Review Questions and Problems Chapter 4

1. Give examples for some *pseudo states* in UML state machines.

2. Give an example for a *UML transition*, illustrating all possible elements (event, etc.). Explain the function of the various elements of a transition.

3. What is the difference between a *UML composite state* and a *UML submachine state*?

4. There are various ways to exit a *UML composite state with orthogonal regions*? Give an example with at least three different ways to exit such a state.

5. What is a *UML junction vertex* good for?

6. To describe events in UML diagrams, the keyword *"when"* can be used in two semantically different ways. Give an example for each meaning.

7. Develop a *UML state machine* for the following requirements:

   You shall handle a bouncing button, i.e., a button that oscillates between the "closed" and "open" position for at most 50 msec after it has been pressed or released.
   You shall send the following signals upon detection of the respective condition:
   "buttonClosed", "buttonPressed2Seconds", the last signal indicating that the button has been pressed for 2 seconds (or more).

8. Develop a *UML state machine* for a simple stop watch. The stop watch shall have a single start/stop button, and a leap button where you can take intermediate times. The regular time is displayed in the first row of the display, the leap time in the second. A third button shall reset the regular time and leap time if pressed for more than 2 seconds. Take into account that the buttons exhibit bouncing.

9. What is the difference between an *action* and an *activity*?

10. Explain the different *types of nodes* in a UML activity diagram.

11. Draw a UML *activity diagram* for the following situation: There are two actions ActionA and action ActionB which can be executed in parallel. There is another action ActionC, which shall only be executed if ActionA and ActionB have been executed. The flow ends after execution of ActionC.

12. Draw an UML *activity diagram* for the following situation: CAN messages are received by a "ReceiveCAN" action. The messages are stored in a data buffer. The message is retrieved from the data buffer by a "ReceiveMsg" action. The "ReceiveMsg" action will wait for a message to arrive in the data buffer, and will retrieve it from there. After this action has retrieved a single message, two actions are executed *in parallel* on this message: a "CheckMsg" action, and a "ProcessMsg" action. When the output of both actions is "ok", a final action "FinalAction" is executed and then another message can be retrieved from the data buffer by the "ReceiveMsg" action. If there is an error anywhere in "CheckMsg", the error handler "ErrorAction" shall be executed, and thereafter the next message shall be processed.

13. Develop a *UML activity diagram* for a simple stop watch. The stop watch shall have a single start/stop button, and a leap button where you can take intermediate times. The regular time is displayed in the first row of the display, the leap time in the second. A third button shall reset the regular time and leap time if pressed for more than 2 seconds. Take into account that the buttons exhibit bouncing.