common are XML or JSON. You could return complex types like a picture.

REST – Is it used?

- Web sites are RESTful
- RSS is RESTful
- Twitter, Flickr and Amazon expose data using REST

- Some things are “accidentally RESTful” in that they offer limited support.

Real Life: Flickr API

- Resource: Photos
- Where:
  - http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}.jpg
  - http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}_[mstb].jpg
  - http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{o-secret}_o.(jpg|gif|png)
- What: JPEG, GIF or PNG (defined in the URL)
- **http://farm1.static.flickr.com/2/1418878_1e92283336_m.jpg**

REST – Methods

HTTP Methods are a key corner stone in REST.
They define the action to be taken with a URL.
Proper RESTful services expose all four – “accidental” expose less.
Nothing stopping you doing some Mix & Match
❖Some URL’s offering all of them and others a limited set

What are the four methods and what should they do?

REST	CRUD (Create, Read, Update, Delete)
POST	Create
GET	Read
PUT	Update or Create
DELETE	Delete

REST – Methods Example

http://bbddb01/northwind/users[firstname="rob%"]

- **+ POST = Error**
- **+ GET = Returns everyone who begins with rob**
- **+ PUT = Error**
- **+ DELETE = Deletes everyone who begins with rob**

http://bbddb01/northwind/users

& we add some input data

- **+ POST = Creates a new user**
- **+ GET = Returns everyone who meets criteria**
- **+ PUT = Creates/Updates a user (based on data)**
- **+ DELETE = Deletes everyone who meets criteria**

REST – Methods Example

http://bbddb01/northwind/users[firstname="rob%"]

+ POST = Error

+ PUT = Error

What would the error be?

HTTP 400 or 500 errors are normally used to indicate problems – same as websites

REST – Commands

You can associate commands with a resource.

Commands can replace the need for using HTTP methods and can provide a more familiar way of dealing with data.

Example:

```
userResource = new Resource('http://example.com/users/001')
userResource.delete()
```

Comparison: REST vs. SOAP

Comparing apples and oranges

REST vs. SOAP – pt I: Technology

REST	SOAP
A STYLE	A Standard
Proper REST: Transport must be HTTP/HTTPS	Normally transport is HTTP/HTTPS but can be something else
Response data is normally transmitted as XML, can be something else. ❖On average the lighter of the two as does not have SOAP header overhead	Response data is transmitted as XML
Request is transmitted as URI ❖Exceptionally light compared to web services ❖Limit on how long it can be ❖Can use input fields	Request is transmitted as XML
Analysis of method and URI indicate intent	Must analyse message payload to understand intent
…	WS* initiatives to improve problems like compression or security

REST vs. SOAP – pt II: Languages

REST	SOAP
Easy to be called from JavaScript	JavaScript can call SOAP but it is hard, and not very elegant.
If JSON is returned it is very powerful (keep this in mind)	JavaScript parsing XML is slow and the methods differ from browser to browser.
C# (Visual Studio) parsing of REST means using HttpWebRequest and parsing the results (string/xml) or normal service consumption (.NET 3.5 SP 1 and later).	C# (Visual Studio) makes consuming SOAP very easy and provides nice object models to work with.
C# version 4 should make this easier thanks to new dynamic methods.	...
There are 3rd party add-on's for parsing JSON with C# so that may make it easier.	...

REST vs. SOAP – pt III: Tools

REST	SOAP
Basic support for REST in BizTalk	BizTalk and SOAP are made to be together.
WCF can consume REST.	WCF can consume SOAP.
WCF can serve REST with a bit of tweaking.	WCF can server SOAP.
The new routing feature in ASP.NET 3.5 SP1 makes building a RESTful service easy.	...

FAQ about Security?

- Are RESTful services secure?
  - It's a style, not a technology so that depends on how you implement it.
- Are you open to SQL injection attacks?
  - When you look at *http://bbddb01/northwind/users[firstname="rob%"]*, you may think so but you shouldn't be. Because:

1) The parameter shouldn't be SQL
2) If it is SQL, why are you not filtering it?
3) Remember the old rule: Do not trust user input

FAQ about Security?

- How can I do authentication?
  - It's built on HTTP, so everything you have for authentication in HTTP is available
  - PLUS
  - You could encode your authentication requirements into the input fields

JSON

JSON – What is it?

- *"JSON (JavaScript Object Notation) is a **lightweight data-interchange format**. It is easy for humans to read and write. It is easy for machines to parse and generate"* – JSON.org

- Importantly: JSON is a subset of JavaScript

JSON – What does it look like?

```
{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 555-1234",
        "646 555-4567"
    ]
}
```

Name/Value Pairs

Child properties

String Array

Number data type

JSON vs. XML

JSON	XML
Data Structure	Data Structure
No validation system	XSD
No namespaces	Has namespaces (can use multiples)
Parsing is just an eval •Fast •Security issues	Parsing requires XML document parsing using things like XPath
In JavaScript you can work with objects – runtime evaluation of types	In JavaScript you can work with strings – may require additional parsing
Security: Eval() means that if the source is not trusted anything could be put into it. Libraries exist to make parsing safe(r)	Security: XML is text/parsing – not code execution.

