13

Application Analysis

13.1 Here are scenarios for the variations and exceptions. The precise student answers will vary a bit.

- **The ATM can't read the card**.
  The ATM asks the user to insert a card.
  The user inserts a cash card.
  The ATM accepts the card and cannot read it.
  The ATM displays an error message.
  The ATM keeps the card.
  The ATM asks the user to insert a card.
- **The card has expired**.
  The ATM asks the user to insert a card.
  The user inserts a cash card.
  The ATM accepts the card and reads its expiration date.
  The ATM notes that the expiration date is before today and that the card has expired.
  The ATM displays an error message.
  The ATM keeps the card.
  The ATM asks the user to insert a card.
- **The ATM times out waiting for a response**.
  The ATM asks the user to insert a card.
  The user inserts a cash card.
  The ATM accepts the card and reads its serial number.
  The ATM requests the password.
  The user does not respond and the ATM requests the password again.
  The user does not respond and the ATM displays an error message.
  The ATM keeps the card.
  The ATM asks the user to insert a card.

- **The amount is invalid**.
  The ATM displays a menu of accounts and commands.
  The user selects an account withdrawal.
  The ATM asks for the amount of cash.
  The user enters $0.
  The ATM complains that this is an illegal amount.
  The user enters $100.
  The ATM verifies that the withdrawal satisfies its policy limits.
  The ATM contacts the consortium and bank and verifies that the account has sufficient funds.
  The ATM dispenses the cash and asks the user to take it.
  The user takes the cash.
  The ATM displays a menu of possible commands.
- **The machine is out of cash or paper**.
  The ATM asks the user to insert a card.
  The user inserts a cash card.
  The ATM accepts the card and reads its serial number.
  The ATM requests the password.
  The user enters “1234.”
  The ATM verifies the password by contacting the consortium and bank.
  The ATM displays a warning that it cannot process withdrawals and is out of cash.
  The user chooses the command to terminate the session.
  The ATM ejects the card and asks the user to take it.
  The user takes the card.
  The ATM asks the user to insert a card
- **The communication lines are down**.
  The ATM asks the user to insert a card.
  The user inserts a cash card.
  The ATM accepts the card and reads its serial number.
  The ATM requests the password.
  The user enters “1234.”
  The ATM displays the warning that the communication lines have gone down.
  The ATM ejects the card and asks the user to take it.
  The user takes the card.
  The ATM displays the warning that the communication lines are down.
- **The transaction is rejected because of suspicious patterns of card usage**.
  The ATM asks the user to insert a card.
  The user inserts a cash card.
  The ATM accepts the card and reads its serial number.
  The ATM displays the message that it is keeping the card and that the user should contact the issuing bank.
  The ATM asks the user to insert a card

1. The only exception is that objects may not have the same value of the object ID attribute. The object ID is

13.2 Figure A13.1, Figure A13.2, and Figure A13.3 show the state diagrams for *Deposit*, *Withdrawal*, and *Query* respectively.

