Introduction to Component-Based Software Engineering

Gentilmente cedido por
Ivica Crnkovic
Mälardalen University
Department of Computer Science, Sweden

Problems of software development

- The size & complexity of software increases rapidly
- Single products become part of product families
- Software is updated after deployment
- Demands of decreasing time to market
- Costs of software development increasing

Observations of the practice of software engineering

- About 80% of software development deals with changing (adaptation, improvement) of existing software
- Time to market is an important competitive advantage:
  - Importance of incorporation of new innovations quickly
- System should be built to facilitate changes
- Easy removal and addition of functionality
- Systems should be built to facilitate reuse
  - Easy integration of existing functions
- Provision of approach, technologies to facilitate
  - Reuse, easy update and modification of software

Requirements:
- Provision of approach, technologies to facilitate
  - Reuse, easy update and modification of software
Answer: Component-based Development

- Idea:
  - Separate development of components from development of systems
  - Build software systems from pre-existing components (like building cars from existing components)
  - Building components that can be reused in different applications

- Component-based Software engineering - supporting all aspects of activities in lifecycle of components and component-based systems

Component-based software systems

Software Component Definition (I)

Szyperski (Component Software beyond OO programming)

- A software component is
  - a unit of composition
  - with contractually specified interfaces
  - and explicit context dependencies only.
- A software component
  - can be deployed independently
  - it is subject to composition by third party.
A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party. –Clemens Szyperski

What is a contract?

- Contract - 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.
  - Non-functional aspects (different constraints, environment requirements, etc.)

A component may provide / implement several interfaces

What is an explicit context dependency?

- Context dependencies - Specification of the deployment environment and run-time environment
  - Example: Which tools, platforms, resources or other components are required?
  - Do existing component models have support for declaring the dependencies?
  - Is it possible to verify if the environment comply with the context required?
A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party.

- Late binding - dependencies are resolved at load or run-time.
- Replacing of a component independent of the client (main application) if the contract is not broken.
- Delegation - interaction with a weak coupling (for example no inheritance).

Which problems can occur in relation to late binding? How can we guarantee that a replacement of a component will not affect other parts of the system?

Example: Interface description: (M)IDL

(Microsoft) Interface Definition Language

```csharp
// uuid:00112233-ABBA-ABBA-ABBA-BADBADBADBAD,
object ]

interface IAddressList {
    HRESULT addAddress ([in] name, [in] address);
    HRESULT deleteAddress ([in] name, [in] address);
}
```

- language independent interface specification
- can be compiled into language dependent code skeletons

Components and Interfaces - UML definition

Component – a set of interfaces required (in-interfaces, the "socket")

Interface – set of operations
Operations – input and output parameters of certain type

Component – a set of interfaces provided (out-interfaces, the "lollipop")
Contractually specified interfaces in a UML metamodel

Component specification

- Components are described by their interfaces
- (A black box character)

Another definition

- 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.

Variety of component models

- The generalized definition allows different component models
  - In different domains there are different requirements and constraints
    - Different interactions (architectural styles)
    - Different extra-functional properties
    - Different integration and deployment policies

Main principles: (1) Reusability

- Reusing components in different systems
- The desire to reuse a component poses few technical constraints.
  - Similar systems architecture
  - Good documentation (component specification…)
  - A well-organized reuse process
  - …

Main principles: (2) Substitutability

- Alternative implementations of a component may be used.
- The system should meet its requirements irrespective of which component is used.
- Substitution principles
  - Function level
  - Non-functional level
- Added technical challenges
  - Design-time: precise definition of interfaces & specification
  - Run-time: replacement mechanism
Main principles: (3) Extensibility

- Comes in two flavors:
  - Extending system functionality by adding components that are part of a system
  - Extending system functionality by increasing the functionality of individual components

- Added technical challenges:
  - Design-time: extensible architecture
  - Run-time: mechanism for discovering new functionality

Main principles: (4) Composability

- Composition of components
  - \( P(c_1 \circ c_2) = P(c_1) \circ P(c_2) \)
- Composition of functions
- Composition of extra-functional properties