JSON vs. XML which to use?

- Scenario 1: You have a website (say Twitter.com) and you want to expose a public API to build apps.

Issue	JSON	XML
The public will be parsing data in. You must make it secure.	Run checks against the data in the object to make sure it's secure. You are working on objects so you must also check for potential code access issues.	Run checks against the data to make sure it's secure.
Data must be in a specific format.	Build something that parses the objects.	XML Schema

JSON vs. XML which to use?

- Scenario 2: You have a website (say gmail.com) and your front end needs to show entries from a mailbox, but needs to be dynamic and so you will use a lot of JavaScript.

Issue	JSON	XML
Your in house developers know objects and would like to use them.	JSON is JavaScript objects.	Write JavaScript to parse the XML to objects.
The site is secure but you worry about people checking the page source.	You page has JavaScript in it and (maybe) code which communicates with a private backend server. No major issues.	You page has JavaScript in it and (maybe) code which communicates with a private backend server. No major issues.

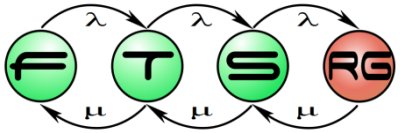
JSON vs. XML

- **Which of them should you use?**
  - Use Both – They both have strengths and weaknesses and you need to identify when one is stronger than the other.

References

- Robert MacLean: JSON and REST
  - http://www.slideshare.net/rmaclean/json-and-rest
- Brian Mulloy: RESTful API design
  - http://www.slideshare.net/apigee/restful-api-design-second-edition
- Christopher Bartling et al: RESTful web services
  - http://www.slideshare.net/cebartling/rest-web-services

JavaScript Object Notation (JSON)



JSON

- **JSON** (JavaScript Object Notation) is a lightweight data-interchange format.
- It is easy for humans to read and write.
- It is easy for machines to parse and generate.
- It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999.
- JSON is a text format that is completely language independent.
- These properties make JSON an ideal data-interchange language.

Why JSON?

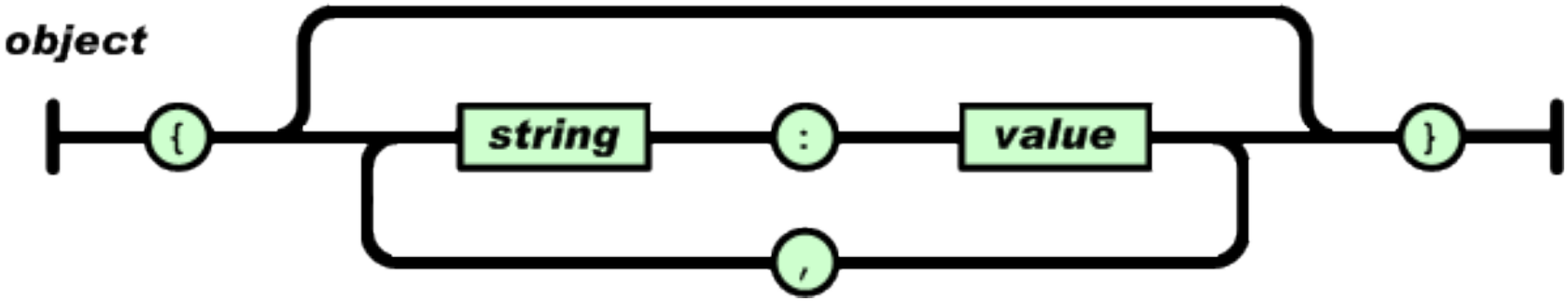
- Because JSON is lightweight, easy to understand, manipulate and generate, it has almost replaced XML which was used previously as the only data-interchange format.
- JSON is preferable because of the following reasons:
  - XML is heavier than JSON
  - to parse XML, we have to use xPath which is an overhead removed in JSON because JSON is native to JavaScript
  - XML uses tags to describe user data and tags increase the size of data

JSON Structures

- JSON is built on two structures:
  - A collection of name/value pairs.
    - In various languages, this is realized as an *object*, record, dictionary, hash table, keyed list, or associative array.
  - An ordered list of values.
    - In most languages, this is realized as an *array*, vector, list, or sequence.

