UBC CPSC 410
Solutions to Midterm
Nov 5th 2010

True/False, 4 points each, include a one or two sentence justification, for both true and false.

1.a. In the 4+1 views model for documenting architecture, each system is broken down into 4 separate layers and communication between layers is described by a scenario.

F: In the 4+1 view model, each view is a different way of looking at how the same system is organized or how execution takes place. Thus each view does not describe any particular layer/component of the system.

b. In an MVC architecture, the model is a visual diagram (e.g. UML) which illustrates the architecture.

F: In MVC, the Model describes the component that manages the data for the system.

c. A data-flow architecture uses a central repository to store information.

F: In the data-flow architecture, data is passed from one component (data-producer) to the next (data-consumer). Thus data is not typically stored in any central repository.

d. A Visitor pattern provides a functional decomposition of a feature in an object-oriented language.

T: all methods belonging to the same feature are grouped together in the same concrete visitor class.

e. In a layered architecture, the number of layers which is used could affect performance.

T: the system has to make an additional procedure-call for each new layer. Slide 16 in http://www.ugrad.cs.ubc.ca/~cs410/arch1.ppt

Short Answer (2 – 6 sentences)

2. (7 pts) What is the difference between the Logical View and the Development View in the 4+1 Views methodology? http://www.ugrad.cs.ubc.ca/~cs410/4plus1views.ppt

	Logical View	**Development View**
definition	Object design model	Static organization of software
For whom	For customers	For developers
Level of abstraction	Higher level	More towards physical level
Java	A package	A JAR-file
	Same package can spread over multiple JAR-files	
Decomposition	Functional/logical decomposition	Component decomposition

3. (7 pts) A company has built a data-flow architecture to process weather information retrieved from real-time sensors. A new input is provided to the system at regular time intervals, as individual readings of current temperature, given as a double type, representing the most recent temperature in Celsius. These inputs pass through a sequence of components which each compute different statistics about the readings recorded by the system over time. One of the components in the architecture computes a running average (called the Averaging Component) of all temperature readings recorded by the system, and it **prints the *new* average as *each input* reading passes through the component.**

If the Averaging Component is used in this system, can we call it a pipe-and-filter style or will the use of this component force the architecture to become a batch-sequential style? Why or why not?

Why Pipe-and-filter: it supports computation on each element of input, there is no buffering required to compute a running average.

- Let say your entire input has 10 numbers, the above spec is saying that the component would print out a new average every time an additional number is passed into it, rather than waiting until a larger set of inputs is available.

Why NOT Batch-sequential? Batch-sequential means you cannot start/resume computing an average until you receive a set with more than one input.

4. (7 pts) A three-tier system consists of a client browser connected to a web server, and the web server is connected to a database. Does the control-flow and data-flow of request processing in a three-tier system behave closer to that of a layered architecture or a pipe-and-filter architecture?

Explain why in two to four sentences. The answer must explain why it is similar to your choice and also not similar to the other.

Q10 in sample midterm:

Data-flow: more similar to Layered Abstract Machine.

- In 3-tier, both input data and output data occur at the Front-end / Top / Application layer.
- Data-flow in both 3-tier and LAM is bi-directional from one layer to the next and then back.
- Data-flow in pipe-and-filter is only 1-way: input at the producer-end and output at the consumer-end.

Control-flow: more similar to Layered Abstract Machine. In 3-tier:

- Front-layer can depend on multiple back-end servers, like lower-level layers in LAM
- Web-services / XMLHttpRequest are like remote procedure-calls.
- In pipe-and-filter, control flow is 1 filter at a time, almost no multiple dependencies.

