Introduction to Component-Based Software Engineering

Barbora Zimmerová

Faculty of Informatics, Masaryk University Brno
xzimmer@fi.muni.cz

October 18, 2004
1 Motivation
   Components around us

2 Software components
   Features and benefits

3 CBD
   What is Component-Based Development
   Component application life-cycle

4 Component architectures
   COM/DCOM/COM+
   CORBA/CCM
   EJB
   Comparison

5 Testing & Verification
   Testing
   Verification

6 Conclusion
Components around us

Industry example

- nearly every industry product is made of mechanical components
- car is assembled from: engine, wheel, tires, seats, ...
- computer is assembled from: processor, hard drive, memory card, CD-ROM, ...
Adoption of industry processes (1)

Evolution:

1. Craft
2. Industrial production
3. Engineering

1. Craft
   • One-man projects
   • Talent and experience instead of design
Adoption of industry processes (2)

2. Industrial production
   - Different people playing different roles
   - Project management
   - Design before implementation
   - Documentation

3. Engineering
   - Importance of efficiency, reliability, flexibility, time-to-market
   - Quality management
   - Automatic testing
   - Standards of products and processes
   - Systematic development of product
   -> Component-Based Software Engineering
Reuse, reuse, reuse!

Levels of reusability in Object-Oriented Approach

- Object/Class
- Pattern
- Component
- ... and others

White box reuse vs. Black box reuse

Benefits of reuse

- Faster development
- Lower cost (old items are reused, new items will be reused)
- Enhanced quality
- Lower risk
- Easier maintenance
What is a software component?

A component is a contractually specified building block for software which can be readily deployed by third parties without understanding its internal structure.

[Synergised from Goos and Szyperski]
Component features

Main features

- Encapsulation
- Unique identification
- Reusability
- Using of multiple instances
- Ready to use
- Interface
- Client anonymity
- Recursive structure

Other features

- Language independent
- Platform independent
- Configurability
Component vs. Class

Component

- Executable run-time entity
- May contain several classes
- No code available
- Description by Interfaces
- Developed separately
- Deployment context changes after compilation

Class

- Design-time entity
- Code often required
- Often designed for one system
- Deployment context does not change after compilation
Component vs. Component instance

Component

- Abstract
- Several implementations later
- Multiple instances
- No, one or several classes

Component instance

- Concrete lump of code in memory
- Defined by one abstract component
- No, one or several objects
Why components?

What are its benefits

- Reusability and all its benefits
  - Faster development
  - Lower cost
  - Enhanced quality
  - Lower risk
  - Easier maintenance

- Configurability of the application
CBD (Component-Based Development) is a software development process based on Component-Based Approach.

Component dictionary:

- CBD - Component-Based Development
- CBSE - Component-Based Software Engineering
- CBAs - Component-Based Applications
- ... and many more CBx
Component life-cycle

- Creation of component
- Interface publication
- Dissemination
- Matching of the appropriate component
- Assembly of components
- Maintenance
Component application life-cycle

• Creation of component A
• Creation of component B
• Assembly of the application [ A B ]
• Purchasing of component B2
• Changing the component B for B2 [ A B2 ]

We use two types of programmers
• Component developers
• Component assemblers
Component changes

- Rule No.1: When you need to change the component, never change the interfaces.
- Rule No.2: If you need to change the interface, save the old one and add a new interface.

There are several possibilities to do so:

- Interface versioning
- Component containment
- Component aggregation
- Inheritance of interfaces
Why do we need component architectures?

The main idea of Component-Based approach is...

- components can be made by third parties
- we can just find the appropriate components, buy them and assembly into final application
- like we can assemble a computers according to our needs

In case of assembling mechanic components, standards are necessary! What about standards for software components?

What must be defined:

- Component: What is "the Component" and how does it look like
- Interfaces: How do the interfaces look like and how are they described
- Interaction: How do the components interact and communicate
COM/DCOM/COM+

- Supported by Microsoft

Dictionary:
- COM - Component Object Model
- DCOM - Distributed COM
- COM+ - DCOM with advanced features
- .NET framework

Specialties:
- Component = binary executable file (component = binary class / instance = binary object)
- Interfaces - at least 2 interfaces (IUnknown), described with MS IDL
- Interaction - Proxy and Stub Objects, Marshalling/Unmarshalling, RPC (Remote Procedure Call)
CORBA/CCM

- Supported by OMG (Object Management Group)

Dictionary:
- CORBA - Common Object Request Broker Architecture
- CCM - CORBA Component Model
- OMA - Object Management Architecture

Specialties:
- Component = CORBA Object
- Interfaces - one IDL interface, described with OMG IDL
- Interaction - ORB (Object Request Broker)
EJB

• Supported by Sun Microsystems

Dictionary:
• EJB - Enterprise JavaBeans
• J2EE - Java 2 Platform, Enterprise Edition

Specialties:
• Component = Bean consisting of Bean Class, Interfaces, Supplementary Classes, Deployment Descriptors
• Interfaces - Home Interface and Component’s Interface
• Interaction - EJB Container, RMI (Remote Method Invocation)
Suitability

- COM applications are suitable for applications running in Windows operating system.
- CORBA applications are suitable for large heterogeneous distributed systems with importance of reliability and robustness (telecommunication industry).
- EJB are suitable for easy development of enterprise applications. EJB application are popular in the area of B2B and B2C Internet portals.
Testing & Verification

Component testing problems:

- We don’t know, in which system the component will be used.
- We don’t know when and by whom the component instance will be created, used and destroyed.

Component application testing problems:

- We don’t have proper component specification and design models.
- Any component can be modified anytime.
- We don’t compile an application after assembly.

These features reduce the applicability of many traditional test and verification techniques.
We distinguish:

- Testing of the component (4 types) - based on the roles it is performing within the application
- Testing of the assembled solution
- Regressive application testing - after change of some component

Note: Component usually has no user interface for sending requests.
Behavior Capture and Test
Using runtime information to automatically derive test cases for component when used in different contexts.

Phases
- Deploy time: Automatic generation and installation of recorders
- Run-time: Recording of executions (single behaviors)
- Run-time: Distilling I/O and interaction invariants
- Run-time: Filtering single behaviors
- Regression: Automatic verification and testing

[Mariani, L.: Behavior Capture and Test for Verifying Evolving Component-Based Systems (ICSE04)]
We need to define:

- General component model - some techniques use UML, some define own models
- Technique for verification of single component - based on requirements that have impact on the neighbours
- Technique for verification of component composition
Verification - Example technique

**Verified Systems by Composition from Verified Components**

Define general component model and verification technique.

- As a software component is built, temporal properties of the component are specified, verified, and then packaged with the component.
- Selecting a component for reuse considers not only its functionality but also its temporal properties.
- Verification of properties of a composed application reuses verified properties of its components and is based on compositional reasoning.

[Xie, F., Browne, J.C.: Verified Systems by Composition from Verified Components (ESEC/FSE03)]
Verification - Example technique

Verified Systems by Composition from Verified Components

Definition of the General Component Model:

A component, $C$, is a four-tuple, $(E, I, V, P)$, where

- $E$ is an executable representation of $C$.
- $I$ is an interface through which $C$ interacts with other components.
- $V$ is a set of variables defined in $E$ and referenced by the properties defined in $P$.
- $P$ is a set of temporal properties that are defined on $I$ and $V$, and have been verified on $E$. A temporal property is denoted by a pair, $(p, A(p))$, where $p$ is a temporal formula defined on $I$ and $V$, and $A(p)$ is a set of temporal formulas defined on $I$ and $V$.
Now you know the main features of Component-Based Applications. How would YOU test and verify them?