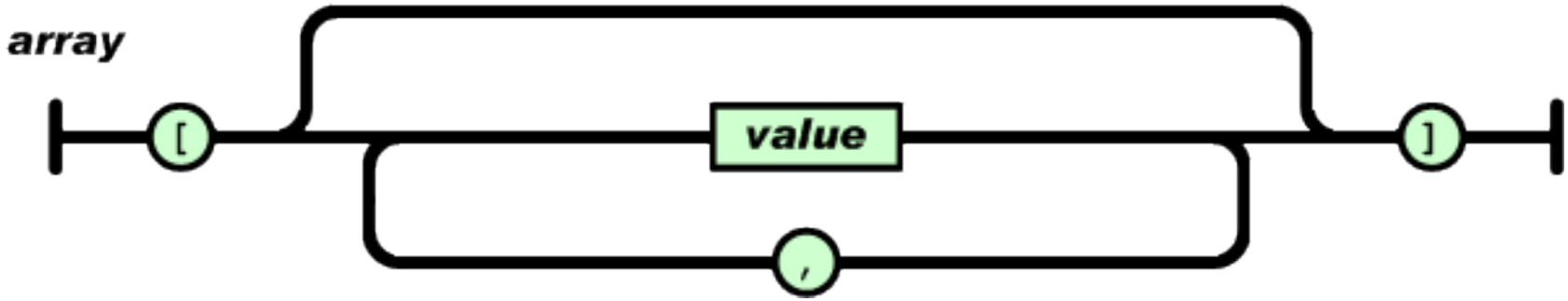
Syntax of JSON

- **Object**
  - An *object* is an unordered set of name/value pairs.
  - An object begins with { (left brace) and ends with } (right brace).
  - Each name is followed by : (colon) and the name/value pairs are separated by , (comma).



Syntax of JSON

- **Array**
  - An *array* is an ordered collection of values.
  - An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).



Syntax of JSON

- A *value* can be a *string* in double quotes, or a *number*, or true or false or null, or an *object* or an *array*. These structures can be nested.
- A *string* is a collection of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.

JSON Example

```
{ “students” :
   [
      {“id":1, "name":"Adnan Sohail"},
      {“id":2, "name":"Irfan Razzaq"}
   ]
}
```

XML Example

```
<?xml version="1.0" ?>
<root>
    <student>
            <id>1</id>
            <name>Adnan Sohail</name>
    </student>
    <student>
            <id>2</id>
            <name>Irfan Razzaq</name>
    </student>
</root>
```

Validating JSON & JSON Security

- JavaScript's built-in method eval() is used to validate a JSON string.
- **Note:**
  - Use eval() only when the source is authentic and trusted which means use it only if you are sure that the string passed to it is a valid JSON string
- When you've security risks use

```
var myObject = myJSONtext.parseJSON();
```

- Which is available in http://www.json.org/json.js
  - but eval() is faster than parseJSON()

Using JSON APIs

- JSON strings can be easily generated using JSON APIs available at http://json.org
- There are two main classes available in `org.json.*` package
  - `org.json.JSONObject`
  - `org.json.JSONArray`
- Strings can be generating from objects of `JSONObject` or `JSONArray` using their `toString()` methods

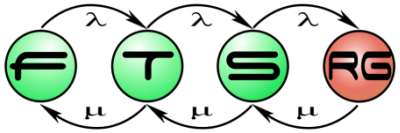
Benefits of JSON over XML

- JSON supports data types like string, integer, boolean etc.
- JSON is native data format for JavaScript and therefore it faster for the browser to read and understand.
- As JSON contains no tags but data and therefore less data to be transferred between client and the server. So, it's lighter than XML.
- Easy for humans to read and write.

JSON References

- http://json.org/
- http://www.json.org/java/
- S. Gupta: JSON
  - http://www.slideshare.net/sanjay/j-s-o-n

RESTful Web services using Java



About Web services

- What is a Web service?
- What is a WSDL?
- What is SOAP?

WSDL Structure

```
<definitions>

    <types>
      definition of types.....
    </types>

    <message>
      definition of a message....
    </message>

    <portType>
      definition of a port.....
    </portType>

    <binding>
      definition of a binding....
    </binding>

    </definitions>
```

SOAP Sample

```
POST /StockQuote HTTP/1.1
SOAPAction: "Some-URI“

<soapenv:Envelope xmlns:soapenv="......">
  <soapenv:Header/>
  <soapenv:Body>
    <it:GetEmployeeName>
      <it:ID>4118830901957010</it:ID>
    </it:GetEmployeeName>
  </soapenv:Body>
</soapenv:Envelope>
```

Have you ever wondered why here is so much overhead in designing a simple web service like ‘GetEmployeeName’ using SOAP?

REpresentational **S**tate **T**ransfer (REST)

- **RE**presentational **S**tate **T**ransfer
- What is JAX-RS?
- What is Jersey?

REST Concepts

- **Resources (nouns)**
  - Identified by a URI
- **Uniform interface (verbs)**
  - Small fixed set:

  Create, Read, Update, Delete

- **State Representations**
  - data and state transferred between client and server

  XML, JSON, Atom, XHTML, ...