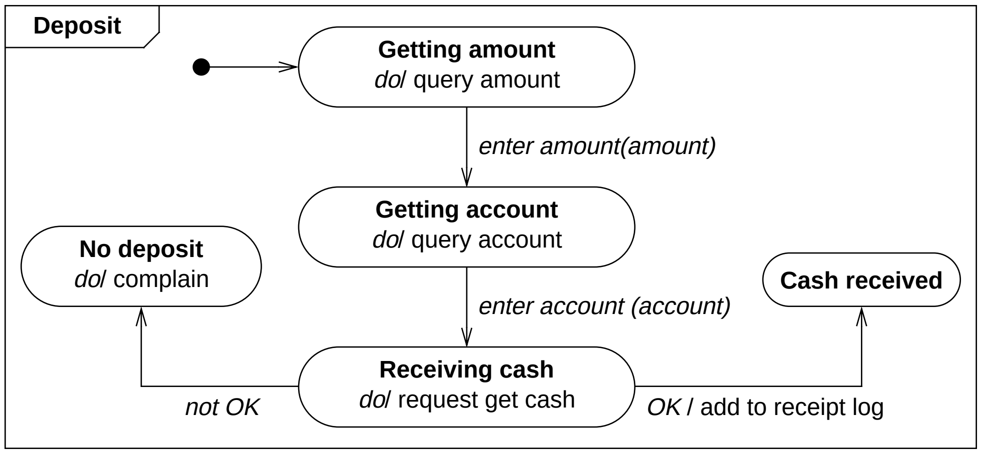


Figure A13.1 State diagram for *Deposit*

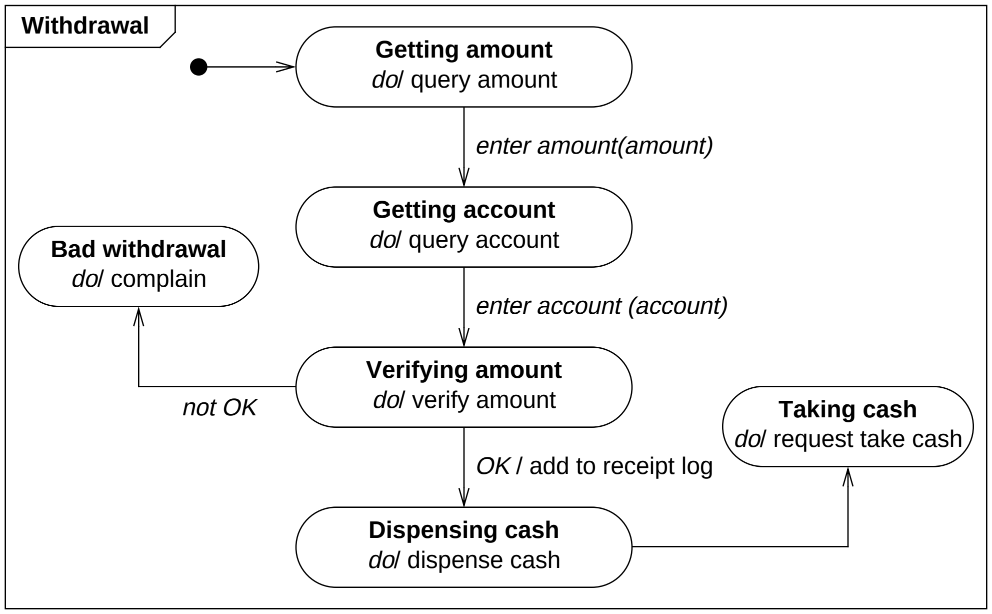


Figure A13.2 State diagram for *Withdrawal*

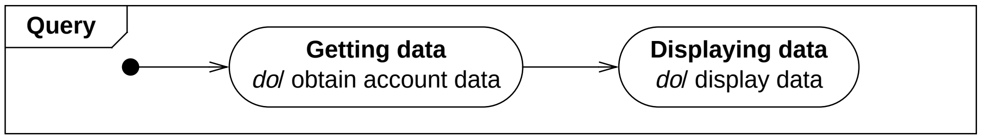


Figure A13.3 State diagram for *Query*

13.3 Figure A13.4 shows the sequence diagram for the given scenario.

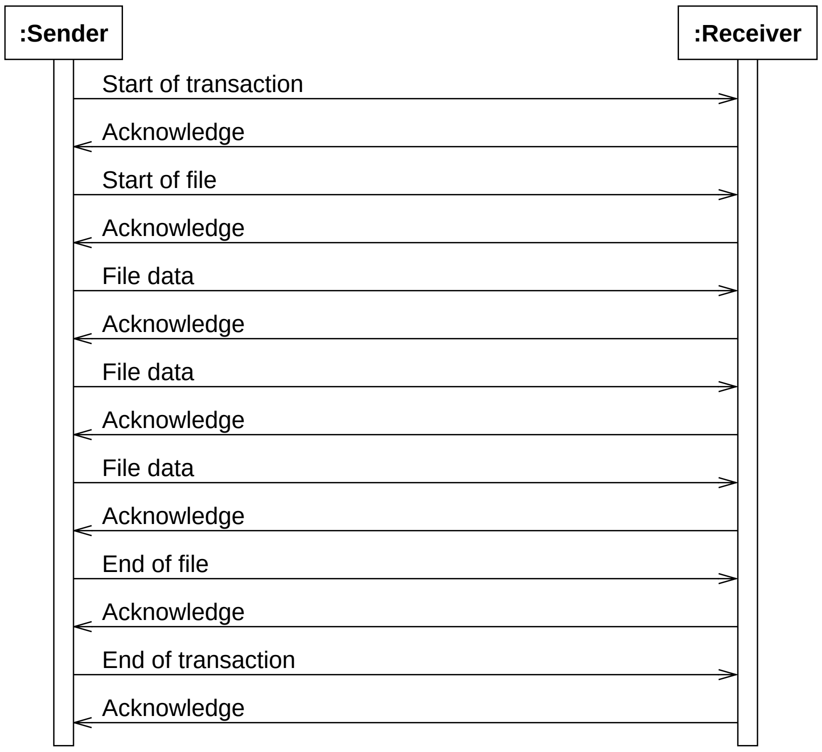


Figure A13.4 Sequence diagram for a data transmission protocol

13.4 Figure A13.5 shows a sequence diagram for a scenario in which several packets are garbled. Note that the exercise did not ask for errors caused by corruption of receiver packets. In an actual protocol, this would have to be taken into account.

13.5 Both the sender and receiver have state diagrams. We assume that a separate process supplies the sender with files to be sent and that another process stores the data on the

