Chap 7. Component-based Development

Part 7.1 Introduction to Component-based Development

1. Introduction
2. Component-Based Development Processes
3. Component Models
4. The CORBA Component Model (CCM)
1. Introduction

Overview
- Component-based development (CBD) emerged in the late 1990s as a reuse-based approach to software systems development.
- It was motivated by the frustration that OO development had not led to extensive reuse as originally suggested.
- Components are more abstract than object classes and can be considered to be stand-alone service providers.
- Components are defined by their interfaces and in general can be thought of as having two related interfaces: provides and requires.
Component Categories and Abstraction

- Software components provide a vehicle for software artifacts reuse, and thereby may be used at all the levels of the software life cycle: analysis, design, implementation, and deployment.

- Hence, there are various kinds of software components:
  - Conceptual components: components at the analysis and design level.
  - Implementation components: development work product components such as source code files, data files etc.
  - Deployment components: involved in an executable system, such as dynamic libraries and executables.
- Components may also exist at different levels of abstraction:

\(\text{\textit{Functional abstraction}}\): the component implements a single function such as a mathematical function. The \textit{provides} interface is the function.

\(\text{\textit{Casual groupings}}\): the component is a collection of loosely related entities that might be data declarations, functions etc.

\(\text{\textit{Data abstractions}}\): the component represents a data abstraction or class in an OO language; the \textit{provides} interface consists of operations to create, modify and access the data.

\(\text{\textit{Cluster abstractions}}\): the component is a group of related classes that work together (called framework); the \textit{provides} interface is the composition of the \textit{provides} interfaces of the objects involved.

\(\text{\textit{System abstraction}}\): the component is an entire self-contained system (also called COTS product); the \textit{provides} interface is an API defined to allow programs to access the system commands and operations.
Constructs

-Principal constructs used in software component modeling:

«Component»: complex, and physical objects that interact with their environments through one or more ports.

«Port»: boundary object that implements some of the interfaces through which a component interacts with its surroundings.

«Connector»: abstraction for communication channels that interconnect two or more ports of components.

«Roles»: boundary object that implements the interfaces through which a connector interacts with its surroundings.

«Protocol»: defines the valid sequence of messages between connected ports.
Component-Based Development Processes

- Component-oriented development can be integrated into a system development process in one of two ways: opportunistic reuse and development with reuse.

**Opportunistic Reuse**

1. **Design System Architecture**
2. **Specify Components**
3. **Search for Reusable Components**
4. **Incorporate Discovered Components**

- The specifications are used to find reusable components which are then incorporated in the architecture.

- Although this approach may result in significant reuse, it contrasts with the approach adopted in other engineering disciplines.
The system requirements are modified according to the reusable components available; the design is also based around existing components.

Since this requires some tradeoff, the design is less efficient than a special purpose design; however, lower costs of development, rapid delivery, and increased system reliability should compensate for that.
3. Component Models

- A software component conforms to a component model and can be independently deployed and composed without modification according to a composition standard.

Component Model

- A component model defines a set of standards for component development, deployment, and evolution.

- The main competing component models currently available include:
  - OMG’s CORBA Component Model (CCM),
  - Microsoft’s Distributed Component Object Model (DCOM)
  - Microsoft DotNET Framework
  - SUN Microsystems JavaBeans and Enterprise JavaBeans (EJB)
Basic Elements of a Component Model

- Basic elements of a component model include standards for interfaces, naming, meta data, customization, composition, evolution, and deployment.