- Many challenges
  - How to reason about a system composed from components?
  - Different type of properties
  - Different principles of compositions

Compositional Reasoning

- Calculating properties of a system by combining properties of its constituents (components)

- Compositional reasoning: Function
  - If \( P(C) \) of program \( C \) is a function from input to output (pipe & filter)
  - then the composition is modeled as a functional composition:
  - \( S = C_1 \circ C_2 \)
  - Then \( P(S) = P(C_1) \circ P(C_2) \)
Predictable assembly

- Functional composition is not always possible
- Question with extra-functional properties
  - Example: dynamic memory usage $M$
  - If $S = C_1 \circ C_2$, then what is the composition $M(S) = M(C_1) \circ M(C_2)$
- $M$ is not defined only by properties $M(C_i)$, but also on properties of the platform "scheduling policy for example"
- Information supplied with $C_i$ is not enough

Predictable assembly = ability to predict properties of an assembly from properties of the involved components.

CBSE Terminology

To make the things easier we need first some definitions...

- Software Component
- Component-based systems
- Component specification
- Component composition
- Component and systems properties

Summary CBSE – basic definitions

- The basis is the Component
- Components can be assembled according to the rules 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
- Component are performing in the context of a component framework
- All parts conform to the component model
- A component technology is a concrete implementation of a component model
Component models classifications

- **Lifecycle.** The lifecycle dimension identifies the support provided (explicitly or implicitly) by the component model, in certain points of a lifecycle of components or component-based systems.

- **Constructs.** The constructs dimension identifies (i) the component interface used for the interaction with other components and external environment, and (ii) the means of component binding and communication.

- **Extra-Functional Properties.** The extra-functional properties dimension identifies specifications and support that includes the provision of property values and means for their composition.

- **Domains.** This dimension shows in which application and business domains component models are used.

Classifications

- **Lifecycle**
  - Modeling
  - Implementation
  - Packaging
  - Deployment

- **Constructs**
  - Interface types
  - Interface specification language
  - Interface Level (signature, contract-based, semantics)
  - Interaction

- **EFP**
  - General support for properties
  - Properties specification
  - Composition support

- **Domain**
  - Specific
  - General-purpose
Some of component models

- AUTOSAR
- BIP
- CCM
- Fractal
- KODA
- EJB
- MS COM
- MS .NET
- OSGi
- PIN
- PECOS
- RUBUS
- SaveCCM
- SOFA 2.0
- ...

Component-based software development process

Time to Market – “Classical” Development Process?

Product Lifecycle

Problems:
- Time To Market
- High Costs
- Meeting deadlines
- Visibility

Requirements

Specification

Design

Implementation

Test

Operation & Maintenance

TIME
Development process
- COTS and outsourcing require different development processes

- Requirements
- Specification
- Design
- Implementation
- Test
- Find & Select
- Adapt
- Deploy

Development process – emphasize reuse
- Managing COTS in the early stage of the development process

- Requirements
- Specification
- Design
- Implementation
- Test
- Find & Select
- Adapt
- Test
- Deploy

CBD – separation of development processes
- Component development
- Application development

- Requirements
- Specification
- Design
- Implementation
- Test
- Find & Select
- Test
- Adapt
- Deploy
- Operation & Maintenance
Types of component-based development

- Internal components as structural entities
- Reusable components developed in-house
- COTS (commercial off the shelf) components

Software Architecture and Software Components

- Architecture Specification
- Structure specification
- Set of interface specification

Software Architecture and components
3. 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."

---

**Two Tier Architecture**

- Presentation
- Business Logic

---

**N-Tier Architecture**

- Presentation Layer
- Business Layer
- Database
N-Tier Architecture - and Components

Distributed Software Components

Aspects of Software Architecture

- Elements and Form
- (UniCon notation)
Different architecture view in different phases

**Phase I**
- System architecture - Decomposition of the system

**Phase 2**
- Implementation Architecture - Component Identification

**Phase 3**
- Deployment architecture

System Design – Phase 2

System Design – Phase 3
Example: Component-based embedded systems

The architectural design challenge

Artigos para Discussão

Artigos para Discussão
