Component-Based Software Engineering
Dr. Rami Bahsoon
School of Computer Science
The University Of Birmingham
r.bahsoon@cs.bham.ac.uk
Office 112 Y9- Computer Science

Week 2. Essentials to Component-Based Software

Learning objectives

• Brief motivation to components-based software
• Components-based essentials
  – Components
  – Components characteristics
  – Interfaces
  – Specification
  – Components vs. objects
  – Component Models
  – Frameworks
  – Middleware
• Exercise: Sketching simple components and define their interfaces...

Component-based software engineering

• Component-based software engineering (CBSE) is an approach to software development that relies on software reuse.
• It emerged from the failure of object-oriented development to support effective reuse.
  – Single object classes are too detailed and specific
• Components are more abstract than object classes and can be considered to be stand-alone service providers
• Application development becomes the selection, adaptation and composition of components rather than implementing the application from scratch

Next week in depth!
Component-based software engineering

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Process stages
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.
- Emphasis is on Reuse → Reuse-oriented development
- This approach is becoming increasingly used as component standards have emerged

Concentration on the business issues:
"Around 30% of the development effort is spent on the infrastructure that add no value"

**Application specific**

**Standard Recusable parts**

- GUI
- Communication
- Data model
- Deployment
- ....

**INFRASTRUCTURE**

Warm-up brainstorming exercise

- How do you define a "component"?
- What are the characteristics of a "component"?
- What is a component comprised of?
- How do you distinguish a "component" from an "object Class"?
What is a component?

• Several definitions of a component in the literature:
  - Agreement that a component is a piece of “potentially reusable software”... We need to clarify this notion:
    • Software?
      - Computer programs AND associated requirements documents, architecture, design, and testing artefacts, documentations etc.
      - Reuse spans these artefacts:
        » Libraries & repositories of software components

• There are many definitions of components:
  - Consideration of CBSE from different viewpoints and focus on different aspects of software engineering
    • Design viewpoint: components as reusable design parts
    • Architecture viewpoint:
      - High-level representation(structural and behavioral) of a service/computation
      - UML & Architecture Description Language(ADL) components
    • Implementation viewpoint:
      - Components confirmed to a specific component model
      - At run time could be binary packages, distributed components
    • Business viewpoint:
      - buy vs. build (Commercial-off-the-shelf COTS components)

A software component

• Provides a service: implementation-independent
• Need not be compiled
• Executable
• More abstract than classes
Component as a service provider

- Components provide a service without regard to where the component is executing or its programming language
- The component is an independent, executable entity that can be made up of one or more executable object classes
- It does not have to be compiled before it is used with other components
- The services offered by a component are made available through an interface
- All component interactions take place through that interface

What is a component comprised of?

- Some code...
  - The code represents the operations that the component will perform when invoked

- An interface...
  - The interface of a component should provide all the information needed by its users for the component to be composed or deployed
  - The specification of a component is to a big extent a specification of its interfaces
  - The component interface is published and all interactions are through the published interfaces

Component as a "service"?

- Booking service
- Flight promotion service
- Hotel booking
- Car hire service
- E.g., Web services composition
Component definition

- Szyperski (Component Software beyond OO programming)
  - A software component is
    - a unit of composition
    - with contractually specified interfaces
    - and explicit context dependencies
  - A software component
    - can be deployed independently
    - and is subject to composition by third-parties

Composition?... Contractual interfaces?... Context Dependency?... Deployed?... Independence?
Component definition

- Councill and Heinmann:
  - A software component is
    - a software element that conforms to a component model
    - and can be independently deployed and composed without modification according to a composition standard.

Software architecture point of view

- The software architecture of a program or computing system is the structure or structures of the system, which comprise
  - software components [and connectors],
  - the externally visible properties of those components [and connectors] and the relationships among them. [and the environment]

Bass L., Clements P., and Kazman R., Software Architecture in Practice

Component characteristics

- Standardised
  - A component has to conform to some standardised component model
    - This model may define component interfaces, component meta-data, documentation, composition and deployment

- Independent
  - A component should be independent
    - It should be possible to compose and deploy it without having to use other specific components
      - In situations where the component needs externally provided services, these should be explicitly set out in a 'requires' interface specification
Component characteristics

- **Composable**
  - For a component to be composable,
    - All external interactions must take place through publicly defined interfaces
    - In addition, it must provide external access to information about itself such as its methods and attributes

- **Deployable**
  - To be deployable,
    - A component has to be self-contained and must be able to operate as a stand-alone entity on some component platform that implements the component model
    - This usually means that the component is a binary component that does not have to be compiled before it is deployed
  - Explicit context dependencies
    - Specification of the deployment and run-time environments
      - Which tools, platforms, resources, other components are required?

- **Documented**
  - Components have to be fully documented so that potential users of the component can decide whether or not they meet their needs
  - The syntax and, ideally, the semantics of all component interfaces have to be specified
Other characteristics for components

- Marketable entity
  - A component is a self-contained binary piece of software that one can typically purchase in the open market
- Not a complete application
  - A component can be combined with other components to form a complete application (or an even bigger component)
- It can be used in unpredictable combinations
  - A component can be used in ways that were totally unanticipated by the original developer
  - Components can be combined with other components using plug-and-play

Component representation in UML

Stereotypes

They mean the same:
- a component called "Order"
  - UML version 2.0
  - UML previous versions 1.4

Interfaces

- Act as a "contract"
- Requires
- Provide
Interfaces

- Interfaces
  - Can include operations, method calls, etc. that pass messages to component to ask for services
  - Can declare exceptions to indicate that the operation did not perform successfully

Component interfaces

- **Contractual interfaces** - A specification attached to an interface that mutually binds the clients and providers of the components
  - Functional Aspects (API)
    - Pre- and post-conditions for the operations specified by API
      - Preconditions denoting the constraints which need to be met by the client
      - Postconditions denoting the constraints which the component promises to fulfill in return
      - A component may also additionally satisfy global constraints called invariants
  - Non-functional aspects (different constrains, environment requirements, etc.)

