Outline

- Introduction
- ACME Architectural Description Language
- Java Bean Component Model
- COM, DCOM, MTS and COM+
- CORBA Component Model (CCM)
- .NET Component Model
- OSGI Component Model
Component Interface and Connections

- **A Short Historical Perspectives ...**
  - Programming languages, can be seen from either
    - The run-time point of view or,
    - The design and reuse perspective

- **ADLs primarily address the issues related to the early phases of software engineering:**
  - Design
  - Analysis

- They identify a number of concepts, such as:
  - Architecture, configurations, connectors, bindings, properties, hierarchical models, style, static analysis and behavior.
Component Interactions

Interactions with Traditional Software Entities

- Interactions with Other Components
- Interactions with Component Infrastructure

- Traditional software entities
- Components
- Component Infrastructure
### Majors Steps in CBD Lifecycle

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Phase</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>Definition</td>
<td>Designer</td>
</tr>
<tr>
<td>Assembly</td>
<td>Assembly</td>
<td>Architect</td>
</tr>
<tr>
<td>Implementation</td>
<td>Implementation</td>
<td>Developer</td>
</tr>
<tr>
<td>Lifecycle</td>
<td>Packaging, Deployment</td>
<td>Administrator</td>
</tr>
<tr>
<td>Framework, run-time support</td>
<td>Execution</td>
<td>End User</td>
</tr>
</tbody>
</table>
ACME Architectural Description Language

- Components and Ports
- Connectors and Roles
- Systems and Attachments
- Representations and Bindings
- Properties, Constraints, Types and Styles
Components and Ports

- **Components**
  - Represent the computational elements and data stores of a system.

- **Ports**
  - Are the points of interaction between a component and its environment.
Connectors and Roles

- **Connectors**
  - Represent interactions between components such as method calls or an SQL connection between a client and a database server.

- **The interface of a connector is defined as a set of roles**
The structure of a system is specified by a set of components, a set of connectors, and a set of attachments.

**Attachment**
- Links a component port to a connector role.
Representations and Bindings

- Component
- Connector
- Port
- Role
- Attachement
- Binding
ACME makes it possible to annotate each entity with an arbitrary set of properties.

ACME includes a constraint language based on FOPL (first-order predicate logic).

One or more constraints can be attached to any architectural entity.

ACME enables us to define the style of architecture, that is, the type of the system.
# Majors Steps in CBD Lifecycle

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Phase</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>Definition</td>
<td>Designer</td>
</tr>
<tr>
<td>Assembly</td>
<td>Assembly</td>
<td>Architect</td>
</tr>
<tr>
<td>Implementation</td>
<td>Implementation</td>
<td>Developer</td>
</tr>
<tr>
<td>Lifecycle</td>
<td>Packaging, Deployment</td>
<td>Administrator</td>
</tr>
<tr>
<td>Framework, run-time support</td>
<td>Execution</td>
<td>End User</td>
</tr>
</tbody>
</table>
Java Bean Component Model

- Key Features
- Interface of a Component
- Implementation of a Component
- Components Assembly
- Packaging and Deployment
Key Features

- Bean Box

- "A Java Bean is a reusable software component that can be manipulated visually in a builder tool".

- The Java Bean was designed for the construction of graphical user interface (GUI).

- Explicitly tailored to interact in two different contexts:
  - At composition time, within the builder tool.
  - At execution time, with the runtime environment.
Interface of a Component

- This model defines four types of port:
  - methods,
  - properties,
  - event sources and
  - event sinks called listeners.
Implementation of a Component

- Most bean components are implemented by a simple Java object, the object being encapsulated in the component, but there are more sophisticated implementations possible.
  
  - **Wrapping a legacy object.**
    - Done through a set of explicit bindings between object methods and component ports.
  
  - **Multiple-objects implementation.**
    - Encapsulated a collection of objects collaborating in the realization of the component.
  
