SWK

JJ+HS

Introduction

Patterns

Components

Design by contract

Components and OO

Java Beans

OSGi

Component Spec. Proc.

Requirements Definition

Component Identification

Component Interaction

Component Specification

Provisioning and Assembly

References

Components

Component technology - introduction

- New trend in software technology
- Basic idea: build software system from smaller (already developed and tested) parts
- Re-build (compiling) of components usually not nessecary; we distingish between
  - White-box components (source code available), and
  - Black-box components (only binary available)
- Interface desciptions and component model/standard are important
- Current Technologies: (Enterprise) Java Beans, OSGi Service Platform, Component Object Model (COM), Corba Component Model (CCM)
- The component approach tries to apply standard engineering methods to software development

Definitions of “Component” I

- Generally, “component” only means “part of ...”
- Doug McIlroy coined the term “software-component” at the Garmisch conference in October 1968
- The term is overloaded, e.g., for software architectures

Definitions of "Component" II

Some definitions for (black-box) components:

- 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 parties. (Szyperski et al. (2002))
- A package of software that is independently developed and that defines interfaces for the services it provides and the services it requires. (D'Souza and Wills (1998))
- A software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard (Heineman and Councill (2001)).
- More definitions, see Szyperski et al. (2002), Chapter 11.

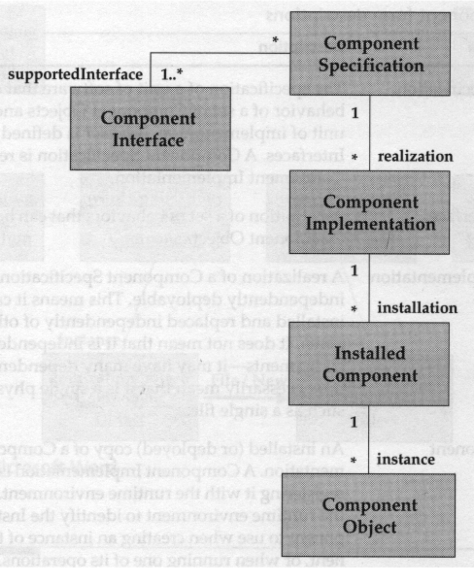
Component forms I

SWK
JJ+HS
Introduction
Patterns
Components
Design by contract
Components and OO
Java Beans
OSGi
Component Spec. Proc.
Requirements Definition
Component Identification
Component Interaction
Component Specification
Provisioning and Assembly
References

Features of components have in the different life-cycle states (Cheesman and Daniels (2001)):

- To use a component it must conform to the **Component Standard** in use, like Enterprise Java Beans (EJB) or Microsoft COM+.
- **Component Specification**: valid definition of the component.
- **Component Interface** or just **Interface** is a major part of the component specification.
- It should be possible to replace one **Component Implementation** with another with the same Component Specification.
- **Installed Component**: installed copy of the implementation.
- **Component Object**: instance of an Installed Component.

Component forms II



Design by Contract

What are preconditions and postconditions good for?

Contracts in daily life, Meyer (1997)

- Contractual partners are clients and sellers or service providers.
- Both expect advantages from the contract and are willing to make a commitment.

Example

SWK
JJ+HS
Introduction
Patterns
Components
Design by contract
Components and OO
Java Beans
OSGi
Component Spec. Proc.
Requirements Definition
Component Identification
Component Interaction
Component Specification
Provisioning and Assembly
References

I want to travel from Berlin to Duisburg.

	Commitments	**Advantages**
Passenger	Pay ticket Be there at departure time **must keep precondition**	getting to Duisburg **Has advantages from the postcondition**
Traffic provider	Must take the passenger to Duisburg **Must guarantee postcondition**	receives the price for the ticket; does not have to take passengers who have not paid or did not arrive in time **Can assume precondition**

Advantages of explicit contracts

Meyer:

A contract document protects both the client, by specifying how much should be done, and the supplier, by stating that the supplier is not liable for failing to carry out tasks outside of the specified scope.

Application to software

A contract is a formal agreement between a software / a class and its environment / clients. It specifies the rights and duties for both sides.

Contents of implementation contracts

SWK
JJ+HS
Introduction
Patterns
Components
Design by contract
Components and OO
Java Beans
OSGi
Component Spec. Proc.
Requirements Definition
Component Identification
Component Interaction
Component Specification
Provisioning and Assembly
References

Precise description of the functional properties of instances of a class at its **interface**:

- What does the class require from its clients?
- What does the class guarantee to its clients?
- What combinations of attribute values are permitted?

Contract

If the client fulfills the requirements of the server, then the server will provide the specified functionality.