Provides interface Requires interface

Component

Provides interface Defines the services that are provided by the component to other components

Requires interface Defines the services that the component uses from the environment

Provide Require dependency

Provide Requires dependency

Component-Based Software Engineering
Dr R Bahsoon
Component interfaces

- Provides interface
  - Defines the services that are provided by the component to other components
- Requires interface
  - Defines the services that specifies what services must be made available for the component to execute as specified

![Component interfaces diagram]

Component composition

![Component composition diagram]

Example of UML Component Diagram

![Example of UML Component Diagram]

Shows a component’s relationship with other components:
The "lollipop" and "socket" notation must also include a dependency arrow (as used in the class diagram). On a component diagram with lollipops and sockets, note that the dependency arrow comes out of the consuming (requiring) socket and its arrow head connects with the provider’s lollipop.
Exercise 1

Sketch the components and interfaces corresponding to the given services.

Exercise 2: Sketch a CBS for data collector

Sensor Management
- Add sensor
- Remove sensor
- Start Sensor
- Stop Sensor
- List all sensor

Data collector from sensors

A Data collector component

Requires
- sensorManagement
- sensorData

Provides
- addSensor
- removeSensor
- startSensor
- stopSensor
- testSensor
- initialise
- report
- listAll
Legacy system components

- Existing legacy systems that fulfill a useful business function can be re-packaged as components for reuse.
- This involves writing a wrapper component that implements provides and requires interfaces then accesses the legacy system.
- Although costly, this can be much less expensive than rewriting the legacy system.

Interface Description Language (IDL)

- For distribution to work in an architecture, we need to define interfaces that both client and server objects (i.e., components) can understand and use easily, regardless of their platform, operating system, programming language, network connection, or other characteristics etc.
- An interface definition must specify:
  - The operation to be performed,
  - All the input and output parameters with their types allowing client and server to encode and decode values for their travel over the network,
  - Since it’s always possible for something to go wrong when we make an invocation, the language should support robust exception handling,
  - Extra non-functional mechanisms.

Ref: Object Management Group

Why do we use IDL?

Reference: W. Emmerich, UCL
Why do we use IDL?

![Diagram of programming languages](image1)

Reference: W. Emmerich, UCL

Objects and Components

- Pfister and Szyperski:
  - View a component as a collection of objects in which the objects cooperate with each other and are intertwined tightly
- D’Souza and Wills:
  - Assert that if a class were packaged together with the explicitly defined interfaces which it requires and implements then this class would be a component
Objects and Components

- But there are important differences:
  - Components have no externally observable state and can thus not be distinguished from a copy, whereas objects have a local state
  - Components have a more extensive set of interaction mechanisms than objects which usually use the messaging mechanism
  - Components are often larger units of granularity than objects, and have complex actions at their interfaces

Components revisited...

- A software component is a software element that
  - confirms a component model
  - can be independently deployed
  - composed without modification according to a composition standard
- A component model defines specific interaction and composition standards

Component models

- A component model is a definition of standards for component implementation, documentation and deployment
- Examples of component models
  - EJB model (Enterprise Java Beans)
  - .NET model
  - Corba Component Model
- The component model specifies how interfaces should be defined and the elements that should be included in an interface definition
- Different application domains have different needs for component-based systems
  - Different non-functional properties: performance, security, reliability, scalability, etc.
Elements of a component model

Component model is a definition of standards for component implementation, documentation and deployment

Architecture of components models

- J2EE - rich services in platform
- CORBA: emphasis is on integration of legacy components
- Components may be written in several languages.

Frameworks

- CBSE means that we build software by "putting pieces together"
- Frameworks provide the context in which the pieces can be used
  - A skeleton of an application which can be customized by an application developer
  - Microarchitecture which provides an incomplete template for systems within a specific domain
  - Describe a larger unit of design and that they are more specialized than patterns - i.e. can be reused
  - Frameworks are also suitable units for sharing and reusing architectures
Component frameworks

- Frameworks in general describe a typical and reusable situation at a model level,
  - a component framework in particular describes a "circuit-board" with empty slots into which components can be inserted to create a working instance
- Not to be confused with a component model:
  - a component model defines a set of standards and conventions used by the component developer
  - a component framework is a support infrastructure for the component model

Middleware support

- Component models are the basis for middleware that provides support for executing components
- Component model implementations provide:
  - Platform services that allow components written according to the model to communicate
  - Horizontal services that are application-independent services used by different components
- To use services provided by a model, components are deployed in a container:
  - This is a set of interfaces used to access the service implementations

Component technologies implement the standards...
Component-Based Software Engineering
Dr R Bahsoon

### Middleware support

- **Horizontal services**
  - Component management
  - Concurrency
  - Transaction management
  - Persistent storage

- **Platform services**
  - Addressing
  - Interface definition
  - Exception management
  - Component communications

Application-independent services used by different components

Allow components to communicate

---

### Summary

- **The basis is the Component**
- Components can be assembled according to the standards specified by the component model
- Components are assembled through their interfaces
- **A Component Composition** is the process of assembling components to form an assembly, a larger component or an application
- Components are performing in the context of a component framework
- All parts conform to the component model
- A component technology/middleware is a concrete implementation of a component model

Reference: Ivica Crnkovic

---

Component-Based Software Engineering
Dr R Bahsoon