Software components and distributed systems

INF 5040/9040 autumn 2013

lecturer: Frank Eliassen

Literature

- Coulouris chap 8.4, 8.5 and 8.6

- Recommended
A history of middleware

- First generation middleware
  - Exclusively based on *client-server model*
  - Examples include Open Group’s DCE
- Second generation middleware
  - Based on *distributed object technology*
  - Examples include CORBA and Java RMI
- Third generation middleware
  - Based on *component technology*

Issues with object-oriented middleware

- 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
Background for Java and CORBA platforms

- Known problems with 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?

=> We need a standard development, deployment and runtime environment for distributed objects (CORBA, Java)

Explicit middleware: lack of “separation of concerns”

- Programs directly towards a middleware API
- Application logic entangled with logic for life cycle management, transactions, security, persistence, etc.
**Implicit middleware: better support for “separation of concerns”**

- Logic for life cycle management, transactions, security, persistence, etc. managed by the middleware
- Requirements for middleware services declared separately and can later be changed without changing the application code
- Middleware can be changed without changing the application code

**Component technologies**

- What is a component [Szyperski]?
  
  “a unit of composition with contractually specified interfaces and explicit context dependencies only”

  “in this context, a component can be deployed independently and is subject to third-party composition”
Rationale for components

- Time to marked
  - Improved productivity/ reduced complexity
  - Focus on reuse
- Programming by assembly rather than by engineering
  - Reduced requirements to knowledge
- Most important advantage: development of server side?
  - (cf. EJB or CORBA Component Model - later)

Designing a component platform

- A standard development, deployment and runtime environment can be designed as a set of contractually specified interfaces
- Contracts agreed between components and a component platform
- Component platform defines the rules for deployment (installation), composition and activation of components.
- For delivering and deploying a component is required a standardized archive format that packages component code and meta-data
Contracts

- What is in a contract?
  - Set of provided interfaces.
    - Some of these may be required by the component platform
  - Set of required interfaces.
    - These 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, ...

Composition

- Components and composition
  - Composition is the fundamental method for construction, extension and reuse of component-based software development
  - In contrast to (implementation) inheritance in object-oriented approaches

“Components are made for composition”
Connection-oriented programming

- Composition of pre-manufactured components
- Binding of incoming and outgoing interfaces
  - provided/required interfaces
  - Reflects direction of method calls
    - Not the direction of data flow
- Outgoing interface
  - The method calls a component potentially may issue
- Support for distribution?
  - When the binding can be made across address spaces and computers

Third party composition

- The composition can be done by a third party external to the components themselves (loading and binding)
- Example
  - Connections (bindings), outgoing and ingoing interfaces
  - Connects (binds) “matching” interfaces
  - Binding can be done during load/run time by a third party (e.g., a tool)
    - Can typically be realized by setting an appropriate attribute of the component with the outgoing interface (for C1, methods: setB, setV)
Composition: Reuse and assembly of components

- **Software architecture**: components, interfaces, connections (between interfaces)

![Software architecture diagram]

An implementation of a component platform is often called a **container**

Responsibilities of the container
- life cycle management, system services, 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**
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 XII]</td>
</tr>
<tr>
<td>Spring Framework</td>
<td>SpringSource (a 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>

Application Servers

- **Advantage**
  - Comprehensive support for one style of distributed programming
    - Three-tier approach, hides complexities

- **Disadvantages**
  - Mandates a particular architectural style
    - E.g. three-tier architecture
  - Large and complex systems that works best on high-end servers
    - Performance and resources overhead
**Enterprise Java Beans (EJB)**

- Component architecture for deployable server side components in Java.
- EJB components contain business logic only – no system-level programming
- EJB 3.0: based on Metadata facility in Java 5
  - **annotations** in source code
    - Example: `@TransactionManagement (CONTAINER)`
- EJB is **managed**
  - Container handles: transactions, security and lifecycle;

---

**EJB in the n-Tier Architecture**

- **Browser**
- **Client Applet**
- **Client Application**
- **EJB Server**
- **Database**
- **Legacy Systems**
- **Clients**
- **Application**
- **Data**

You Are Here

Frank Eliassen, IfI/UiO
**EJB component model**

- A bean is a component offering one or more **business interfaces** (provided interfaces)
  - Session Beans (stateful, stateless)
  - Message-driven beans
- Bean implementation: Plain Old Java Object (POJO) with annotations
- Annotations for **Dependency injection** (required interfaces)
  - @Resource `interface-name inf-n`
- **Interception**
  - Method Invocations
    - @AroundInvoke
    - @Interceptors
  - Lifecycle events (Creation and deletion of components)
    - @PostConstruct, @PreDestroy

**Lightweight Component Model**

- Component Models as EJB are heavyweight and prescriptive
  - Cannot be used for different classes of DS, such as peer-to-peer
  - Not suitable for constrained and embedded devices
- Need for a more stripped-down, domain-independent and minimal component model
  - Fractal
  - OpenCOM
  - OSGi
Fractal

- Programming with interfaces
  - Uniform model for *provided* and *required* interfaces
  - Explicit representation of the architecture
- No support for deployment, full container patterns, etc.
- Configurable and reconfigurable at runtime

Fractal Component Model

- **Server** (provided) and **Client** (required) interfaces
- Composition: **bindings** between interfaces
  - **Primitive Binding**: client and server interfaces 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**
- System is fully configurable and reconfigurable: including components and their interconnections
Fractal

Architecture Description Language (ADL)

<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

Resulting architecture

Frank Eliassen, IfI/UiO 25
The structure of a Fractal component

Composing adaptive software using components

- Importance and interest in adaptive software is increasing dramatically
  - mobile, ubiquitous and autonomic computing
- Components play a major part
- Compositional adaptation (component reconfiguration)
  - dynamic adaptation of architecture of component-based application
    - change component impl
    - redeploy component
    - parameter adaptation
    - change overall architectural framework
    - combinations of the above
**MUSIC middleware: Adaptability through component frameworks**

![Diagram of MUSIC middleware components]

**Resources:** ist-music.berlios.de

---

**Summary**

- **Components**
  - Programming according to LEGO-principle
  - Contractually specified interfaces and composition
  - Support for connection oriented programming

- **Component architecture**
  - Contractually specified interfaces between components and application servers
  - Realizes “implicit middleware”
  - Application servers: Java: EJB, CORBA: CCM ...
  - Light-weight component models: Fractal, OSGi ...

Frank Eliassen, IF/UiO