LABO “Software Specification Techniques”	2 LIC INFO
Professor Serge Demeyer	Last modified: 26 October 2001

UML as a Software Specification Technique

Understanding

1. Open the application “Rhapsody Modeler in C++” in demo release. (Thus without a license which implies you can’t save your models.)
   - On the request to register; press [Next]
   - On the request to specify license file/server, press [Cancel]
   - Confirm that you want to work in demo version without save
2. Open the document “Dishwasher.rpy”
   - Directory [$Rhapsody_Home\Samples\CppSamples\Dishwasher\]
3. Browse the two specified object models [Abstract Dishwasher] and [Acme Dishwasher with Factory].
   » *What do you think these object models are specifying ? Why do you think so ?*
4. Browse the one specified sequence diagram [Dishwasher Cycle].
   » *What do you think this diagram is specifying ? Why do you think so ?*
5. Browse the one specified use case diagram [Dishwasher].
   » *What do you think this diagram is specifying ? Why do you think so ?*
6. Read the description of the use case [Wash Dishes].
   - Right-click the oval entitled [Wash Dishes]
   - choose “features …” from the pop-up menu
   - Read the description and interpret the diagrams you’ve seen before
     » *Which one of the above specifications helped you the most in understanding what this software system was supposed to do ?*
     » *What did your neighbours answer to this question? Discuss your answers.*

Design

7. Inspect the public interface of class [Heater]. Compare with the use case [Wash Dishes] and the sequence diagram [Dishwasher Cycle].
   - To inspect a class: click Packages > Default > Classes > Heater > Operations
     » *After creating a new heater, should it be in the on or off state? Why do you think so? How can you be certain?*
8. Inspect the public interface of class [Jet]. Compare with the use case [Wash Dishes] and the sequence diagram [Dishwasher Cycle].
   » *Is it allowed to send the message [evJetPulse] while the jet is spraying ? Why do you think so? How can you be certain?*
9. Inspect the public interface of class [DishWasher]. Compare with the use case [Wash Dishes] and the sequence diagram [Dishwasher Cycle].
   » *What happens if the user switches the washing mode (i.e. intensive / normal / quick) while the machine is operating? Why do you think so? How can you be certain?*
10. Open the respective statecharts for the classes [Heater], [Jet] and [DishWasher].
    - right-click the class
    - choose “Open Statechart” from the pop-up menu
      » *Reconsider the above questions; do you need to adapt your answer?*
      » *Did your neighbours have to adapt his answer? Discuss.*

Coding

11. Inspect the specification for the methods of class [Heater], [Jet] and [Tank].
    - To inspect a specification of a method for class: select the method and the specification will appear in the right pane of the window.
      » *Does the specification for these methods include source-code?*
12. Inspect the specification for the methods of class [Dishwasher].
    - Note the icons besides the methods: they illustrate new kinds of methods.
      » *Does the specification for these methods include source-code?*
13. Inspect the specification for the attributes of class [Dishwasher] and [AbstractFactory].
      » *Does the specification for these methods include source-code?*
14. Inspect the specification for the states and transitions in the statechart [Dishwasher].
    - To inspect a state, right-click the state and choose "features …".
      » *Does the specification for these states and transitions include source-code?*
15. Generate code for this application.
    - Open Components > EXE; verify whether all elements are checked for code generation
    - Open EXE > Configurations; select the "Host" Configuration
    - Right-click > choose "Set as Active Configuration"
    - From the menu select Code > Generate > Host
    - Confirm creation of a new directory
    - Inspect the code generated for these models
      » *Do you think this code is complete?*
16. Compare the code with the specification.
    - Count the number of event-methods and compare with the number of other methods.
    - Count the lines of code specified in normal method + states, and compare with the total code size.
    - Assess the quality of the code that is generated.
      » *What do you conclude from this comparison? Is it worthwhile to spend time in specifying so that you can save time while coding? Discuss with your neighbour.*

Verification

17. Verify whether the specification for the class [Jet] is complete, consistent and deterministic. If not, make appropriate modifications.
    - [complete] Every event/state pair has at least one transition leading to another state. This may imply creating extra states representing an error.
      To verify completeness, its best to draw a table listing vertically the events (+ guards) and horizontally the states. Each cell lists the corresponding actions.
    - [consistent] All state diagrams have an explicit initial and final state; every state is reachable from the initial state; the final state is reachable from all other states.
      To verify consistency, its best to draw a breadth-first spanning tree of the state-chart and see whether all leaf nodes are the final state. In case of cycles, pass via the cycle only once.
    - [unambiguous] The same event (incl. Guards) does not appear on more than one transition leaving any given state. (deterministic).
      To verify, use the same table as for verifying completeness.
      » *Do you feel such verification improves your confidence in the specification? Discuss with your neighbour.*

Contracts

18. Define pre- and post-conditions for the methods defined on the class [Jet].
    - Define a predicate for each state, which answers whether an object is in this state or not.
    - For each event-method, define a pre-condition based on whether the transition is acceptable or not (i.e. it leads to an error state).
    - For each event-method, define a post-condition to declare the arrival state of the method.

    » *Do you feel state-charts help you in finding the contracts for your methods? Discuss with your neighbour.*

Testing

19. Create test cases that cover the whole state-chart for the class [Jet].
    - Define a predicate for each state, which answers whether an object is in this state or not.
    - Build test-cases that cover the spanning tree used to verify the consistency. Every step in the test case exercises a single transition and verifies whether it arrives in the right state.

    » *Do you feel these test cases cover all what is necessary? Too much? Too Little? Discuss with your neighbour.*