Software Components and Distributed Systems

INF5040/9040 Autumn 2015
Lecturer: Amir Taherkordi (ifi/UiO)

October 5, 2015

Outline

1. Introduction to Components
2. Basic Design Concepts
3. Distributed Components
4. Main Technologies for Distributed Components
5. Summary
Distributed Components 1. Introduction

Long History of Components

- 1968 NATO Workshop on Software Eng.
  - D. McIlroy introduced the notion of components
  - to further industrialize software industry
  - *His definition:*
    - Components: **families of routines**
    - with varying degrees of precision, robustness, generality, etc.
    - an industry-oriented viewpoint

Software Components

- **Some Definitions**
  - A unit of composition with contractually specified interfaces and explicit dependencies.
    (Clemens Szyperski)
  - A piece of self-contained, self-deployable code, assembled with other components through its interface.
    (Wang and Qian)
  - A nearly independent, and replaceable part of a system with a clear function, implementing a set of interfaces.
    (Philippe Krutchen, Rational Software)

- For example: JavaBeans, COM, CORBA, OSGi
**Distributed Components**

### Why Components?

- A natural way for building systems, e.g., automotive industry
- Industrialized viewpoint to software production?
- Avoid handmade software products
- Main goals:
  - *Conquering Complexity*: increase in software size
  - *Managing Change*
  - *Software Reuse*: black-box, gray-box and white-box reuse

### Three Basic Design Concepts

#### I. Component Model

- Binding of provided and required interfaces
  - Reflects direction of method calls (Not the direction of data flow)
  - Required interface
    - A set of method calls a component potentially may issue
  - Support for distribution?
    - When the binding can be made across address spaces and computers
II. Connection Models and Composition
- Integrate components to generate a new component with pre-defined composition operators.
- Composition is the fundamental method for construction, extension and reuse of components.
- In contrast to inheritance in object-oriented models.
- Main connectors:
  - Method-based: composition of components
  - Event-based

III. Deployment Models
- The process and activities for component installation and any necessary configuration.
- E.g., EJB produces a XML-based deployment descriptor.

Designing a Component Platform
- The underlying foundation to construct, assemble, deploy and manage components.
- Defines rules for deployment, composition and activation of components.
- To deliver and deploy components: a standardized archive format that packages component code and meta-data.
- Embraces three design concepts:
  - component model, connection model, and deployment model.
- designed as a set of contractually specified interfaces.
- Contracts agreed between components and a component platform.
Contracts as the key design element

What is a contract?
- Set of provided interfaces: Some may be required by the component platform
- Set of required interfaces: must be offered by other components available on the platform
- Pre and post conditions/invariants
- Extra-functional requirements: transactions, security, performance, ...
- Functions defined both syntactically and semantically
  ```
  int add(int a, int b)
  pre: a + b <= Integer.MAXINT
  post: result' = a + b
  ```
- Extra-functional requirements
  - Guarantees: Response within 10 ms
  - Conditions: Needs 1000 CPU-cycles
  - Transaction requirements: e.g., create new transaction when component is invoked, serializable, ...

Components vs. Objects

- Objects
  - one mission: encapsulation for reusability
  - reusable class libraries, e.g., Foundation Classes for Java or C++
- Objects for reuse in the large?
  - fine-grained classes with complex relationships and dependencies
  - Difficult to take classes out of the lib and reuse

<table>
<thead>
<tr>
<th>Object-Oriented</th>
<th>Component-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>classes and object</td>
<td>components</td>
</tr>
<tr>
<td>data types and hierarchies</td>
<td>interfaces and composition</td>
</tr>
<tr>
<td>implementation technology</td>
<td>packaging &amp; distribution technology</td>
</tr>
<tr>
<td>tightly coupled: low-level reuse</td>
<td>loosely coupled: high-level reuse</td>
</tr>
<tr>
<td>limited sets of supported services: security, transactions, ...</td>
<td>more support for high-level services: security and transactions, ...</td>
</tr>
</tbody>
</table>
Distributed Components

