

EXAM

TDDC18 Component-based Software 2 juni 2005, 08:00–12:00, TER1

Solution proposal:

We give solution proposals for the assignments below. Note that, in some cases, there may be more than one possible correct answer; we show for brevity only one or a few of these. ◇

Examinator: Christoph Kessler

Jour: Christoph Kessler (070-3666687, 013-282406)

Hjälpmedel / Admitted material:

Engelsk ordbok / Dictionary from English to your native language

General instructions

- This exam has 11 assignments and 6 pages, including this one. Read all assignments carefully and completely before you begin.
- It is recommended that you use a new sheet for each assignment. Number all your sheets, and mark each sheet on top with your name, personnummer, and the course code (TDDC18).
- You may answer in either English or Swedish.
- Write clearly. Unreadable text will be ignored.
- Be precise in your statements. Unprecise formulations may lead to a reduction of points.
- Motivate clearly all statements and reasoning.
- Explain calculations and solution procedures.
- The assignments are *not* ordered according to difficulty.
- The exam is designed for 40 points. You may thus plan about 5 minutes per point.
- Grading: U, 3, 4, 5. The preliminary threshold for grade 3 is 20 points.
- **OBS Care:** Om du vill ha ditt betyg i det gamla betygssystemet (U, G, VG) skriv detta tydligt på omslaget av tentan. Annars kommer vi att använda det nya systemet (U, 3, 4, 5).

1. (6 p.) **Component models**

- (a) An early component system is Unix' *pipes and filters* (also known as shells and pipes or streams and filters). Describe the Unix pipes and filters model by pointing out its component model, composition technique and composition language. (3p)

Solution proposal:

Components are binary Unix programs (black box components). Their interfaces are the file I/O byte channels stdin (standard input), stdout (standard output) and stderr (standard error), which can be regarded as ports of a component. The composition technique consists of Unix shell utilities such as pipe that allow to statically bind and connect these ports and thereby redirect byte streams. For instance, the Unix pipe operator connects stdout of one component to stdin of another one, feeding output data produced by the former directly as input data consumed by the latter. Predefined programmable filter programs (such as grep, sed) can be used to adapt byte streams between two ports. There are several composition languages, such as the Unix shells, C, and scripting languages, which allow to specify such static connections. ◇

- (b) What is the main difference between whitebox, blackbox and graybox re-use of software components? Give an example component/composition framework or scenario for each of these. (3p)

Solution proposal:

Whitebox reuse adapts a component for a new context by editing its *source code*. Blackbox reuse does not change the component's code, which thus may even binary if the same platform is used. Instead, adaptation and glue code must be added separately. Only the component's interface is known. Graybox reuse allows to modify the component's code (e.g., by adding or exchanging adaptation code) only at certain, well-defined places, so-called hooks, which are the composition interface of the (graybox) component. Usually, recompilation is required also for (static) graybox composition. Example for whitebox reuse: editing the source code, e.g. by subclassing and overwriting some methods, refactoring, e.g. using Recoder. Example for blackbox reuse: re-wiring components in a software architecture framework, or COTS components, or web services. Example for graybox reuse: changing an aspect advice in Aspect-J, or instantiating a C++ template with a different (type) parameter, or (re)binding a hook in COMPOST. ◇

2. (3 p.) **Metaprogramming**

- (a) What is the difference between static and dynamic metaprogramming? (2p)

Solution proposal:

Static metaprograms are evaluated at compile time in the compiler; there is no runtime overhead (as the metaprogramming constructs will be resolved completely in the compiler frontend and not forwarded to code generation). In other words, static metaprograms run only at the metalevel, not at the base level (metalevel architecture). Hence, metalevel language and base-level language need not be identical.