JAX-RS Introduction

- Set of Java APIs for development of
  web services built according to the REST principals
- Annotations
- Jersey is the open source , production quality, JAX-RS (JSR 311) Reference Implementation for building RESTful Web services.
- Jersey Homepage
  https://jersey.dev.java.net/

Use Standard Methods

Method	Purpose
GET	Read, possibly cached
POST	Update or create without a known ID
PUT	Update or create with a known ID
DELETE	Remove

Sample RESTful Web service

```
@Path("/helloworld/{name}")
public class HelloworldResource {
    @Context
    private UriInfo context;

    /** Creates a new instance of HelloworldResource */
    public HelloworldResource() {
    }

    /**
     * Retrieves representation of an instance of jaxrstest.HelloworldResource
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("text/plain")
    public String sayHello(@PathParam("name") String name,
                           @DefaultValue("HR") @QueryParam("dep") String department) {
        return "Hi "+name+" Welcome to "+ department +" department";
    }

    @GET
    @Path("/sunresource")
    public String testSubResource(){
        return "This is from Sub Resource";
    }

    /**
     * PUT method for updating or creating an instance of HelloworldResource
     * @param content representation for the resource
     * @return an HTTP response with content of the updated or created resource.
     */
    @PUT
    @Consumes("text/plain")
    public void putText(String content) {
        System.out.println("======"+content+"=====");
    }

}
```

HTTP Example

Examples on the web

- **Google AJAX Search API**
  – http://code.google.com/apis/ajaxsearch/
- **Amazon S3**
  – http://aws.amazon.com/s3
- **Services exposing Atom Publishing Protocol or GData**
  – i.e. Google apps like Google Calendar
- **Accidentally RESTful**
  – Flickr, Del.icio.us API

REST Framework alternatives

- **Jersey (opensource reference implementation)**
  - http://jersey.java.net/
- **Restlet (opensource client and server API)**
  - http://www.restlet.org/
- **CXF**
  - HttpBinding
  - JAX-WS Provider/Dispatch API
- **Axis2**
  - HttpBinding (WSDL 2.0)

For More Information

- **BOF-5613 - Jersey: RESTful Web Services Made Easy**
- **Official JSR Page**
  > http://jcp.org/en/jsr/detail?id=311
- **JSR Project**
  > http://jsr311.dev.java.net/

http://www.cwinters.com/rest/

RESTful WS OSGi alatt

- Java annotációk használata
- OSGi Service osztályra:
  - @Path: a szolgáltatás elérési útja
- OSGi Service egy metódusra:
  - @Path: adott metódus elérési útja
  - @GET, @POST, @PUT, stb
  - @PathParam: a paraméter neve az elérési útban
- Component Definition az OSGi szolgáltatáshoz
- JAX-RS Connector ajánlja ki REST szolgáltatásként

Webszolgáltatások .NET platformon

- ASP.NET: alapvetően SOAP-ra felkészítve
- Codebehind pattern: nem MVC-szerű struktúra
  - Nem értelmezhetőek a REST-es URI template-ek
  - Ezen az ASP.NET MVC már segít (kb. 2010 óta)
- Windows Communication Foundation:
  Egészen más filozófia
  - Valódi „contract-alapú” megközelítés
  - Explicit interfészt definiál a külvilág felé, nem pedig egyszerű *[WebMethod]*-annotált metódusokat
  - Cél: a .NET platform sajátosságait „eltakarni”
  - Fel van készítve REST-re (jól!)

RESTful over WCF

- A WCF szolgáltatás fejlesztésének tipikus lépései:
  - Metódusok rögzítése az interfészben
  - Interfész implementálása (szerver oldalon)
  - Adattípusok publikálása metaadatként a kliensek felé
- „REST-esítés”
  - Speciális attribútumokkal látjuk el a metódusokat
  - Ezek URI template-eket írnak le
  - A HTTP kéréseket ezek alapján osztja szét a szerver

== annotáció a Javában

Példa: HelloWorld

- WCF
  - Interfész

```
[OperationContract]
string HelloWorld();
```

  - Implementáció

```
public string HelloWorld(){
  return "Hello World";
}
```

- „REST-esítés”
  - Csak az interfész változik – az implementáció nem!

```
[OperationContract]
[WebGet(UriTemplate="/SayHello"]
string HelloWorld();
```

Adattípusok megosztása

- Mi történik, ha összetett adatstruktúráink vannak?
  - Hogy adjuk át?
  - Metaadatokkal

```
public class Book{
 public int id {get;set;}
 public string title {get;set;}
}
```

- Csak egy helper osztály és egy attribútum kell!
- DataHelper osztály:

```
knownTypes.Add(typeof(Book));
```

- Attribútum az interfészben:

```
[ServiceKnownType("GetKnownTypes", typeof(DataHelper))]
```

- Innentől a kliens ismerni fogja a DTO-t
  - És itt is csak az interfész változott (minimálisan)!

