

Hands-on: Developing Scout Business applications

Spring, Hibernate, Maven

11100110101010010101010011011
10100111100101001010010110110
00100101010001001001000110010
01000101100010010001010010101
00110010111001000100101010001
01110101001001110010001101011
10010001010100101010100101011
00101010101010101010101000010
01010101000101100010001000011



- Intro
- Use Cases
- Architecture
- Front-end: Scout
- Back-end: Spring and Hibernate
- Industrialization of build process (Maven)
- Outlook

Intro

- Development during last 8 months
- Going live 2014 and 2015
- Work in progress

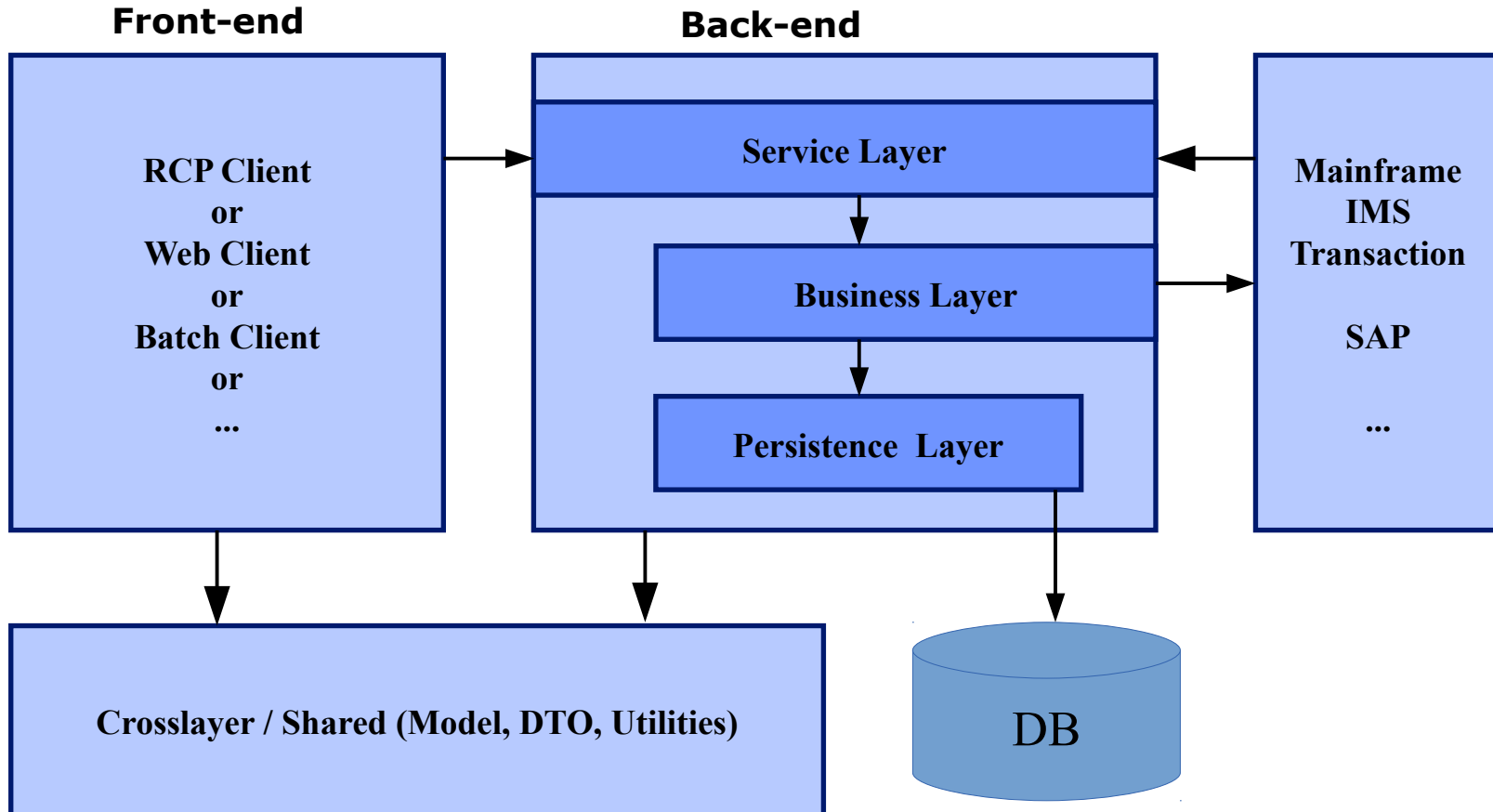
Use Cases

- requirements
 - Rich Client, RCP/SWT
 - Multi Layer Backend
 - Multi Module, base project and dependent projects
Core (shared), product A, product B
 - CI, Maven
 - Mainframe (run down -> DB independency)
 - Deployment environment: Citrix (RCP Client), IBM Websphere
Application Server, IBM DB2 database
 - Maintainability (>20 man years of development)