AspectJ and Software Architecture

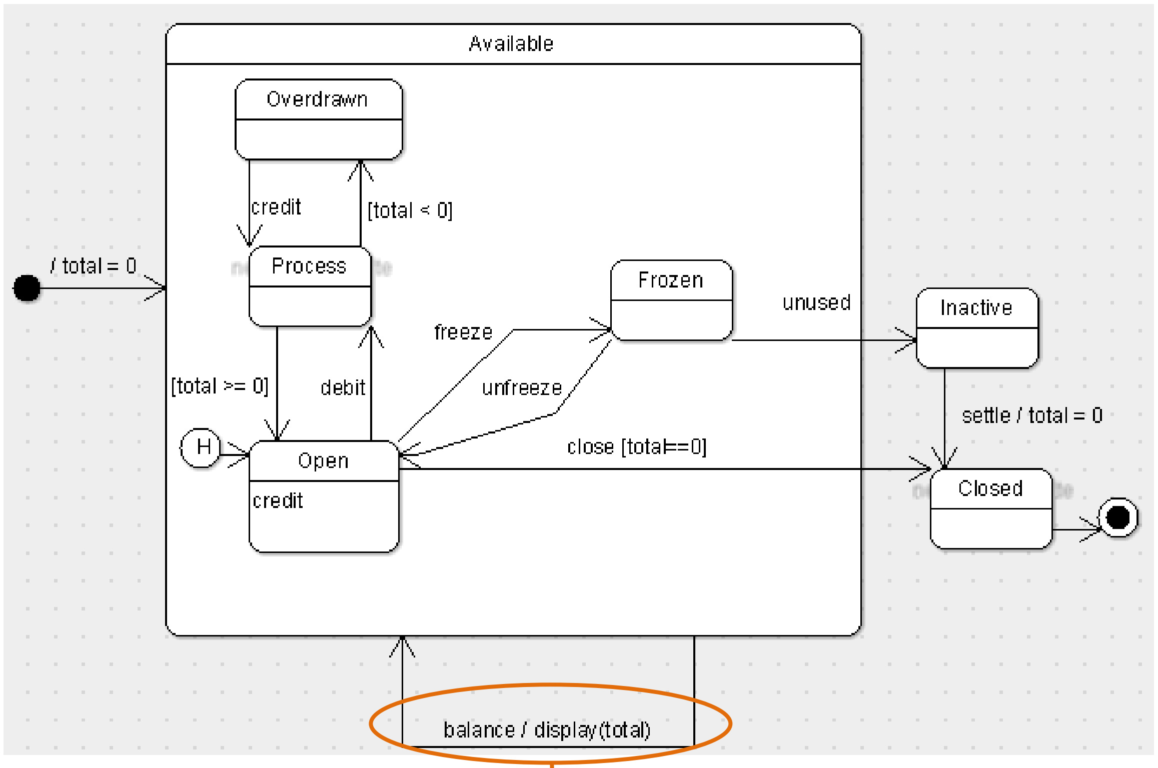
5. (20 pts) A software project uses a layered architecture (i.e. layered abstract machine). There are three layers and each layer is implemented by a corresponding Java package. The three layer packages are named: `ubc.ui`, `ubc.application`, and `ubc.network`. So, according to the rules for a layered architecture, classes in the `ubc.ui` package can make calls to classes in the `ubc.application` package, and classes in the `ubc.application` package can make calls to classes in the `ubc.network` package. Also, classes can certainly make calls to classes in the same package. All other calls between classes are forbidden (as is typical for a layered architecture).

Write an AspectJ pointcut which can detect when any method calls are made during program execution which violate these rules. For reference, several pointcut operators are described at the end of this exam. You will need to use some but perhaps not all of those operators.

```
public aspect Detector {

  pointcut ui(): within(ubc.ui.*) &&
                  call(* ubc.network.*.*(..));


  pointcut apps(): within(ubc.application.*) &&
                  call(* ubc.ui.*.*(..));

  pointcut ntwk(): within(ubc.network.*) &&
                  (call(* ubc.ui.*.*(..)) ||
                   call(* ubc.application.*.*(..)) );

  before(): ui() || apps() || ntwk() {
    System.out.print("call forbidden: layered-architecture
    violated");
  }
}
```

6. Statechart Logic (7 pts each)

Consider the Account statechart as discussed in class, and answer the following:



Hint: For (2) and (3), first solve for all possibilities of *t*, then find all s, which satisfy the statement for any of your answers for *t*.

1. For which states, s, is it true on at least one state sequence (i.e. path):

next(s, s) : All states inside *Available*:

Overdrawn/Process/Open/Frozen

2. For which states, s, is it true on at least one state sequence (i.e. path). ***Open***

There exists a state, t, such that: until(s, t, Inactive) && globally(t, total == 0)

t s.t. `globally(t, total == 0)` : Overdrawn, Process, Open, Frozen, Inactive

`s == Inactive, Frozen`

3. For which states, s, is it true on at least one state sequence (i.e. path):

There exists a state, t, such that: next(s, total < 0) && next(s, t) && next(t, total > 0)

a. next(s, total < 0) && next(s, t)

`t` must have `total < 0`
`t` might be `Process(total < 0) or Overdrawn(total < 0)`

b. `Process(total < 0) -> x(total > 0)`

no possible sequence

c. `Overdrawn(total < 0) -> x(total > 0)`

`x could be Process(total > 0)`

d. `next(s, t)`

`s -> Overdrawn(total < 0) -> Process(total > 0)`

So, `s` can be `Process` or `Overdrawn`

7. Visitor Pattern

A program uses a Visitor pattern to encapsulate the functionality for searching a tree data-structure. The Search feature takes an object as an argument and finds all of the nodes in the tree which reference the same object value from their `data` field. Answer the following two questions by considering the Visitor pattern code on the next page. For reference, an illustration of depth-first and breadth-first is provided at the end of the exam, if you need a reminder.

a. Does the `accept` method in the class `InnerNode` cause Visitors to visit the Tree nodes in **depth-first order** or breadth-first order?

```
class InnerNode extends Node {
      List<Node> children = new LinkedList<Node>();
      void accept(TreeVisitor tv) {

            // visit depth-first, before doing breath/for-loop
            tv.visit(this);  // 1.

            for(Node node : children) { // 2.
                  node.accept(tv);
            }
      }
}
```

b. Provide a new implementation for the Search functionality which uses a traditional object-oriented decomposition for including the Search feature as part of the object structure, instead of encapsulating the functionality into a Visitor. To simplify your answer, you only need to show the new implementation of the InnerNode class.

```
class InnerNode {

  List<Node> children = new LinkedList<Node>();
  Object data; //Or you can extend Node

  void search(Object find) {
    if (data.equals(find)) { // 1.
      System.out.println("Found innerNode");
    }
    for (Node node: children) { // 2.
      node.search(find);
    }
  }
}
```