  - **Dependency on traditional entities.**
    - Called another object outside the component boundary,
Implementations of Bean Components

A simple implementation

A more complex implementation
Assembly is one of the key features of Java Bean though no specific solution is provided.

- Different ways of assembling components are supplied.
Packaging and Deployment

- Java Beans define a model for packaging components into archives.
  - Includes the definition of dependency relationships between the package items.

- The customization code can be more complex than the component itself!

- Each package item can be marked "Design Only", so that they can be removed in a final application.
COM, DCOM, MTS and COM+

- Interfaces and Assembly
- Implementation
- Framework
- Lifecycle
Interfaces and Assembly

- A COM interface is seen as a C++ virtual class and takes the form of a list of data and function declarations without associated code.

- All interfaces are descendants of the IUnknown interface (QueryInterface, AddRef, and Release).
Implementation

- A COM object is a piece of binary code, written in any programming language, as long as the compiler generates code following the binary interoperability convention.

- COM supports two composition techniques:
  - **Containment**: one COM object contains other COM object(s).
  - **Aggregation**: outer object can expose interfaces of the inner object as if the outer object implemented them.
Framework

- **Standard interfaces**
  - IUnknown
  - IDispatch

- **A simple run-time** that interprets the calls for creating COM objects, returning interface handles, and managing the reference count for releasing objects.

- DCOM extends COM with distribution based on RPC mechanism.

- MTS (Microsoft Transaction Server) extends DCOM with the container approach.
Lifecycle

- COM, and COM+ are strictly execution time and binary component models.
- No lifecycle issues are explicitly supported.
CORBA Component Model (CCM)

- Interface and Assembly
- Framework: The Container Approach
- Lifecycle
A component interface is made of ports divided into:
- Facets
- Receptacles
- Event sources
- Event sinks
Framework: The Container Approach

- Services can be made available to components without having to change that component’s source code.
Lifecycle

- **CCM is the best effort to date:**
  - To gather the advances made in different fields,
  - To include a wide spectrum of lifecycle activities, while still claiming efficiency and heterogeneity capabilities,

- However, the whole does not provide the feeling of being as “simple” as claimed.
.NET Component Model

- Interfaces and Assembly
- Implementation
- Framework
- Lifecycle
Interfaces and Assembly

- Programming language approach for component programming.

- The program contains the information related to the relationships with other “components”, and that the compiler is responsible for generating the information needed at execution.

- There is no explicit concept of connection but rather the traditional list of imported and exported resources.
Implementation

- A component (assembly) is made of modules, which are traditional executable files (DLL).
- Modules cannot be assemblies, thus the .NET model is not hierarchical.
Framework

- .NET relies on the traditional programming approach: the framework is seen as the language run-time support.

- Transaction control relies on MTS.
Lifecycle

- Assemblies (and their modules) are local to an application, and thus different DLLs with same name can run simultaneously.

- Each assembly has a versioning information about itself and about the assemblies it depends on.
  - Version control is delegated to the dynamic loader, which selects the “right” version.

- Significantly improve the application packaging and deployment.

- Early lifecycles phases do not seem to have received much attention.
OSGI Component Model

- Components

- Interface of a Bundle Component

- Assembly of Bundle Components

- Implementation of a Bundle Component
Components

- A bundle uses *three kinds of ports* to express its interactions with:
  - Traditional technology
  - Other components
  - The run-time environment

- Bundles may listen to events published by the framework such as the insertion of a new component in a system.
Interface of a Bundle Component

- Package export
- Package import
- Service interface
- Service use

Ports

Interface of a bundle component
Assembly of Bundle Components

- A system is an evolving set of bundle components.

- A bundle component publishes a service interface
  - It can attach to it a set of properties describing its characteristics.

- A component requires an interface for its use,
  - It will select one via a query expression based on these properties.

- This flexibility also has its counterpart
  - There is no guarantee than the service will continue to be available.
Implementation of a Bundle Component

- JAR archive containing:
  - Service components
  - Java packages
  - Other resources files