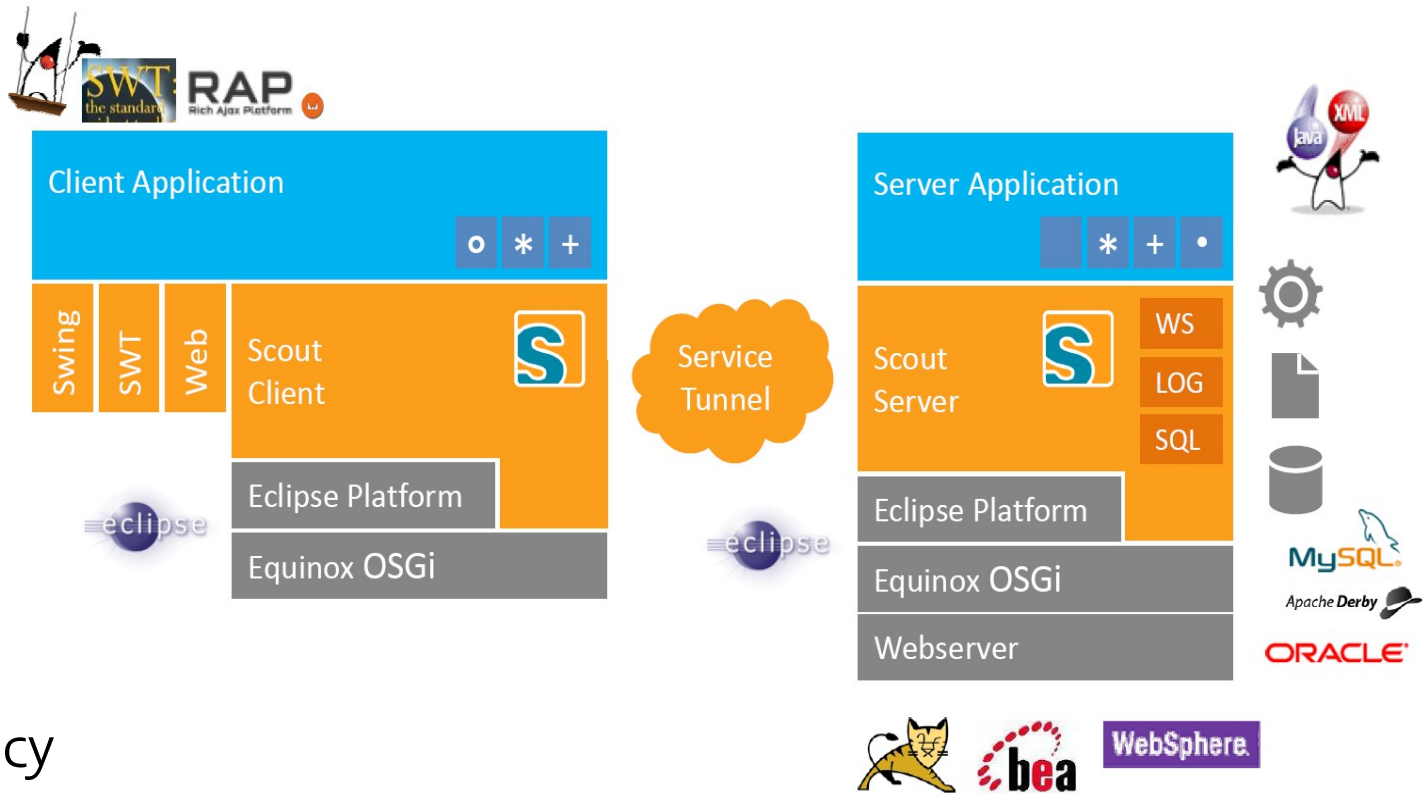
Use Cases

- requirements
 - Replace home made UI abstraction in RCP and RAP (Scout)
 - Replace home made persistence abstraction (Spring)
 - Industrialization of build process (Maven)