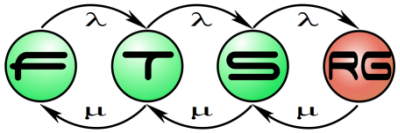
Linkek

- https://github.com/hstaudacher/osgi-jax-rs-connector
- http://www.oracle.com/technetwork/articles/javase/index-137171.html
- http://www.soapui.org/REST-Testing/working-with-rest-services.html
- http://code.google.com/p/rest-client/
- https://addons.mozilla.org/en-US/firefox/addon/restclient/
- http://restscratchpad.com/

References

- Carol McDonald: REST
  - http://www.slideshare.net/caroljmcdonald/td09restcarol
- S. Ghosh: Developing RESTful Web services with JAX-RS
  - http://www.slideshare.net/neilghosh/restfull-webservices-with-jaxrs
- R. Costello: Building Web Services the REST way
  - http://www.xfront.com/REST-Web-Services.html
- J. Cotamraju: Introduction to JAX-RS
  - http://www.slideshare.net/arungupta1/svcc-2010jaxrs

Végrehajtható munkafolyamatok



Végrehajtható munkafolyamatok

- BPMN
  - Business Process **Modeling Notation**
  - „Csak” modell és jelölésrendszer → nem végrehajtható
- Kérdés: lehet-e **végrehajtható** folyamatokat definiálni?
- Válasz: IGEN!
  - A munkafolyamat továbbra is egy modell
  - De szerkesztési időben forráskódra fordul (pl. Java)
- Elterjedt eszközök:BPEL, jBPM, MS WF

Pro's & con's

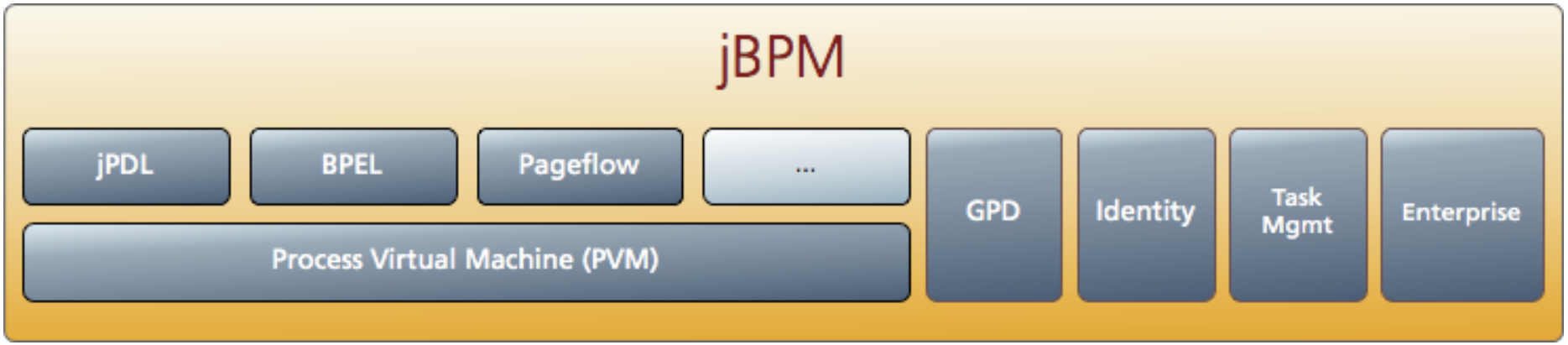
- Mi indokolja a folyamat alapú megközelítést?
  - Imperatív programozás esetén is egy munkafolyamat modell alapján fejlesztünk (specifikációban rögzített)
  - Könnyen átlátható (grafikus modellek)
- Mikor ne használjuk?
  - Ha túl nagy a folyamatmodell (nehezen kezelhető)
  - Ha túl egyszerű a funkcionalitás (nincs is valódi WF)
  - Teljesítménykritikus rendszereknél nem mindig jó
- További jellemző problémák
  - Nehezebb tesztelni és debugolni

jBPM5 filozófia

- Szakterület(Domain)-független megközelítés
  - Jellemzően általános elemek: elágazások, script node…
  - A fejlesztő a „problémára szabja” az elemkészletet
- WorkItem
  - A BPMN2.0 szabvány Service Task elemét valósítja meg
  - Ha a folyamatban egy ilyenre futunk, a hozzá kapcsolt WorkItemHandler végzi el a feladatot
  - A WorkItemHandlert regisztráljuk, az a toolbarra kerül
  - Ezután már egyszerűen használható a modellezésben
  - Előny: valóban „custom” funkcionalitás

