A Lightweight Component-based Development Approach for Enterprise Applications

Jingang Zhou\textsuperscript{1, 2}, Dazhe Zhao\textsuperscript{1, 2}, Jiren Liu\textsuperscript{2}

\textsuperscript{1}College of Information Science and Engineering, Northeastern University
\textsuperscript{2}State Key Laboratory of Software Architecture (Neusoft Corporation)
Component-based Development (CBD)

• What
  – Software is (partially) composed by some existing software parts/modules instead of being developed from scratch.

• Recognized 40 years ago
  – A basic format of software reuse
    • Plug-able, substitutable, easy to maintain
  – A natural way to divide complex systems (divide & conquer)
    • Parallel working,

• Benefits
  – Decrease development cost
  – Increase software quality
  – Faster to markets
Component-based Development (CBD)

- Ivica Crnkovic “Without a possibility to reuse system components already developed it will be not possible to meet the requirements from the market with requirements and technologies changing extremely fast.”

- However (problems)
  - What is a component? (Many component models exist, which is for large enterprise applications?)[6, 8]
    - Academic / Industrial
    - OO / architectural units
    - Embedded systems / enterprise systems
  - For industrial practices, the components must be supported in implementation level, e.g., COM/EJB
    - Language characteristics for CBD
    - Development tools
Enterprise Application Characteristics

- Software size is bigger and bigger
- Involving many different domains and technologies
- Depends on Java EE Application Servers
- Low maturity / level 1 / modularity
  - Complex and implicit dependencies
  - Example from Accenture & Infosys[4]

Alex Miller: “Dependency management is one of the most important (and challenging) features of modern software development.”
Typical CBD & Limitations

• Java/EJB/Java Bean
  – Fine granularity
  – Target for special aspect of an EA
  – No modular support (class, package, Jar) [3]
  – Java 7? When?

• MJ [3]
  – A Modular System for Java
  – Customized classloader and modified Web Server (i.e., Tomacat)

• SAP NetWeaver
  – Heavyweight, costive, locked in specific technology

• OSGi / Eclipse
  – Emerging shift paradigm, mainly for AS providers
  – Problems exist in a purely OSGi-based EA [9]
Our Goal

• To provide a lightweight CBD approach for EA
  – Lightweight: easy to adopt with less efforts
  – Pragmatic: leveraging many developer friendly tools and techniques
  – Effective/Applicable: confirmed by our applications

• Our focus
  – On how to develop a true component and component reuse
  – Not on CBD process and other specific aspects, e.g., quality assurance
### Enterprise Component Model

- **Project/Product**: the final software to be delivered
- **Software component (SC)**: a logical category for grouping related DCs, mainly for delivery
- **Development Component (DC)**: the basic unit for reuse, an aggregation of classes, web resources (optional)
- **Module**: an inner modular part resides in a DC
- **Development object (DO)**: basic development formats in language level

Some concepts (e.g., SC/DC) are borrowed from SAP, please c.f. [http://help.sap.com/saphelp_nw70ehp1/helpdata/en/1c/bca99c220c0e45a1cae3c4ccc4346/content.htm](http://help.sap.com/saphelp_nw70ehp1/helpdata/en/1c/bca99c220c0e45a1cae3c4ccc4346/content.htm)
Enterprise Component Model

- Criteria for the classification
  - Application size
  - Component granularity
    - Delivery
    - Development and reuse
  - Intuitive
- So that we can get a natural mapping from UML models to the component hierarchy
Component Hierarchy Mapping

<table>
<thead>
<tr>
<th>UML</th>
<th>Comp.</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Product</td>
<td>Java Project</td>
</tr>
<tr>
<td>subsystem</td>
<td>SC</td>
<td>Feature Project</td>
</tr>
<tr>
<td>component</td>
<td>DC</td>
<td>Plug-in Project (OSGi Bundle)</td>
</tr>
<tr>
<td>package</td>
<td>Module</td>
<td>directory</td>
</tr>
<tr>
<td>class</td>
<td>DO</td>
<td>As is</td>
</tr>
</tbody>
</table>
OSGi bundle Preliminary

- OSGi is a modular framework above Java language level
- A bundle (OSGi component) has its own classloader to control the class space
- Bundle dependencies are explicit declared in the MANIFEST.MF file of the bundle
- A bundle can only allowed to access the public part of another bundle with explicit import (or require) directive
- A bundle can declare its public part via the export directive
- ...