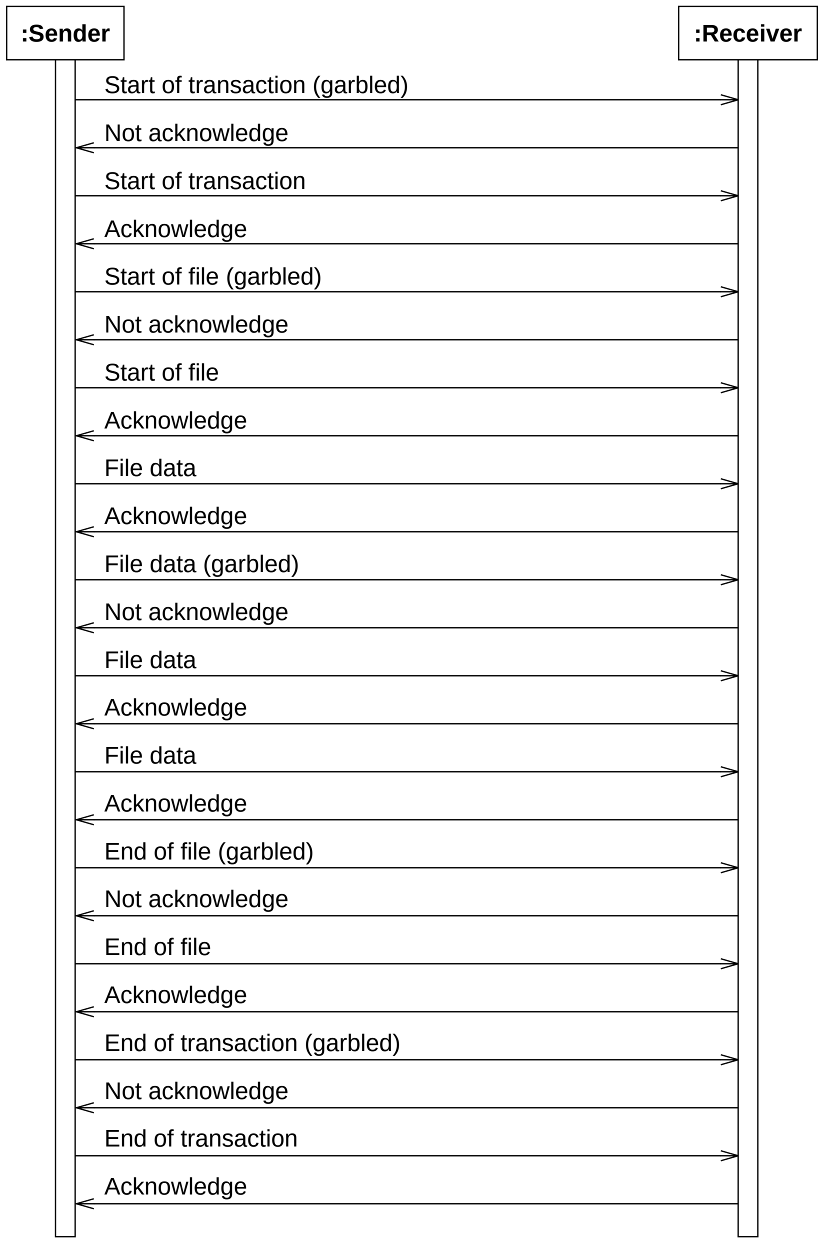


Figure A13.5 Sequence diagram for a data transmission protocol with errors

receiver end. Figure A13.6 shows a simplified state diagram for the sender. Figure A13.7 shows the state diagram for the receiver. The abbreviations *ack* and *nack* indicate acknowledged and not acknowledged. The diagram does not show the states needed to control the flow of data to processes that supply the sender and service the receiver. Note that neither of the state diagrams deals with complete system failure. In practice, time outs and retry counts would probably be used to make the system more robust.

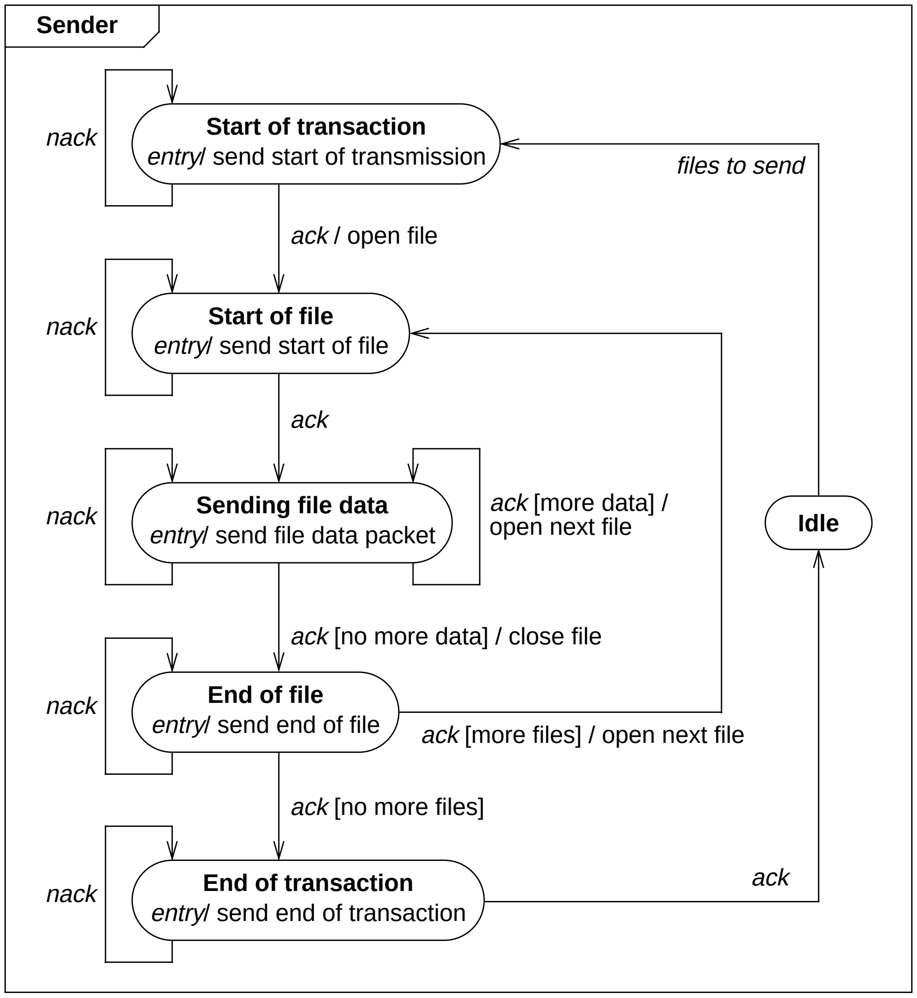

Sender
nack
Start of transaction
entry/ send start of transmission
files to send
ack / open file
nack
Start of file
entry/ send start of file
ack
nack
Sending file data
entry/ send file data packet
ack [more data] / open next file
Idle
ack [no more data] / close file
nack
End of file
entry/ send end of file
ack [more files] / open next file
ack [no more files]
nack
End of transaction
entry/ send end of transaction
ack