Dynamic metaprograms (i.e., reflection) are evaluated at runtime, therefore the necessary metainformation must be made accessible via some language constructs or API (application programmer interface) of the same base-level programming language, and executing such metainformation queries causes run-time overhead. ◇

- (b) Give one concrete example system for static metaprogramming. (0.5p)

Solution proposal:

Compost (Recoder), or C++ templates, or Beta fragments. ◇

- (c) Give one concrete example system for dynamic metaprogramming. (0.5p)

Solution proposal:

Java Reflection. ◇

3. (4 p.) **CORBA**

- (a) Without middleware support, a program written in programming language *A* usually cannot directly call a method written in a different programming language *B* even if they would execute on the same computer. Give one of the technical reasons. (1p)

Solution proposal:

Differences in parameter passing (via stack/register) and ordering, byte order (little- or big-endian), datatype format (size and encoding), memory layout (array layout, memory word alignment). ◇

- (b) How does CORBA achieve programming language transparency for static calls? (Describe the principle and those main parts involved in CORBA static calls that are primarily relevant for enabling language transparency.) (3p)

Solution proposal:

From a description of the component's interface in *IDL*, a client-side stub and a server-side skeleton are *generated*. To make this work, a *bijective mapping* between IDL types and language-specific types must be available for each language involved. The *stub* consists of code that marshals (flattens, serializes) the client-language-specific data (e.g., call parameters) sent by the client into a language-neutral exchange format (e.g., a byte stream). The *skeleton* reads and unmarshals (parses, deserializes) these data into the corresponding data types in the server component's language, and delivers the call to the server method in the expected format.

(Other CORBA parts used with static calls, such as object adapters, IORs and IIOP, are only relevant for remote invocations and need therefore not be named here.) ◇

4. (3 p.) **JavaBeans**

- (a) What changes must be made to the class `PieChartDiagram` in order to turn it into a JavaBean? (2p)

```
public class PieChartDiagram {  
  
    public PieChartDiagram ( boolean defaultValue ) {  
        }  
  
}
```

Solution proposal:

Implement the interface `Serializable` and create a public parameterless constructor. Answer in code:

```
public class PieChartDiagram implements Serializable {  
    public PieChartDiagram () {}  
    ... ◇
```

- (b) If there was no reflection mechanism in Java, the JavaBeans component model would not work. Why? (1p)

Solution proposal:

The assembly tools use reflection to find out which classes are beans and which properties and events they have. ◇

5. (4 p.) **Enterprise JavaBeans (EJB)**

- (a) Why do Enterprise JavaBean developers have to specify if their session beans are stateful or stateless? (2p)

Solution proposal:

It makes it possible for the EJB container to reuse bean instances for stateless beans. It makes it possible for the EJB container to know what must be done when temporarily persisting bean instances (deactivation, activation). ◇

- (b) One could argue about whether Enterprise JavaBeans are object oriented or not. What are the arguments against? (2p)

Solution proposal:

Separation of data and operations (entity beans and session beans). No inheritance between beans. ◇

6. (3 p.) **COM**

- (a) How can a client compare the identity of COM component instances? (2p)

Solution proposal:

By comparing the pointers to the `IUnknown` interface of each instance. If the pointers are the same, it is the same instance. ◇

- (b) How is instance deletion handled in COM (name the technique, no details)? (1p)

Solution proposal:

By reference counting (garbage collection). ◇

7. (3 p.) **Aspect-Oriented Programming**

- (a) AspectJ is a powerful tool that aims to solve the problem of crosscutting concerns in Java. What is the greatest new problem that arises with using AspectJ aspects? (2p)

Solution proposal:

Code is difficult to understand, since its semantics are modified by aspects and the final interwoven code is not shown to the programmer. In AspectJ programs, names of classes/methods etc. can hold semantic meaning. (The unspecified weaving order of multiple aspects can also lead to problems.) ◇

- (b) What is the purpose of Join Points in aspect oriented programming? (1p)

Solution proposal:

The goal of a join point is to match a well defined execution point. ◇

8. (6 p.) **Software Architecture Systems**

- (a) What is the component model of software architecture systems? (in other words, what are the (major) different building blocks of a software architecture configuration, and what is their specific purpose?) (2p)

Solution proposal:

(i) Blackbox *components* with *ports* (declared interfaces) that can be aggregated to *services* (such as call). (ii) *Connector* components as bundles of logical wires called *roles* that connect specific ports (typed) of different components. [Components and connectors are composed to a *configuration* according to a composition plan specified in a software architecture description language.] Ports and connectors have protocols that are formally specified, e.g. by finite state machines. From the connectors, glue code is generated for a specific middleware infrastructure (e.g., messages or RPC). ◇