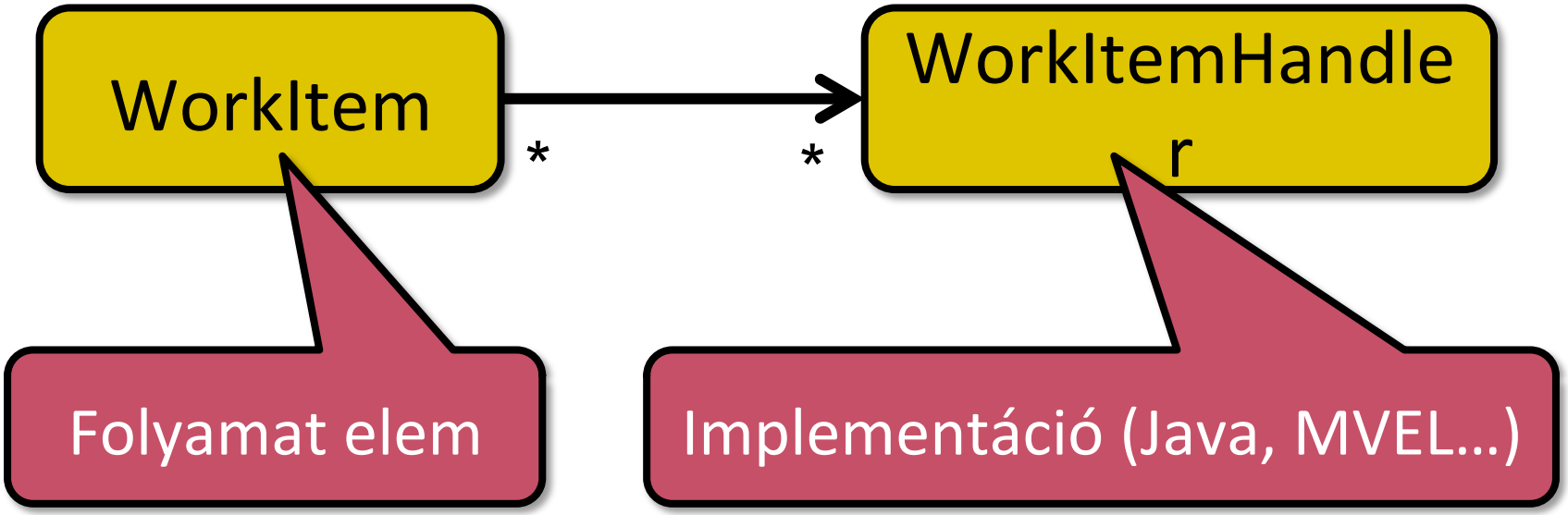
JBoss jBPM

- Platform végrehajtható folyamat leíró nyelvekhez
- Üzleti folyamatok integrációja Java alkalmazásba
  - jPDL, BPEL, Pageflow támogatás
- Process Virtual Machine
  - Natív folyamat végrehajtás



jBPM5 WorkItem példa

- Webszolgáltatást szeretnénk hívni
- De ilyen elem nincs
  - Külső szolgáltatást igénybe vétele: Service Task
  - A szolgáltatás WS jellege már domain specifikus
  - Lehetne OSGi szolgáltatás, platform feature, …



jBPM5 WorkItem példa

- **WorkItem elem definiálása**

```
[
  [
    "name" : "WSInvocation",
    "parameters" : [
        "URL" : new StringDataType(),
        "Message" : new StringDataType(),
        "ReturnValue" : new IntegerDataType(),
    ],
    "displayName" : "WSInvocation",
    "icon" : "icons/webservice.gif"
  ]
]
```

jBPM5 WorkItem példa

- **WorkItemHandler implementálása**

```
package com.handlers;

import org.drools.process.instance.WorkItemHandler;

public class WSInvocationWorkItemHandler implements WorkItemHandler {

    @Override
    public void abortWorkItem(WorkItem wi, WorkItemManager wim) {
        // TODO Auto-generated method stub
    }

    @Override
    public void executeWorkItem(WorkItem wi, WorkItemManager wim) {
        String address = (String)wi.getParameter("address");
        String message = (String)wi.getParameter("message");
        System.out.println("Calling WS @"+address+", with message "+message+"...");

        Client client = Client.create();
        WebResource webResource = client.resource(address+message);

        String s = webResource.get(String.class);

        System.out.println(s);

        wim.completeWorkItem(wi.getId(), null);
    }
}
```