Figure A13.6 State diagram for the sender

13.6 Figure A13.8 shows a state diagram consistent with the scenarios.

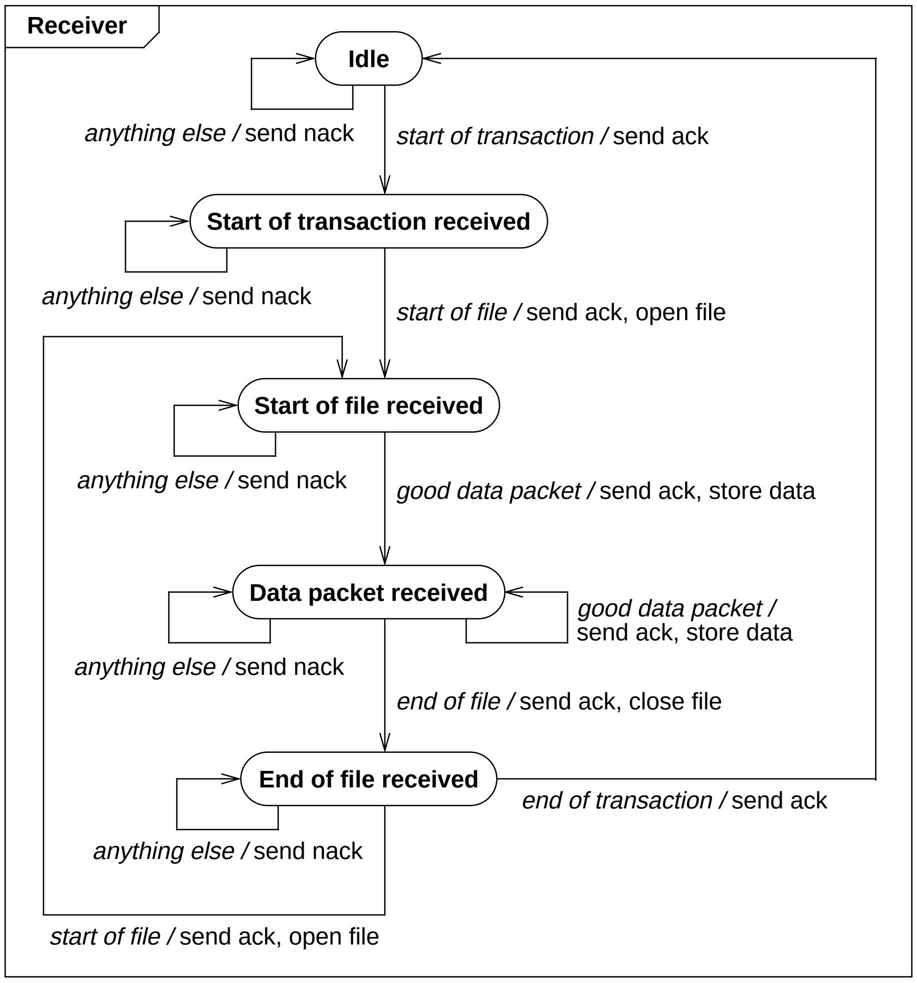

Receiver
Idle
anything else / send nack
start of transaction / send ack
Start of transaction received
anything else / send nack
start of file / send ack, open file
Start of file received
anything else / send nack
good data packet / send ack, store data
Data packet received
good data packet / send ack, store data
anything else / send nack
end of file / send ack, close file
End of file received
end of transaction / send ack
anything else / send nack
start of file / send ack, open file

Figure A13.7 State diagram for the receiver

13.7 The application is a simple editor that supports only boxes, links, and text. Text is allowed only in boxes and the text size and font is fixed. Boxes automatically adjust to fit the enclosed text. Links consist solely of horizontal and vertical lines.

13.8 Actors are the user and the file system.

13.9 [Instructor’s note: you should give the students the actors from the previous exercise.]
Use cases represent a kind of service that the system provides and should be at the same level of detail. Consequently the use cases should not reflect detailed operations,