- (b) Define the term *blackboard (repository) architectural style* and give an example of a software system with this style. (2p)

Solution proposal:

In a blackboard or repository architectural style, all components are primarily connected to one central component that represents a common data repository. All data accesses (write, read) go via ports and connectors to the central repository component. An example is the CoSy compiler system's central repository storing the intermediate program representation, which is accessed by compiler components (the engines). All main data flow between CoSy engines is via the central repository. ◇

- (c) Software architecture systems are said to be a first step towards separation of concerns. Which concerns are separated, and what are the advantages of this? (2p)

Solution proposal:

Software architecture systems separate the *communication aspect* (topology and technical realization) from the main application functionality, which is packaged into the components. The advantage is that both components and interconnection can be varied separately: A component may be updated or substituted without having to touch its neighbors; a connector may be mapped to a different middleware; or the existing components may be re-wired. ◇

9. (2 p.) **Model-driven architecture**

Give a short description of Model-driven architecture (MDA) (main ideas). What is the main motivation for it?

Solution proposal:

Model-driven architecture consists in the step-wise refinement of software models, starting from high-level, platform-independent models (PIM) via platform-specific models (PSM) down to code. Usually, models are described in UML, using problem- and platform-specific profiles (metamodel extensions). Each refinement step is a partially automatic model transformation, using additional platform descriptions, but may also require manual complementation. The (most) platform-specific model is then (partially automatically) transformed into target code by code generation.

The main motivation is increased programmer productivity (less handwriting of code) and more reuse (e.g. the PIM survives technical changes in the platform). ◇

10. (2 p.) **Web services and parallel computing**

Assume you have got a few dozen Linux PCs with fast interconnect and should use them as a cluster to run a communication-intensive parallel application (e.g., parallel sorting of float-point numbers). Assume that high performance and effective exploitation of the available

hardware are very important. Do you consider web services as a suitable middleware for this scenario? If yes, why? If not, why not?

Solution proposal:

No, because web services have a complicated multilayer protocol and use lengthy XML encoding for messages, which costs both network bandwidth and data marshalling/demmarshalling overhead. This data bloat increases message latency on LAN connections by a factor of about 10 [Szyperki]. In parallel HPC systems, communication must be fast, while the platform, communication and component model transparency provided by web services is not needed there. A better choice would be e.g. MPI.^{1 2} ◇

11. (4 p.) **Invasive Software Composition**

- (a) What is a hook? Explain the difference between implicit and declared hooks. (2p)

Solution proposal:

Hooks are well-defined points of a component's code that may be subject to change. Invasive software composition changes components at hooks by binding new code to hooks. The hooks of a component form its composition interface.

Implicit hooks are defined by the component's programming language and mark certain points in a component's program structure, such as the list of class members or program control points for method entry and exit, etc.

Declared hooks are explicitly defined by the programmer and are given a name by which the composition system can locate them. [Unparsed, they are represented by comments or dummy constructs in the respective programming language.] ◇

- (b) What kind of operation does the method *x* in the following COMPOST program realize? (Give the technical term and a short description.) (2p)

```
public class ClassBox {
    // ...
    public ClassBox x ( ClassBox m, String subClassName ) {
        ClassBox s = this.copy ( subClassName ); // copy and rename a ClassBox
        return s.extend(m);
    }
    // where the extend composition operator on ClassBoxes is defined by
    public void extend ( ClassBox e ) {
        this.findHook ("members").extend ( e.findHook ("members") );
    }
}
```

Solution proposal:

Mixin-based inheritance.

The (super)class referred to by the **this** ClassBox object is copied into a new (sub)class *s* with new name *subClassName*, and the set of fields (members) of *s* is then extended with the new fields (members) taken from the mixin class *m*. ◇

¹Another delaying factor may be the interpretation of SOAP messages and WSDL interfaces. However, some recent implementations allow to compile stubs and skeletons from WSDL, which removes some of that additional runtime overhead.

²In spite of being slow, web services are becoming increasingly popular as middleware in Grid computing (e.g., Globus Toolkit v.4), which is explained by the fact that grid computing applications are usually *embarrassingly parallel*, that is, involve no interprocess(or) communication at all beyond dispatching parallel jobs and collecting their results.)