jBPM5 WorkItem példa

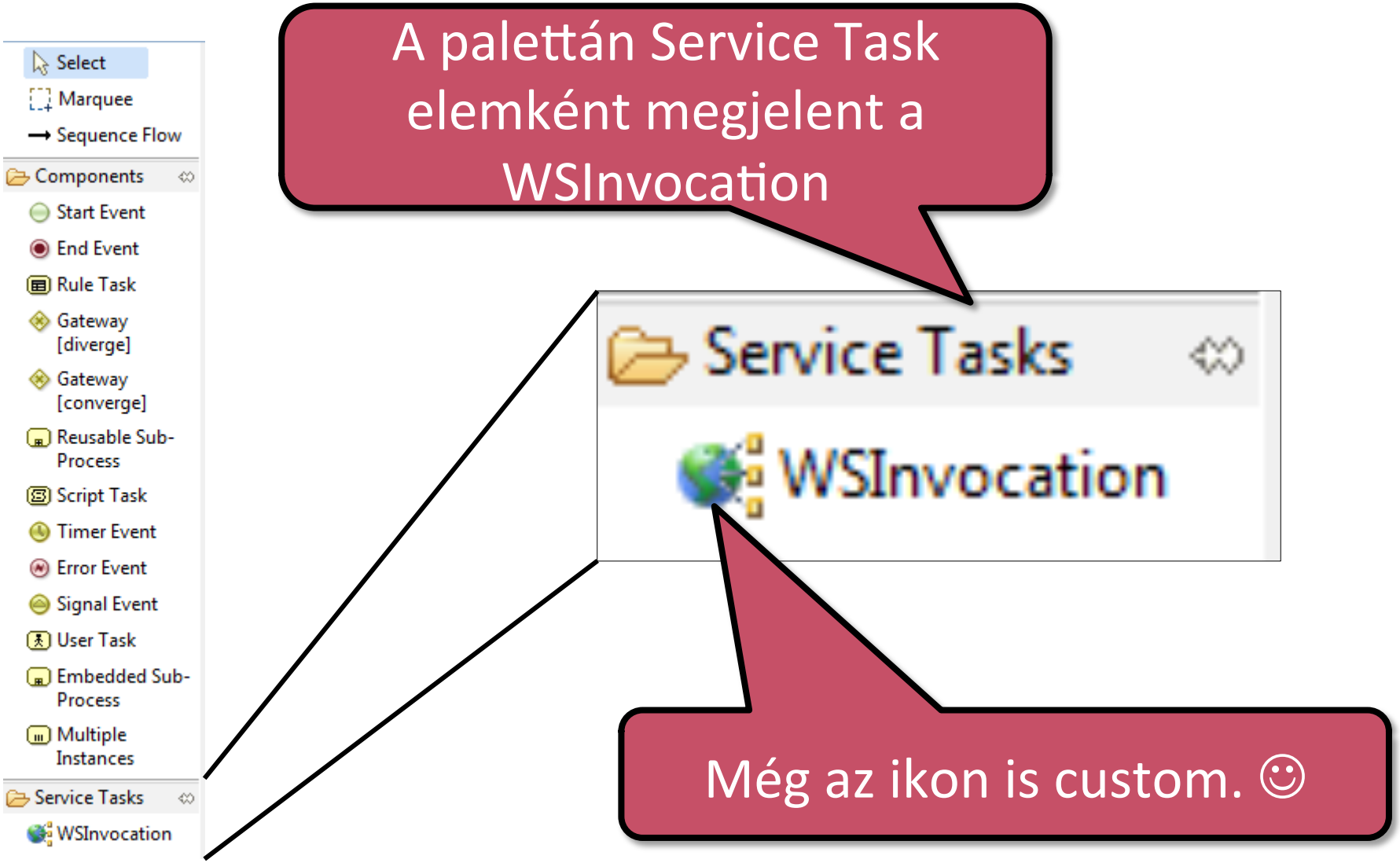
- **WorkItem és WorkItemHandler összerendelése**

```
ksession.getWorkItemManager()
        .registerWorkItemHandler(
              "WSInvocation",
              new WSInvocationWorkItemHandler()
        );
```

