Dipartimento di Elettronica e Informazione
Politecnico di Milano
prof. Carlo Ghezzi e Elisabetta Di Nitto
20133 Milano (Italia)
Piazza Leonardo da Vinci, 32
Tel. (39) 02-2399.3400
Fax (39) 02-2399.3411

Software Engineering II
Part I

10 January 2008

Last name

First name

Identification number (Matricola)

Section (specify the professor you are associated with, either Prof. Ghezzi or Prof. Di Nitto)

Notes

1. Missing identification data invalidate the exam.
2. Return **only** these pages. Extra sheets will be ignored. You may use a pencil.
3. The exam is in 2 parts. For part 1 you will not be allowed to use books or class notes. For part 2 you will be allowed to use books and class notes. The first part must be in 30 min., the second part in 55 min. The final result is the sum of the scores obtained in both parts.
4. Any use of electronic devices (computers, calculators, cell phones, …) is forbidden.
5. You cannot keep a copy of the exam when you leave the room.

Question 1 – Software development process (2 points)

What is a software prototype?

It is an intermediate, partial implementation of a system developed to assess some property of the system before starting the final development, to reduce project risks.
This practice is common in other fields. For example, airplane wings are evaluated in a wind tunnel before moving to production.

How can you classify prototypes?

Throw-away prototypes are only used for assessment (like the airplane wing above). Evolutionary prototypes may be evolved progressively into the final product. Both are possible in software. For example, a component that simulates the external environment in a plant control software can be viewed as throw-away.

When and why can it be useful?

Typically, when requirements are not fully understood. For example, the look-and-feel of a user interface.

Question 2 – Requirement engineering (2.5 points)

What is a black-box test set of a module?

A test set designed by looking at the module's specification.

What is a glass-box test set of a module?

A test set designed by looking at the internal code, trying to cover its structural characteristics.

Are the two alternative or not? (Yes or No is not an answer. You must motivate it!)

They are complementary. BB tests what the module should do. WB tests how the program is done; it may ignore problems due to missing functionalities. BB may not detect errors due to specific input data that force execution to follow certain branches of the control flow, where for example a statement might generate overflow.

Question 3 – Design (2.5 points)

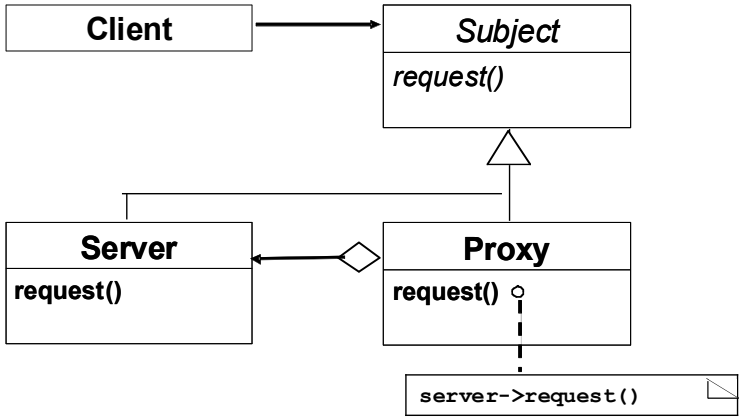
What is a tuple-space based (also called "LINDA-like") software architecture? Briefly describe its main features and its pros and cons.

A logically global persistent data space is used for coordination among components. Each can deposit a tuple and read/remove one via pattern matching (which depends on the pattern language of the middleware). Operations read and remove are blocking and nondeterministic. As opposed to publish/subscribe, here the data used for coordination are persistent.
Pro: high decoupling among components (which may for example be added dynamically)
Con: blocking coordination may be a problem. Publish/subscribe may have less overhead because no persistency, which may be not required by the application.

Question 4 – Testing (2 points)

Briefly describe in UML the **proxy** pattern and say in a few words what is its purpose and when it can be used.

It is a structural pattern. The goal is to allow multiple clients access the services offered by a server but not through direct access. By avoiding direct access, we achieve location transparency, guarantee security and efficiency. This is achieved by introducing an intermediary which may perform some pre- e post-processing (access control, caching, ...). Below is the UML diagram that describes it



Software Engineering II

10 January 2008

Last name

First name

Identification number (Matricola)

Section (specify the professor you are associated with, either Prof. Ghezzi or Prof. Di Nitto)

Part II

Question 1 Alloy (7 points)

Consider the following partial Alloy definition of a directed graph

```
sig Node {}
sig Edge {}
        source: Node,
        target: Node,
}
one sig Graph {
        nodes: set Node,
        edges: set Edge,
}
```

1. Add a constraint such that every element in Node belongs to a Graph.
2. Add a constraint on a Graph to specify the source and target nodes of edges are in nodes.
3. Define a particular Graph such that there is one privileged node that has no entering edges.
4. Define an assertion specifying that Graph has no cycles. Assume that the function reach exists that, given a node computes all nodes that can be reached by the first node:
   fun reach [n:Node]: set Node

Answers

1) This answer assumes that "one" is deleted from sig Graph, so that more than one Graph can exist.

```
fact {
        all n: Node | one g: Graph | n in g.nodes
}
```

Otherwise, under the assumption that there is only one Graph (as in the partial spec we provided), we can write:

```
fact {
        all n: Node | n in Graph.nodes
}
```

2)

```
fact {
        all e: Edge | e in Graph.edges implies e.source in Graph.nodes&& e.target in Graph.nodes
}
```

3)

```
sig RootedGraph extends Graph {
        root: Node
}
{ root in nodes && no e:Edge | e in edges and root in e.target}
```

4) Let us first provide function reach (even though you were not asked to provide it)

```
fun reach [n: Node]: set Node {
        n.^((~(Graph.edges<:source)).(Graph.edges<:target))
}
```

Assertio noCycle can be expressed as follows. There are of course counterexamples that the Alloy analyzer would generate.

```
assert noCycle {
        all n: Node | no n & reach[n]
}
```

Question 1 RE (6 points)

Consider the following requirements engineering problem for a pay-per-view system:

The goal G can be expressed as follows:

> "Only users who have paid for the service can access application SV (streaming video)"

We can assume the following valid domain properties D (it is a domain property because it is guaranteed by an existing access control system):

> "After submitting a payment, personal data and e-mail address, a user receives a password which authorizes access SV"

Assume the following requirement R for the "software-to-be":

> "Access to SV shall only be granted after a user types an authorized password"

1. What kind of reasoning should you do to prove the correctness of the requirements?
2. Can you formalize such reasoning?
3. What could possibly be wrong in this?

Answer

2) This is a possible formalization:

G: $\forall$ u$\in$User canAccess(u) $\Rightarrow$ hasPaid(u)
D: $\forall$ u$\in$User hasPaid(u) $\Rightarrow$ hasPwd(u)
R: $\forall$ u$\in$User canAccess(u) $\Rightarrow$ hasPwd(u)

1) One should prove that R **and** D $\Rightarrow$ G
This would be true if D were expressed as a $\Leftrightarrow$
3) In practice, nothing prevents that someone who did not pay shares the pwd with one who paid!