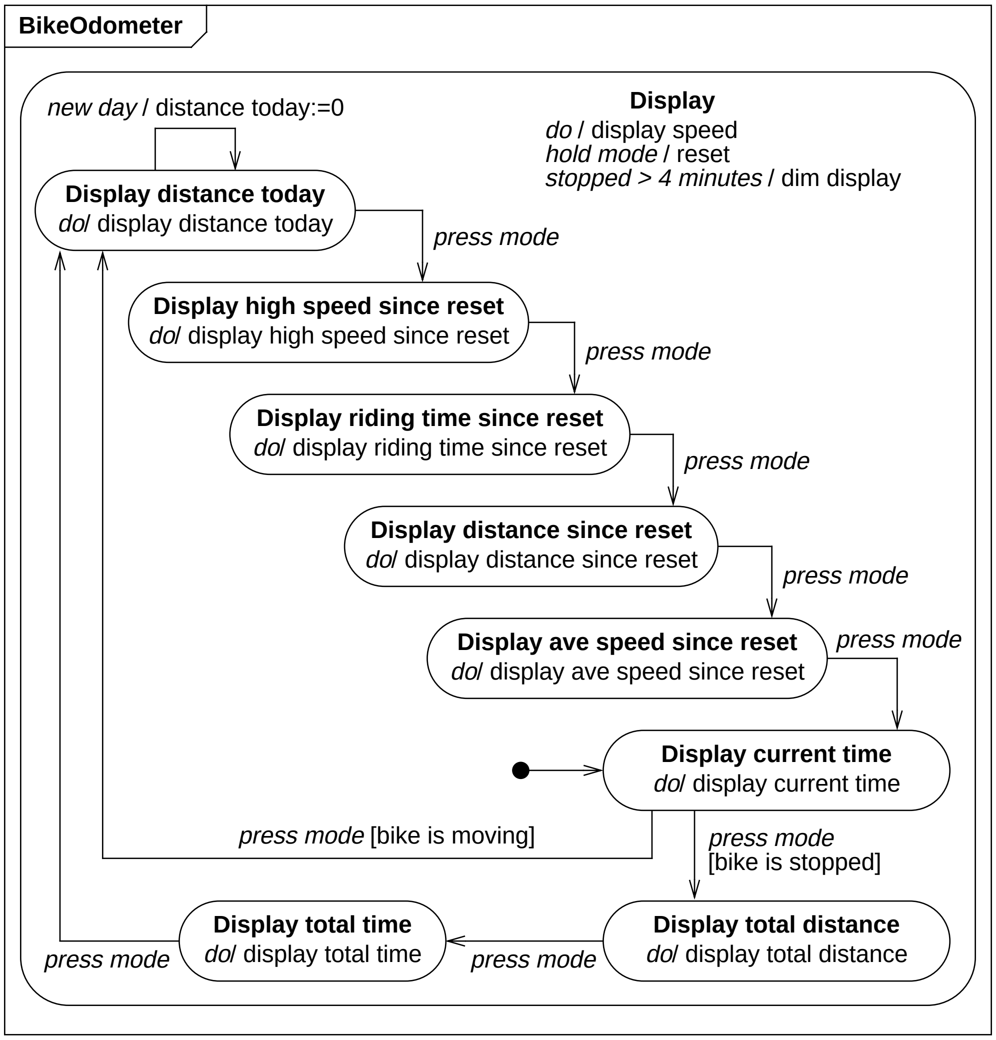

BikeOdometer
Display
do / display speed
hold mode / reset
stopped > 4 minutes / dim display
new day / distance today:=0
Display distance today
do/ display distance today
press mode
Display high speed since reset
do/ display high speed since reset
press mode
Display riding time since reset
do/ display riding time since reset
press mode
Display distance since reset
do/ display distance since reset
press mode
Display ave speed since reset
do/ display ave speed since reset
press mode
Display current time
do/ display current time
press mode [bike is moving]
press mode [bike is stopped]
Display total distance
do/ display total distance
press mode
Display total time
do/ display total time
press mode

Figure A13.8 State diagram for a bike odometer

but instead should focus on high level tasks that the user can perform. Here are the use cases. Figure A13.9 shows a use case diagram.

- **Create drawing**. Start a new, empty drawing in memory and overwrite any prior contents. Have the user confirm, if there is a prior drawing that has not been saved.
- **Modify drawing**. Change the contents of the drawing that is loaded into memory.
- **Save to file**. Save the drawing in memory to a file.

- **Load from file**. Read a file and load a drawing into memory overwriting any prior contents. Have the user confirm, if there is a prior drawing that has not been saved.
- **Quit drawing**. Abort the changes to a drawing and clear the contents of memory.

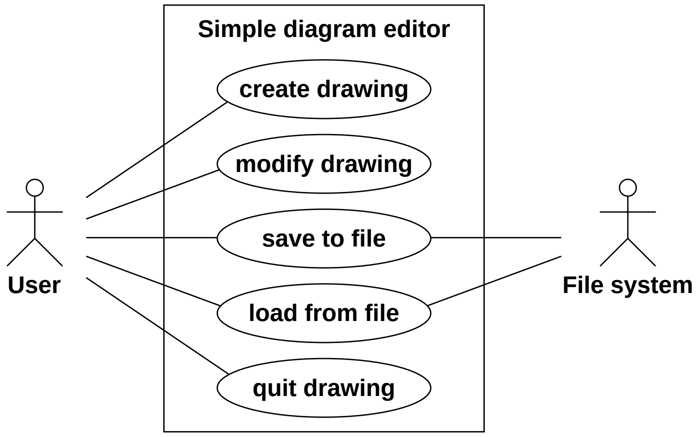


Figure A13.9 Use case diagram for the simple diagram editor

13.10 Figure A13.10 organizes the use cases. The overwrite confirmation is another use case that is included in creating a drawing and loading from a file.

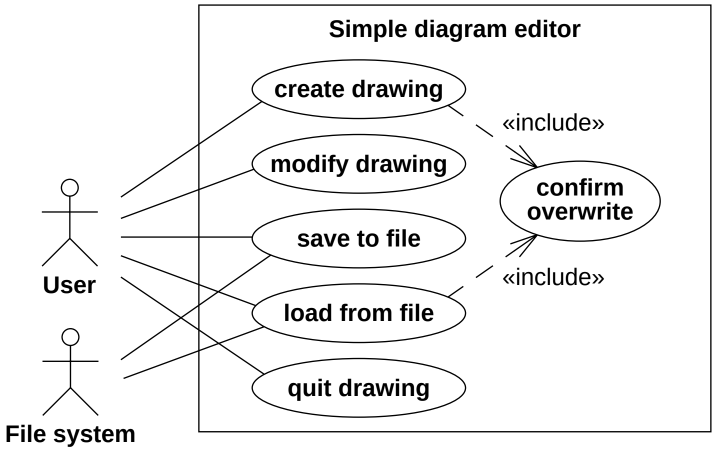


Figure A13.10 Use case diagram with relationships for the simple diagram editor

13.11 There are an infinite number of correct answers to this exercise, one of which is listed below.

- User loads an existing drawing. Editor retrieves the document and sets the cursor to the last referenced sheet.
  User goes to the first sheet. Editor moves the cursor.
  User goes to the next sheet. Editor moves the cursor.
  User deletes this sheet. Editor requests confirmation. User confirms.
  User goes to the last sheet. Editor moves the cursor.
  User goes to the previous sheet. Editor moves the cursor.
  User deletes all existing sheets. Editor requests confirmation. User confirms.
  User creates a new sheet. Editor sets the cursor to this sheet.
  User creates a box. Editor highlights newly created box. User enters text “x”.
  User copies x-box. Editor highlights new copy of box. User moves selected box.
  User selects text in box. Editor highlights text and unhighlights box. User cuts text. User selects empty box. Editor highlights empty box. User enters text “y”.
  User copies y-box. Editor highlights new copy of box. User moves selected box. User edits the “y” and changes it to “+”.
  User selects y-box. Editor highlights y-box. User copies y-box. Editor highlights new copy of box. User moves selected box. User edits “y” and changes it to “x+y”.
  User cuts x+y-box. User changes his/her mind and pastes the x+y-box back in.
  User selects the x-box. Editor highlights x-box. User also selects the +-box. Editor also highlights +-box. User links the boxes.
  User selects the y-box. Editor highlights y-box. User also selects the +-box. Editor also highlights +-box. User links the boxes.
  User selects the +-box. Editor highlights +-box. User also selects the x+y-box. Editor also highlights x+y-box. User links the boxes.
  User selects all boxes and links. Editor highlights all boxes and links. User groups the selections. User aligns the grouped selection with regard to the left-right center of the page.
  User renames the drawing file and saves the drawing.