2. Design Concepts

An Example: JavaBeans

- Java-based Component model
- A JavaBean component: Properties, Methods, Events, Customization, and Persistence.
- Requirements for developing beans:
  - implementing the Serializable interface to store/retrieve a bean
  - Properties: exposed through the " set" and " get" methods
  - Events: exposed through public "add" and "remove" methods
- Example: JavaBean Events

Example: JavaBeans

- User Action
- Trigger an event
- source: SourceComp
  - addXListener(listener: XListener)
  - handler(event: XEvent)

1. A listener object is an instance of a listener interface
2. Register by invoking source.addXListener(listener);

Distributed Components

- Advantages of distribution
  - Load sharing
  - Increased availability
  - Heterogeneity
  - Replication
  - ...
- Distributed components
  - characteristics of components + functionality of middleware systems
  - inter-process communication across machine boundaries
- An evolution of distributed objects
Distributed Components 3. Distributed Components

Revisit Distributed Objects

- Objects that
  - reside in separate address spaces
  - their methods are remotely accessible: client & server objects
- Distributed object middleware
  - Infrastructure for access to remote objects transparently
  - based on the Remote Procedure Call (RPC)
- Application logic entangled with logic for life cycle management, transactions, security, persistence, etc.
- Object developer
  - particular implementations of services for particular settings

Implicit dependencies
- It is not clear what dependencies an object have on other objects
- Interaction with the middleware
  - Many low-level details
- Lack of separation of distributed concerns
  - Security, transactions, coordination, etc.
- No support for deployment
- For example in CORBA and Java-RMI
  - How to deploy the components of my application?
  - Which services will be available on a given host?
  - Who activates my objects?
  - Who manages the life-cycle of my objects?
Implicit Middleware

- Better support for “separation of concerns”:
  
  ![Diagram showing separation of concerns]

- Changing middleware services separately without changing the application code

![Diagram showing implicit middleware]

Component-based Middleware

- To realize implicit middleware: How?
- Distributed Components + Container

  **Distributed Component**
  
  The designer only focuses on the component logic, not burdened with the implementation of location, persistence, transactional capabilities and security.

  **Container**

- Responsibilities of the container
  
  - life cycle management, system services (e.g., transactions), security
  - dynamic deployment and activation of new components
    
    - e.g., resolve dependencies dynamically or activate components requested in method calls
    
    - Front-end for remote communication including interception of incoming invocations (cf. implicit middleware)

- Middleware that supports the container pattern: **Application Server**
3. Distributed Components

Application Servers: Key Players

<table>
<thead>
<tr>
<th>Technology</th>
<th>Developed by</th>
<th>Further details</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebSphere Application Server</td>
<td>IBM</td>
<td>[<a href="http://www.ibm.com">www.ibm.com</a>]</td>
</tr>
<tr>
<td>Enterprise JavaBeans</td>
<td>SUN</td>
<td>[java.sun.com XI]</td>
</tr>
<tr>
<td>Spring Framework</td>
<td>SpringSource (division of VMWare)</td>
<td>[<a href="http://www.springframework.org">www.springframework.org</a>]</td>
</tr>
<tr>
<td>JBoss</td>
<td>JBoss Community</td>
<td>[<a href="http://www.jboss.org">www.jboss.org</a>]</td>
</tr>
<tr>
<td>CORBA Component Model</td>
<td>OMG</td>
<td>[Wang et al. 2001]</td>
</tr>
<tr>
<td>JOnAS</td>
<td>OW2 Consortium</td>
<td>[jonas.ow2.org]</td>
</tr>
<tr>
<td>GlassFish</td>
<td>SUN</td>
<td>[glassfish.dev.java.net]</td>
</tr>
</tbody>
</table>

Distributed Components - Main Technologies

- **Sun/Oracle**
  - Defined the *Enterprise Java Beans (EJB)* specification as part of their Enterprise Edition of the Java 2 platform.