<table>
<thead>
<tr>
<th>Standards for</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interfaces</strong></td>
<td>Specification of component behavior and interfaces; definition of an Interface Definition Language (IDL)</td>
</tr>
<tr>
<td><strong>Naming</strong></td>
<td>Global unique names for interfaces and components.</td>
</tr>
<tr>
<td><strong>Meta data</strong></td>
<td>Information about components and interfaces.</td>
</tr>
<tr>
<td><strong>Interoperability</strong></td>
<td>Communication among components from different vendors, and/or implemented in different languages.</td>
</tr>
<tr>
<td><strong>Customization</strong></td>
<td>Interfaces for customizing components.</td>
</tr>
<tr>
<td><strong>Composition</strong></td>
<td>Interfaces and rules for combining components.</td>
</tr>
<tr>
<td><strong>Evolution Support</strong></td>
<td>Rules and services for evolving components.</td>
</tr>
<tr>
<td><strong>Packaging and deployment</strong></td>
<td>Packaging implementation and resources needed for installing and configuring a component.</td>
</tr>
</tbody>
</table>
Component Model Implementation
-Dedicated set of executable software elements required to support the execution of components that conform to the model.

-Provide:
÷ A run-time environment
÷ Basic Services
÷ Horizontal services that are useful across multiple domains
÷ Vertical services providing functionality for a particular domain for software components.
4. The CORBA Component Model (CCM)

Overview of the CCM
- The goals of the CORBA Component Model (CCM), like any other component model (e.g., DCOM, EJB etc.) is to facilitate reuse of CORBA applications.

- The CCM extends the standard CORBA Interface Definition Language (IDL) by including specific features for component description.

- The CCM also introduces a new declarative language, named the *Component Implementation Definition Language (CIDL)*, which is used by code generators to generate code needed to deploy the components (in containers).

- Developers have to deal only with the development of the components and their inherent logic and functionality.
CBD Process using the CCM

IDL/CIDL/PSDL

Design

IDL/CIDL/PSDL

Functional code

Compiler

Stubs/
Skeletons

Component
descriptor

Packet

Language Tools

Implementation

Packaging Tool

Softpkg
descriptor

Home properties

Component
descriptor

CORBA
Component
Package

Assembling Tool

Assembly description

CORBA
Assembly
Package

Deployment Tool

BDP Process using the CCM

Home properties

Component properties

Integrator

Provider

Dispatcher

Administrator
- The CCM defines a component type to represent component instances.

- Component type definitions consist of a collection of ports definitions. The CCM defines 2 kinds of ports: facets and configuration ports.

  - Facets: consist of a set of interfaces that define the functionality supported or provided by the component.

  - Configuration ports: correspond to a set of interfaces that specify how a component may interconnect and communicate with other components.
**Configuration Ports**
Several kinds of ports supported by the CCM, namely *receptacle*, *attribute*, *emitter*, *publisher*, and *consumer*.

- **Receptacles**: specify the external dependencies of the component, by describing the interfaces used by the component.

- **Attributes**: describe the properties of the component, and thereby serve as medium for their configuration and customization.

- **Event sources**: specify the events published by the component; two forms of events can be generated by the component:
  - *Publisher*: events for which the component is exclusive provider
  - *Emitters*: events that share event channels with other event sources

- **Event sinks**: specify the events consumed by the component.
CCM Examples

An example CCM Component
An example CCM Component With IDL Specification

```idl
interface Sell, Buy;
// Define an equivalent, supported interfaces
component Stock_Exchange supports Sell, Buy {
    provides Stock_Quote;   // Facet
    consumes Buy_Offers;    // Event Sinks
    consumes Sell_Offers;
    publishes Price_Change; // Event Source
    uses SEC;               // Receptacle
    ...                     // Other definitions
}
```
Example of CCM Components Interactions

Note: CCM components interact through port mechanisms
**Component Container**

- Represents the run-time environment of component instances.
- The CORBA component container implements component access to global system services such as transactions, security, events, and persistence.
- The container reuses the existing CORBA infrastructure. In doing so, the inherent complexity of CORBA is hidden both to the developer and to the container.

- Container and component instances interact through two kinds of interfaces:
  - *Internal API*: a set of interfaces provided by the container to component implementations.
  - *Callback Interfaces*: a set of interfaces provided by component implementations to the container.