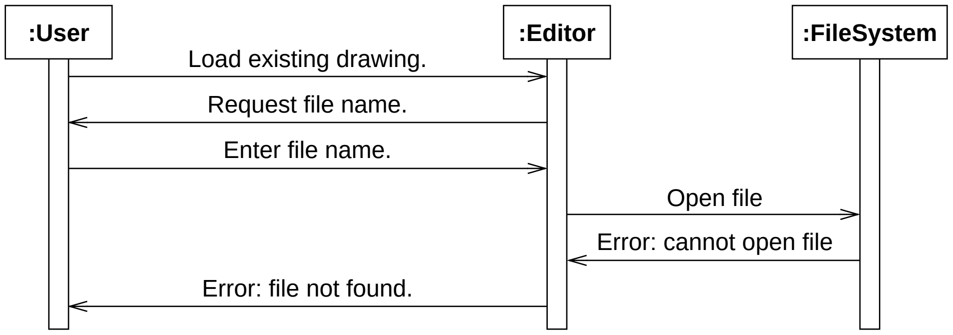
13.12 Error scenario 1:
  User enters command: *load existing drawing file* and supplies a file name.
  Command fails: file not found.

Error scenario 2:
  User selects the x-box. Editor highlights x-box.
  User also selects the y-box. Editor also highlights y-box.
  User also selects the +-box. Editor also highlights +-box.
  User tries to select the *link box* command.
  Command fails: must pick exactly two boxes for linking.

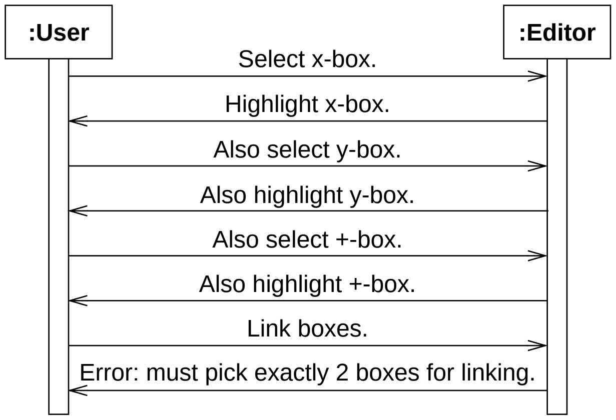
Error scenario 3:
  User selects the x-box. Editor highlights x-box.
  User tries to select the *enter text* command.
  Command fails: box already has text.

13.13 Figure A13.11 has the sequence diagrams for Exercise 13.12.

Sequence diagram for error scenario 1:



Sequence diagram for error scenario 2:



Sequence diagram for error scenario 3:

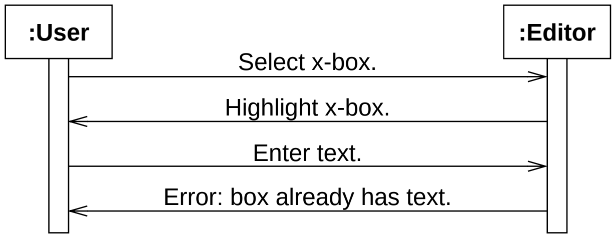


Figure A13.11 Sequence diagram for error editor scenarios

13.14 The application manages data for competitive meets in a swimming league. The system stores swimming scores that judges award and computes various summary statistics.

13.15 Actors are competitor, scorekeeper, judge, and team.

13.16 [Instructor's note: you should give the students the actors from the previous exercise.] Here are definitions for the use cases. Figure A13.12 shows a use case diagram.

- **Register child**. Add a new child to the scoring system and record the name, age, address, and team name. Assign the child a number.
- **Schedule meet**. Assign competitors to figures and determine their starting times. Assign scorekeepers and judges to stations.
- **Schedule season**. Determine the meets that comprise a season. For each meet, determine the date, the figures that will be performed, and the competing teams.
- **Score figure**. A scorekeeper observes a competitor's performance of a figure and assigns what he/she considers to be an appropriate raw score.
- **Judge figure**. A judge receives the scorekeepers' raw scores for a competitor's performance of a figure and determines the net score.
- **Compute statistics**. The system computes relevant summary information such as top individual score for a figure and total team score for a meet.