- The client can rely on the assertions of the server. The internals of the server class are of no interest to the client.
- If the client does not fulfill the requirements of the server, then the server has no obligations whatsoever, it can behave arbitrarily (including breakdown).
- **It is not the server that has to test if the precondition holds, but the client!**

Example: Stack (generic class)

class	Stack[T]
attribute	nb_elements: integer
	max_size: integer
method	empty(): Boolean
	full(): Boolean
	push(x: T)
	pop()
	top(): T
end class	Stack[T]

Specification of the stack operations with preconditions and postconditions I

- empty()

  pre true
  post **noChange and**
  Result = true ⇔ nb_elements = 0

- full()

  pre true
  post **noChange and**
  Result = true ⇔ nb_elements = max_size

- push(x: T)

  pre **not** full
  post **not** empty **and**
  nb_elements = **old** nb_elements + 1 **and**
  top = x

Specification of the stack operations with preconditions and postconditions II

- pop()

  pre **not** empty
  post **not** full **and**
  nb_elements = **old** nb_elements - 1
  and "top element of the stack is deleted"

- top(): T

  pre **not** empty
  post **noChange and**
  Result = "top element of the stack"

Commitments and advantages

	Commitments	Advantages
Client	Call *push*(x) only if stack is not full **Must keep precondition**	Element x is put on stack, top() results in x, *nb_elements* increases by 1. **Has advantages from postcondition**
Server	Makes sure that x is placed on the stack **Must guarantee postcondition**	Unnecessary to handle the case if stack is full. **Can assume precondition**

Method specification – precondition

Methods form the operational interface between client and server.
Hence, a contract on the level of methods must describe the condition under which a client is allowed to call a method (precondition) and the effect the server guarantees in that case (postcondition).

Precondition: Predicate on the parameters of the method and the attributes of the class.

Requirement of the server to its clients – must hold when method is called.
Example: **not** full

Method specification – postcondition

The effect of a method describes the state that holds after the method has terminated and the values of the output parameters in terms of the input parameters and the state that holds when the method is called.

Postcondition: Relation between input parameters, attributes of the class **before** executing the method, and the attributes of the class **after** executing the method, and the output parameters.

Example: **not** empty **and** nb_elements = **old** nb_elements + 1 **and** top = x

Class invariant

SWK
JJ+HS
Introduction
Patterns
Components
Design by contract
Components and OO
Java Beans
OSGi
Component Spec. Proc.
Requirements Definition
Component Identification
Component Interaction
Component Specification
Provisioning and Assembly
References

Not all combinations of attribute values describe an admissible instance of a class.

class invariant: Property describing an integrity condition on the attributes of a class.

Example: $0 \leq$ nb_elements $\leq$ max_size **and** max_size ≥ 1

The class invariant is implicitly contained in the pre- and postconditions of all methods!

Contract – Relation between client and server

Commitment of the client

- Satisfy preconditions of creation routines (constructor)
- Satisfy preconditions of methods

Commitment of the server:

- Creation routines establish class invariant
- Methods keep class invariant
- Methods establish postconditions

Relation to abstract data types

- Classes correspond to implementations of abstract data types (ADTs).
- In an ADT specification of a stack, we would have the following axioms:
  pop(push(x,s)) = s
  top(push(x,s)) = x
- These axioms cannot be expressed in terms of pre- and postconditions of single methods, because they express relations between several different methods.
- However, a stack implementation should guarantee that the axioms are fulfilled.

Contracts and inheritance

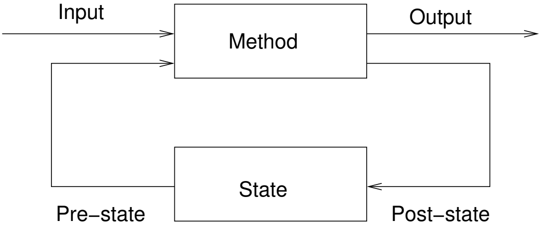
Introduction
Patterns
Components
Design by contract
Components and OO
Java Beans
OSGi
Component Spec. Proc.
Requirements Definition
Component Identification
Component Interaction
Component Specification
Provisioning and Assembly
References

If an inheritance hierarchy is part of an interface, i.e., clients can access servers polymorphically, then a subclass must keep all contracts of all superclasses.

- The class invariant must imply all the class invariants of the superclasses.
- Preconditions of re-defined methods must be implied by the preconditions of the super-methods.
- Postconditions of re-defined methods must imply the postconditions of the super-methods.

These conditions guarantee that a client does not experience any "surprises" when using a polymorphic server without knowing its exact dynamic type.

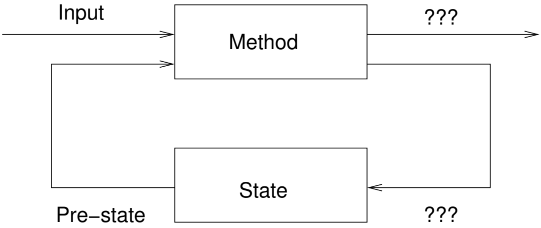
Design by Contract: Overview



Precondition describes input and pre-state

Postcondition describes relation between input/pre-state and output/post-state

Design by Contract: If precondition is not satisfied



If we call a method with the precondition not satisfied

- we do not know if there is any output and – if so – how it looks like
- we do not know if the method will terminate and – if so – how the post-state will look like

Advantages of Design by Contract

- Contracts make given restrictions explicit.
- Clear distribution of functionality at the interface between client and server.
- Avoiding unnecessary checks through overly defensive programming.
- Abstraction from the implementation of the server (replaceability).
- (Partial) checks at runtime by **assertions**.

What have we learned?

- The principle of design by contract makes explicit the obligations of users and providers of services.
- The caller of a method/a procedure (i.e., the client) must guarantee that the precondition is fulfilled; the server must in turn guarantee that the postcondition is fulfilled.
- Assertions should be added to the code and checked at runtime. Thus, errors are easier to find.