Bundle-Name: %pluginName
Bundle-SymbolicName: org.eclipse.equinox.supplement
Bundle-Version: 1.3.0.qualifier

Export-Package:
org.eclipse.osgi.framework.log;version="1.0",
org.eclipse.osgi.service.datalocation;version="1.3",
org.eclipse.osgi.service.debug;version="1.1",
org.eclipse.osgi.service.urlconversion;version="1.0",
org.eclipse.osgi.storagemanager;version="1.0",
org.eclipse.osgi.util;version="1.1",
org.eclipse.osgi.framework.debug;x-internal:=true,
org.eclipse.osgi.framework.util;x-internal:=true,
org.eclipse.osgi.framework.internal.core;x-internal:=true,
org.eclipse.osgi.util.tracker;x-internal:=true

Bundle-RequiredExecutionEnvironment: J2SE-1.4,
CDC-1.0/Foundation-1.0,
J2SE-1.3

Import-Package: org.osgi.framework,
org.osgi.util.tracker,
org.eclipse.osgi.framework.log,
org.eclipse.osgi.service.datalocation,
org.eclipse.osgi.service.debug,
org.eclipse.osgi.storagemanager,
org.eclipse.osgi.util
Component Development

- CBDE
  - Project Viewer
    - An integrated viewer showing the components in an hierarchical and collaborative way.
    - Allows to create project, SCs, DCs, Modules, and DOs.
    - Built on Eclipse Common Navigator Framework.
    - 7 Java classes, nearly 900 Loc for the basic functionality.
    - Different colors on SCs indicate different ownership of them, to facilitate SoC.

All components own and control all its own parts!
Component Assembly

- Component-based software product line
- An assembly is not a component, but a group of components with collaboration
- An assembly description file servers the identity of the assembly
- A bottom-up way to broaden components reuse scopes

```
Component : NotifyByEmail {
   //GUID
   Id: be97b25772024ab2a9946f975cabd7d0
   Type: Assembly
   Body {
      //Event (version = 4.0)
      Ref = 03ad53d6e71e44f1bb58295be66c3c14
      //Mail (version = 4.0)
      Ref = 0f34cd8d5172462d85424f34c0de57a9
   }
   Composition_rules {
      //for simplicity, we use component or feature
      //name instead of Id in the rules below.
      Event recommends SimpleText
      HighReliability requires NeedPersistence
   }
}
```
Application Development

- Also component-based
- Uniform with component development
- Typical steps include:
  - Component browsing
  - Component selection
  - Component loading into PV
  - Create new components
  - Categorize the newly created components and upload
- The whole application is modular, to facilitate components extraction
Target Platform

- A refactored concept from Eclipse OSGi-based applications.
- Encapsulate the 3rd party (binary) components

Logical layer

Our development components

Depend-on

Target Platform (Libs, Jars)

Deployed with

Java EE Application Server
Application Deployment

- Design time component architecture **mismatch** runtime application architecture.
- We must obey the Java EE specification and application server environment in an industrial practices.
- CBDE (actually, *Components builder*, extends the default *Java Builder* in Eclipse).
  - Component dependency analysis to avoid circular dependencies.
  - Class compilation (same to the default).
  - Configuration files building and resources merging and distribution to the target directories specified as the Java EE application structure.
Case Study

- UniEAP
  - A privatory software platform targets EAs development, which includes many frameworks and components
  - The core part, nearly 120K LOC Java
  - 7 technical staff, 3 months

- Modularization
  - Web resource
  - Java code
  - Configuration files
  - Data schema

<table>
<thead>
<tr>
<th>SCs</th>
<th>DCs</th>
<th>Size</th>
<th>1K~3K</th>
<th>3K~7K</th>
<th>&gt;7K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td></td>
<td></td>
<td></td>
<td>kernel</td>
<td></td>
</tr>
<tr>
<td>TechComp</td>
<td>cache, exception</td>
<td>menu</td>
<td>code-list, RIA</td>
<td>variability</td>
<td></td>
</tr>
<tr>
<td>BizComp</td>
<td>data-pipe</td>
<td>mail, event</td>
<td>scheduler, security, log, monitor</td>
<td>biz-data</td>
<td></td>
</tr>
<tr>
<td>CusComp</td>
<td></td>
<td></td>
<td>web-console</td>
<td>organization</td>
<td></td>
</tr>
</tbody>
</table>
Benefits

• Developer may be familiar with the feature and plug-in projects, as well as the Eclipse development activities
• The OSGi-based component is strongly supported by many industrial vendors
• The components are natural gained from system analysis and have proper granularity and business oriented
• No special and lock-in technology, just modular design

pragmatic

lightweight
limitations

• Component model
  – Simple (OSGi-based)
• Design time components rather runtime
  – Cannot substitute a component on the fly
  – Malicious calls
    • Aspect-based constraints violation checking (AOP)
  – JAR Hell may still alive
• Still needs some efforts to make Eclipse environment a more CBDE.
Results & Conclusion

• Not yet another component model but leveraging exist ones.
• Refactor some popular techniques and develop some useful tools for our CBD concepts.
• OSGi-based design time components, avoid some risks in runtime due to the immature of OSGi in enterprise application domain.
• Eclipse-based and related tools make our CBD lightweight and pragmatic.
• Plugin-based domain-specific components make the CBDE a software production line / software factory [11].
• Keep ROI on OSGi-based components since more and more application server vendors are refactoring their product with OSGi and OSGi bundles may be directly supported in the future.
• easy to evolution for more advanced development paradigm, e.g., software product lines.
Questions?

10. [http://help.sap.com/saphelp_nw70ehp1/helpdata/en/1c/bca99c220c0e45a1cae3c4ccc43346/content.htm](http://help.sap.com/saphelp_nw70ehp1/helpdata/en/1c/bca99c220c0e45a1cae3c4ccc43346/content.htm)