- **OMG**
  - Defined the *CORBA Component Model (CCM)*, providing a distributed component model for languages other than Java.

- **Microsoft**
  - Defined the Distributed *Component Object Model (DCOM)*, extending Microsoft's COM and supporting distributed communication under Microsoft's COM+ application server.
Distributed Components 4. Technologies: EJB

Enterprise JavaBeans

- A server-side component model
- Three-tier architecture

Beans in EJB: to capture business logic
- EJB container: supporting key distribution services: transactions, security and lifecycle
  - container-managed: injecting calls to the associated services
  - bean-managed: developer takes more control over these services

EJB Component Model

- Bean: a component offering business interfaces (remote and local)
  - Session beans: stateless and stateful
  - Message-driven beans: listener-style interface

- Bean implementation
  - Plain Old Java Object (POJO) with annotations, e.g.:
    ```java
    @Stateful public class eShop implements Orders {...}
    @Remote public interface Orders {...}
    ```
  - A significant number of annotations for container services
Distributed Components 4. Technologies: EJB

An Example: Transactions

```java
@Stateful
@TransactionManagement(BEAN)
public class eShop implements Orders {
    @Resource javax.transaction.UserTransaction ut;
    public void MakeOrder (...) {
        ut.begin();
        ...
        ut.commit();
    }
}
```

Bean-Managed

```java
@Stateful
@TransactionManagement(Container)
public class eShop implements Orders {
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void MakeOrder(...) {
        ...
    }
}
```

Container-Managed

Distributed Components 4. Technologies: EJB

Other Aspects of EJB

- Dependency injection in container:
  - managing and resolving the relationships between a component and its dependencies, e.g.

```java
@Resource javax.transaction.UserTransaction ut;
```

- EJB Interception:
  - to associate particular action(s) with an incoming call on a business interface, e.g.

```java
public class eShop implements Orders {
    public void MakeOrder (...) {...
    @AroundInvoke
    public Object log(InvocationContext ctx) throws Exception {
        System.out.println("invoked method:" +
                      ctx.getMethod().getName());
        return invocationContext.proceed();
    }
}
```
A lightweight component model
- Programming with interfaces
  - Uniform model for provided and required interfaces
  - Explicit representation of the architecture
- No support for deployment, container patterns, etc.
- Configurable and reconfigurable at runtime
- Programming language agnostic model
  - Implementations of the model available in several programming languages (Java, C, C#, Smalltalk, Python)

Fractal Component Model – cont’d
- **Server** (provided) and **Client** (required) interfaces
- Composition: **bindings** between interfaces
  - **Primitive Binding**: client and server within the same address space
  - **Composite Binding**: arbitrarily complex architectures (consisting of components and bindings) implementing communication between two or more interfaces potentially on different machines
- Component model is **hierarchical**
  - a component: subcomponents and associated bindings
  - subcomponents may themselves be composite
- System is fully configurable and reconfigurable: including components and their interconnections
Distributed Components 4. Technologies: Fractal

Fractal: Example

- Describing components through Architecture Description Language (ADL)

```xml
<definition name="HelloWorld">
  <interface name="r" role="server" signature="Runnable"/>
  <component name="client">
    <interface name="r" role="server" signature="Runnable"/>
    <interface name="s" role="client" signature="Service"/>
    <content class="ClientImpl"/>
  </component>
  <component name="server">
    <interface name="s" role="server" signature="Service"/>
    <content class="ServerImpl"/>
  </component>
  <binding client="this.r" server="client.r"/>
  <binding client="client.s" server="server.s"/>
</definition>
```

Fractal: Example – cont’d

- Resulting Architecture

![Resulting Architecture Diagram](image)
Distributed Components 4. Technologies: Fractal

Fractal: Component Structure

Summary