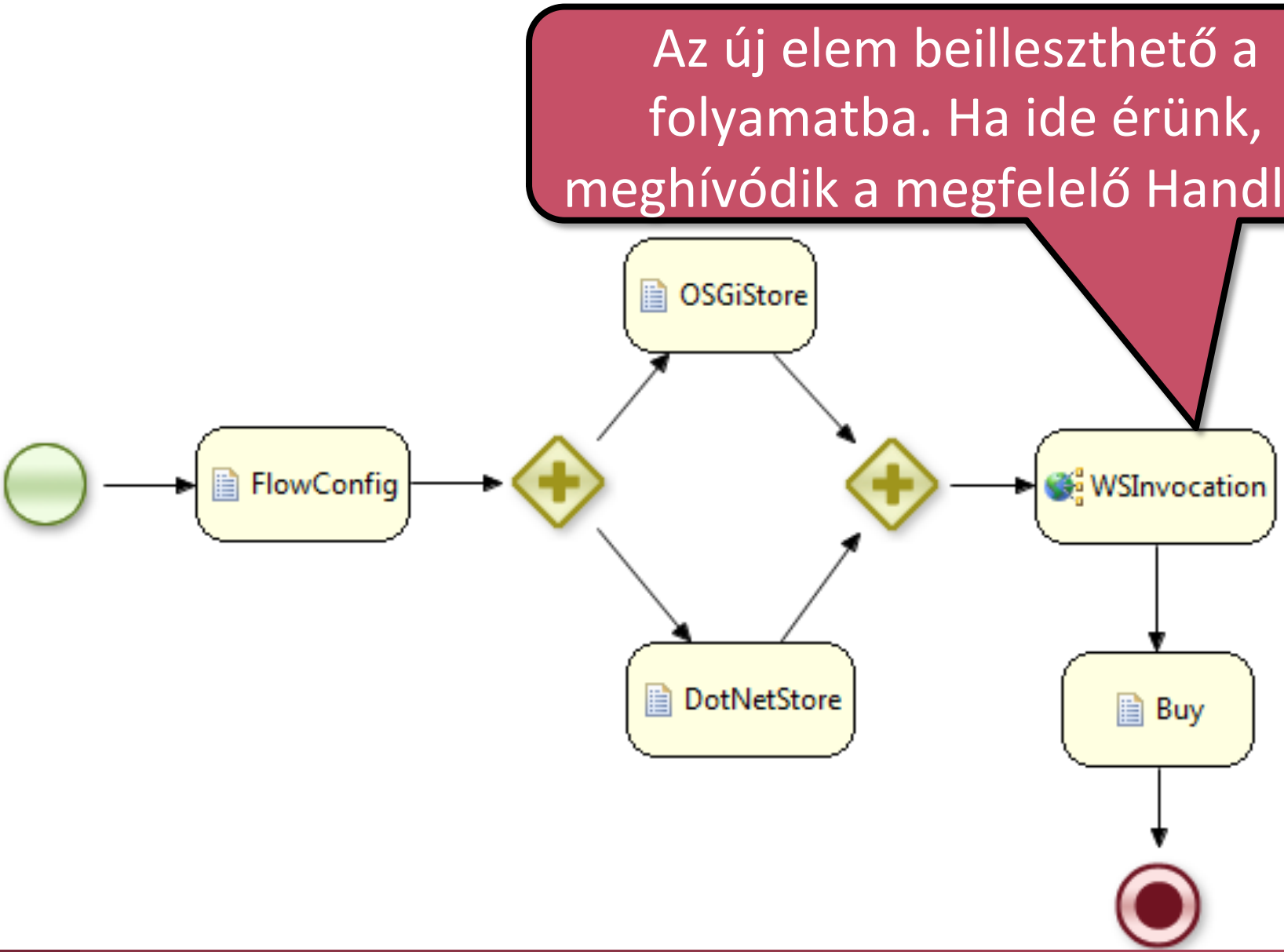
- **WorkItem regisztrálása a toolbarra**

```
drools.workDefinitions = MyWorkDefinitions.wid
```

jBPM5 WorkItem példa



jBPM5 WorkItem példa



„What's in the box?”

- Semmi meglepő: egy halom XML
- Példa: a folyamat első eleme (FlowConfig):

```
<scriptTask id="_9" name="FlowConfig" >
  <script>
    kcontext.setVariable("wsaddress",
                         "http://localhost:1047/Services");
    kcontext.setVariable("wsmessage", "getprice/1984");
  </script>
</scriptTask>
```

JBPM5 REST API

Process Management
Process related data.

POST	/gwt-console-server/rs/process/definition/{id}/new_instance
POST	/gwt-console-server/rs/process/instance/{id}/state/{next}
GET	/gwt-console-server/rs/process/definition/{id}/image
GET	/gwt-console-server/rs/process/instance/{id}/activeNodeInfo
GET	/gwt-console-server/rs/process/definition/history/{id}/nodeInfo
GET	/gwt-console-server/rs/process/definitions
POST	/gwt-console-server/rs/process/definition/{id}/remove
GET	/gwt-console-server/rs/process/definition/{id}/instances
GET	/gwt-console-server/rs/process/instance/{id}/dataset
POST	/gwt-console-server/rs/process/instance/{id}/end/{result}
POST	/gwt-console-server/rs/process/instance/{id}/delete
POST	/gwt-console-server/rs/process/tokens/{id}/transition
POST	/gwt-console-server/rs/process/tokens/{id}/transition/default

Task Lists
Access task lists

GET	/gwt-console-server/rs/tasks/{idRef}
GET	/gwt-console-server/rs/tasks/{idRef}/participation

Task Management
Manage task instances

POST	/gwt-console-server/rs/task/{taskId}/assign/{ifRef}
POST	/gwt-console-server/rs/task/{taskId}/release
POST	/gwt-console-server/rs/task/{taskId}/close
POST	/gwt-console-server/rs/task/{taskId}/close/{outcome}

Házi feladat menetrend

- 1. lépcső (1. jBPM gyakorlat, feb. 22.): "draft" folyamatmodell, script taskok, human taskok (1. HF)
- 2. lépcső (2. jBPM labor, március 27.): végrehajtható folyamatmodell (2. HF)
  - script taskok lecserélése service taskra
  - work itemek definíciója (WSInvocation)
  - work itemek felvétele, paraméterátadás megtervezése
  - workitemhandler implementáció, integráció az egyes szolgáltatásokkal