Architecture



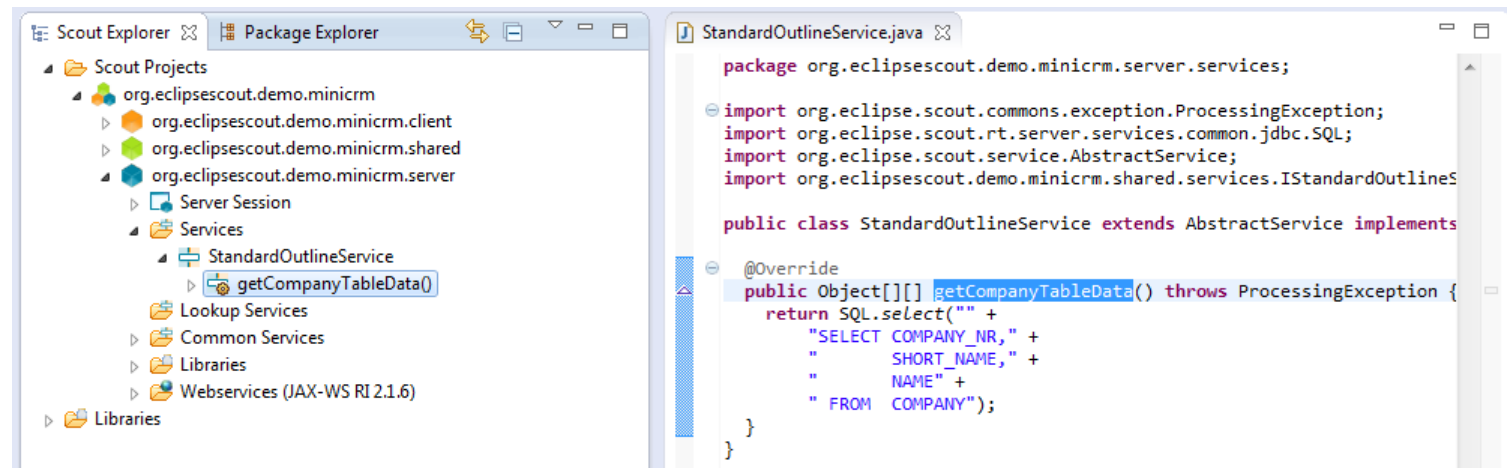
Front-end: Scout



- efficiency
- SWT, Swing, Web, Mobile
- Client Server communication

- Security

Front-end: Scout



Back-end

- Technology: EJB, WebService, ... ?
- OSGI bundles together with Scout Server
- Standard JAR projects with manual MANIFEST.MF
- Problem: location of MANIFEST.MF – maven vs. eclipse

Back-end

- The Scout way to call a Scout Service:
`SERVICES.getService(IStandardOutlineService.class)`
- Our way to call a backend Service
`BACKEND.getService(IBasicDataService.class)`
- `BACKEND.java`

```
ApplicationContext ctx = new ClassPathXmlApplicationContext(new String[] {"...", "..."});  
bean = ctx.getBean(serviceInterfaceClass);
```
- Service bean instantiated by Spring -> deeper layers can use injection: `@Service`, `@Resource` ...

Back-end

- JPA generic vs. Hibernate specific
- Annotation vs. XML config
- OSGI...ClassLoader issues
- “normal” issues with legacy databases (composite-id,...)
- Transaction boundaries, lazy loading

Back-end

@Override

```
public void execLoad() throws ProcessingException{  
    ICompanyService service = SERVICES.getService(ICompanyService.class);  
    CompanyFormData formData = new CompanyFormData();  
    exportFormData(formData);  
    formData = service.load(formData);  
    importFormData(formData);  
    setEnabledPermission(new UpdateCompanyPermission());  
}
```

New Service Operation

Create a new Service Operation.

Operation Name	<input type="text" value="getCompanyTableData"/>		
Return Type	<input type="text" value="Object[][]"/>		
Name Arg 1	<input type="text"/>	Type	<input type="text"/>
Name Arg 2	<input type="text"/>	Type	<input type="text"/>

Industrialization of build process

- Tons of projects
 - Parents
 - Targets
 - Features (+ Test)
 - jar / OSGI bundles
 - Plugins (+ Test)
 - Products, Repositories
- Sequence
 - Build Parents, Targets -> Maven Repo
 - Build Backend OSGI bundles (POM) -> Maven Repo
 - Build Remaining (Tycho) -> P2 Repo
- P2 Repo

Lessons learned and outlook

- Positive
 - Efficiency
 - Flexibility
- Problems
 - OSGI
 - Target
 - P2 Repo
- Next steps
 - Finalize Hibernate lazy loading
 - Check elimination of Scout form data, Object[][]
 - Spring – Gemini blueprint
 - Client notifications – multi node