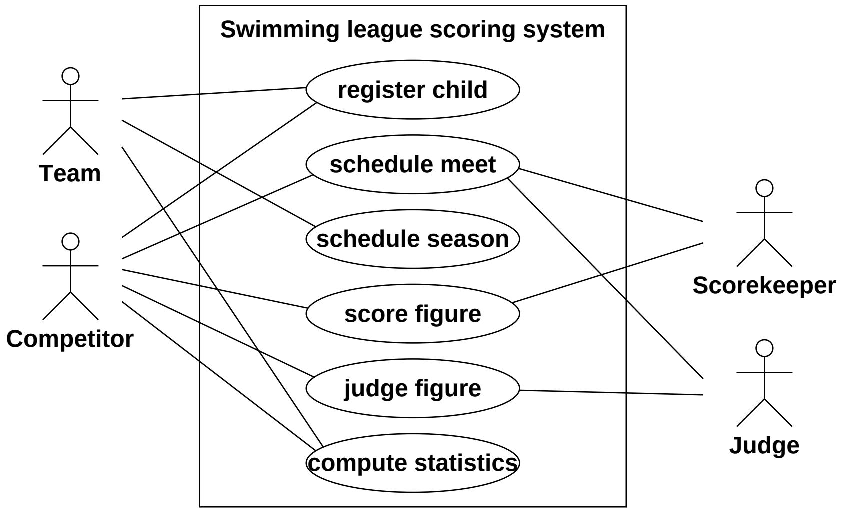


Figure A13.12 Use case diagram for the swimming league scoring system

13.17 Scenarios are intrinsically less interesting for the swimming league than for the earlier diagram editor exercise. The diagram editor is an interactive application that has an important interaction model. In contrast, the swimming league is a data management problem, where most operations merely store and retrieve data. The swimming league has a complex class model and a rich variety of queries, but relatively simple scenarios for acquiring data.

At the beginning of the 1991 swimming season, the following data was entered into the system. The two teams were the Dolphins and the Whales. The six competitors were Heather Martin, Elissa Martin, Cathy Lewis, Christine Brown, Karen Solheim, and Jane Smith. The name, age, address, and telephone number of each competitor was entered into the system. Heather Martin, Elissa Martin, and Cathy Lewis were members of the Dolphins team. The other three competitors were members of the Whales team. The three judges were Bill Martin, Mike Solheim, and Jim Morrow.

The meets were scheduled for July 6 at Niskayuna, July 20 at Glens Falls, and August 3 in Altamont. In each of the three meets the same four figures were scheduled to be performed for a total of twelve events. These figures are the Ballet Leg (1.0), the Dolphin (1.2), the Front Pike Somersault (1.3), and the Back Pike Somersault (1.5). Difficulty factors for figures are parenthesized.

For the July 6 meet at Niskayuna, the Ballet Leg event was scheduled to begin at 9:00 AM, the Dolphin at 9:00 AM, the Front Pike Somersault at 10:00 AM, and finally the Back Pike Somersault at 10:00 AM.

For the July 20 meet at Glens Falls, the Back Pike Somersault begins at 9:00 AM, the Front Pike Somersault at 9:00 AM, the Dolphin at 10:00 AM, and the Ballet Leg at 10:00 AM.

For the August 3 meet at Altamont, the Ballet Leg event was scheduled to begin at 9:30 AM, the Dolphin at 9:30 AM, the Front Pike Somersault at 10:30 AM, and the Back Pike Somersault at 10:30 AM.

13.18 First error scenario. Someone tries to schedule a meet for July 20 at Voorheesville. This is not permitted, since there is already a meet scheduled for that date at Glens Falls.

Second error scenario. Competitor Karen Solheim tries to give the registrar two phone numbers, one for her home and another for her father's office. The current class model cannot accept this data, since *telephoneNumber* is a simple attribute of *Competitor*.

Third error scenario. Competitor Cathy Lewis informs the registrar that she would like to compete with the Dolphins team on July 6 and 20 and the Whales team on August 3. Christine Brown and Jane Smith will miss the meet on August 3 due to vacation plans, and Cathy Lewis would like to even up the teams. The class model forbids this data entry, since a *Competitor* is specified as belonging to exactly one *Team*.

13.19 Printing forms. For each competitor currently in the system:

- Print out the current name, age, address, and telephone number, then
- Mail the form to each competitor.

Processing forms. There are a number of possibilities, such as adding a new competitor, deleting an old competitor, keeping records for a competitor but deactivating the competitor for the upcoming season, and changing information for a competitor. Note that *age* is a poor choice of base attribute for the *Competitor* class. The model would be improved by making *birthdate* a base attribute and *age* a derived attribute from the *currentDate* and *birthdate*.

Karen Solheim returned her form and changed her address from 722 State Street to 1561 Northumberland Drive in anticipation of an upcoming move.

Heather Martin called to complain because her sister Elissa Martin received a form and she did not. After some checking, it became apparent that the wrong address was entered in the computer for Heather Martin. Heather's address was corrected.

Both Ann Davis and Loren Jones called and said they would be unavailable to compete in the 1990 season. They were kept as competitors in the system, but were no longer assigned to a team. This required a minor change to the class model to make team optional for a competitor.

The class model also requires extension to track contestant number. We decided to add another attribute to the *Competitor* class called *number* to accomplish this. Heather Martin was assigned number 3; Elissa Martin was assigned number 4; and Cathy Lewis was assigned number 1. Christine Brown was assigned number 5; Karen Solheim was assigned number 1; and Jane Smith was assigned number 9. All these numbers were the same as last year. Two competitors may have the same number as long as they are members of different teams.

13.20 Figure A13.13 shows the activity diagram.

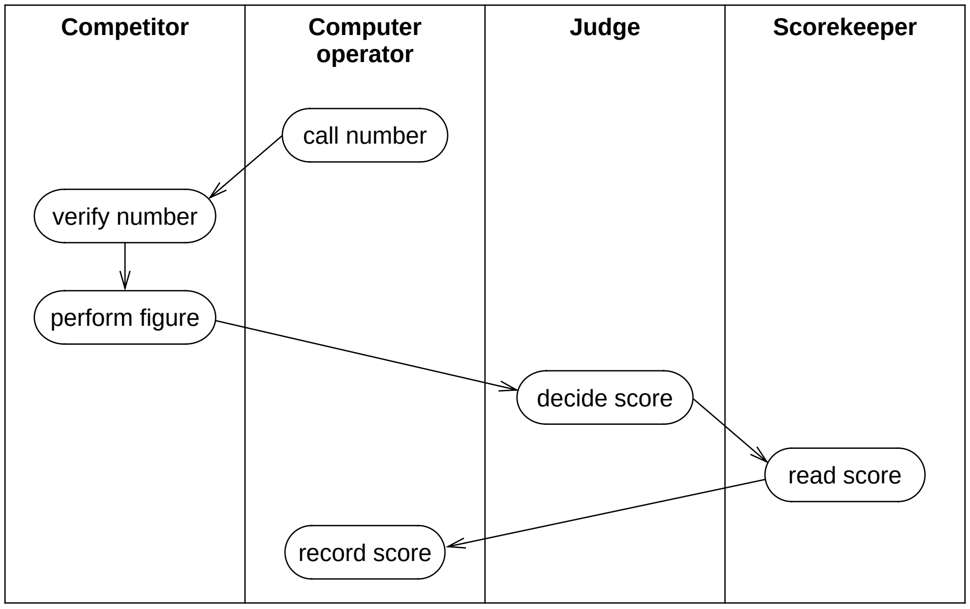


Figure A13.13 Activity diagram for recording swim scores

13.21 Figure A13.14 shows a partial shopping list of operations.

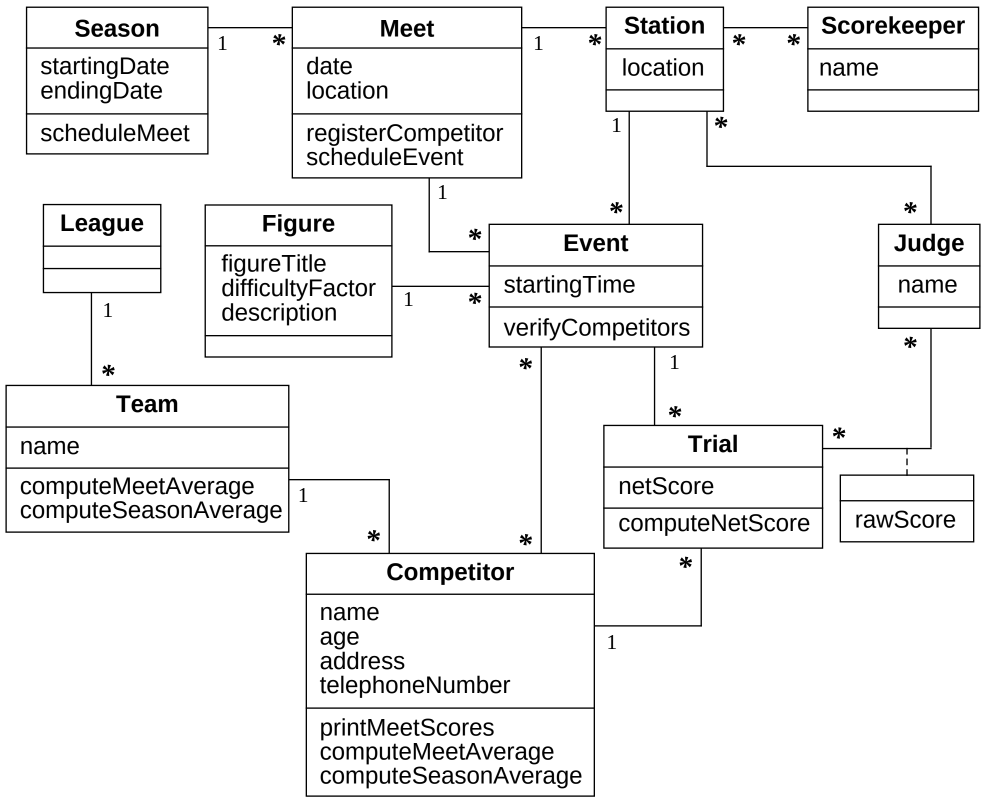


Figure A13.14 Partial class diagram for a scoring system including operations

13.22 The following bullets summarize what the operations should do.

- *scheduleMeet*. If the meet is not already known, add it to the system along with its required link. (The class model specifies that a meet must have exactly one season.) Specify a date and location for the meet.
- *registerCompetitor*. Given the name of a competitor, make sure that a *Competitor* object is entered into the system. Make sure that the competitor is assigned a team. Note that there is no explicit association between *Meet* and *Competitor*; thus we can make sure that a competitor is known to the system, but the model cannot specifically register a competitor for a meet. (Exercise 12.12l addresses this.)
- *scheduleEvent*. If the event is not already known, add it to the system along with its required links. (The class model specifies that an event must have exactly one figure, meet, and station.) Assign the event a starting time.

- *verifyCompetitors*. Make sure all attributes are filled in for each competitor. Make sure that each competitor is assigned a team.
- *computeNetScore*. For a given trial, retrieve the set of raw scores from the judges. Delete the high and low score. Average the remaining scores.
- *Team.computeMeetAverage*. For the given team, retrieve *team.competitor.trial.netScore*. For the given meet, retrieve *meet.event.trial.netScore*. Intersect these two sets of scores and compute the average of the resulting set. (Note that we have described the *computeMeetAverage* operation with a procedure. The operation need not be computed in the manner that we have specified but may be computed with any algorithm that yields an equivalent result.)
- *Team.computeSeasonAverage*. For the given team, retrieve *team.competitor.trial.netScore*. For the given season, retrieve *season.meet.event.trial.netScore*. Intersect these two sets of scores and compute the average of the resulting set.
- *printMeetScores*. For the given competitor and meet, retrieve the set of trials. For each trial, print the competitor name, meet date, meet location, and net score.
- *Competitor.computeMeetAverage*. For the given competitor and meet, retrieve the set of trials. Compute the average of the net scores for these trials.
- *Competitor.computeSeasonAverage*. For the given competitor and season, retrieve the set of trials. Compute the average of the net scores for these trials.