CS211 Lecture: Modeling Dynamic Behaviors of Systems; Interaction
Diagrams and Statecharts Diagrams in UML

last revised September 18, 2002

Objectives:

1. To introduce the notion of dynamic analysis
2. To show how to create and read Sequence Diagrams
3. To show how to create and read Collaboration Diagrams
4. To show how to create and read Statecharts

Materials:

1. Handout of baseball double play Sequence and Collaboration Diagrams
2. Handout of Session Sequence diagram
3. Handout of Session Collaboration diagram
4. Handout of ATM AddressBook use case flows of events and CRC cards
5. Handout: ATM session statechart
6. Transparency: statechart for Lab 3 GUI
7. Transparencies of statecharts on book pages 281 (top), 282 (both)
8. Handout: code for Session class performSession() method

I. Introduction

A. In trying to understand any system, there are really two aspects of the system to be considered: its static aspects and its dynamic aspects. The static aspects of a system have to do with its component parts and how they are related to one another; the dynamic aspects of a system have to do with how the components interact with one another and/or change state internally over time.

Example: Suppose we were analyzing the game of baseball - in particular, the way that a team functions when it is on defense (out in the field)

1. The static aspects of this system include things like the fact that there are 9 players on the field at a time, and the various positions that they play. In fact, if you were watching a baseball game, this is mostly what you would see - most of the time the players are in their positions, ready for a ball to come their way.

2. The dynamic aspects of this system include the various interactions (“plays”) that occur under various circumstances. For example, one such interaction is called a “double play”, which can happen when there is a runner on first base and the batter hits a ground ball. It commonly takes one of two forms:
a) If the ball is hit between second and third bases, then the shortstop runs to the ball while the second baseman runs toward second base. The shortstop fields the ball and throws it to the second baseman, who in turn throws it to the first baseman. In baseball jargon, this is called a “6-4-3 double play”.

b) On the other hand, if the ball is hit between first and second bases, then the second baseman fields the ball and throws it to the shortstop (covering second) who in turn throws it to first. In baseball jargon, this is called a “4-6-3 double play”.

B. Thus far, we have largely discussed static aspects of systems - e.g. a class diagram represents the (static) relationship between the classes comprising a system, etc. CRC cards are developed by thinking through the dynamic behavior of a system, but basically record static information (who is responsible for what; who collaborates with whom, etc.)

C. We now want to consider various ways of modeling the dynamic behavior of a system.

1. We are exploring this topic in the context of system design, but actually this kind of modeling is relevant (in different ways) both during analysis and during detailed design as well.

2. Example: As part of doing the domain analysis of a traffic light system, it is important to note that individual signals go through a series of states in a prescribed order - modeled by the following statechart:

![Statechart Diagram]

(Note: this is correct for the US but not for all countries in the world!)

An important “business rule” that any traffic light system must obey is that the light must be yellow for a certain minimum period of time (related to vehicle speed in the intersection) between displaying green and displaying red. Ensuring that a system exhibits this behavior is essential for safety.
D. We will now consider several tools that can be used to model various aspects of the dynamic behavior of a system. Each can be used either for analysis (if our goal is to record how behavior actually transpires in a system we are modeling) or for design (if our goal is to describe a behavior we want to produce.)

1. Interaction Diagrams - of two different kinds
   a) Sequence Diagrams
   b) Collaboration Diagrams

2. Statechart Diagrams

3. In a subsequent lecture, we will consider Activity Diagrams. (The book discusses this with the other two, but we will deal with later.)

II. Interaction Diagrams

A. The purpose of an interaction diagram is to describe how various objects comprising a system interact with one another to accomplish some task. For example, the following interaction diagram shows a “6-4-3” double play: It is more the kind of diagram that would be produced during analysis than a diagram that would be produced to design software. (HANDOUT)
B. Interaction diagrams are also a useful design tool. In order to be useful to us, the design that emerges from our analysis and our work with use cases and CRC cards must be documented in some suitable fashion. While CRC cards are a good tool for developing a design, they do not serve well as a permanent documentation tool. An interaction diagram is a useful vehicle for describing the interaction that lies behind a set of CRC cards. (We’ll look at an example of this shortly.)

C. UML actually defines two different types of interaction diagram: a sequence diagram and a collaboration diagram. The two types of diagram are equivalent, in the sense that either one can be constructed from information in the other. For example, the following collaboration diagram shows the same interaction (ALSO IN HANDOUT)

D. I will focus first on the sequence diagram, because I think it is the easier of the two for a beginner to understand and construct.

1. HANDOUT: Sequence diagram for Session.

2. General observations:

   a) An interaction diagram (of either type) depicts a society of objects that work together to carry out a particular use case. Thus, in general, there will be an interaction diagram for each use case.

   b) The vertical dimension in a sequence diagram is time. The interaction starts at the top of the diagram, and progresses through time to the bottom.

   c) The boxes in the interaction diagram stand for objects. Note that UML uses very similar notation for objects and for classes in
diagrams - both are represented by rectangles. Two things distinguish the symbol for an object from that for a class:

(1) An object has both an object name and a class name. These are written separated by a colon - e.g. joe: Student. In a UML diagram, it is common for an object to be anonymous - in which case the box contains just the class name, *preceded* by a colon. (If there were a name followed by a colon, it would be an object name of anonymous class.)

(2) In UML object diagrams, it is common to *underline* the object: class name. (This is not done consistently, though; one of the three book UML set does it and another doesn’t!) I had to add this to the diagram I have handed out, because the drawing tool I used to produce it doesn’t provide for this!

(3) Underneath each object is a dotted line called its *lifeline*. The lifeline indicates when the object is in existence.

(a) If the lifeline extends the entire length of the diagram, it means that the object exists before the interaction begins and continues to exist after it ends.

i) Examples from double-play

ASK

Shortstop, Second Baseman, First Baseman

ii) Examples from ATM Session

ASK

CardReader, ATM, CustomerConsole

(b) If the lifeline begins part-way through the interaction, it indicates that the corresponding object is created during the interaction. Likewise, if the lifeline terminates in an “X” before the end of the interaction, this indicates that the corresponding object is destroyed during the interaction.

i) Examples from double play:

ASK
Batter and Runner are both destroyed in the interaction (a)

ii) Examples from session

Session, Transaction are both created and destroyed in the interaction

iii) It is also possible - though not illustrated in either example - for an object created during an interaction to continue to exist after the interaction completes

(c) Rectangular boxes on the lifeline show times during the interaction when a particular object is active - i.e. performing some operation, or waiting for some other object to perform an operation and return a result.

(Discuss examples on charts)

d) The diagram shows messages sent from one object to another. We’ll illustrate these from the session example. Each message has:

(1) A direction - indicated by the arrow near the message description. One of the objects (at the tail of the arrow) is the sender of the message and the other (at the head end) is the receiver of the message.

(2) A name

(a) EXAMPLE: the first message from the CardReader to the ATM is called cardInserted().

(b) NOTE: The first message from the ATM object to the Session object is called « create ». This is a stereotype, indicating a special kind of message (actually sent to the class) which results in creating the object. The object does not exist before the « create » message is sent.

(c) If the collaboration is implemented in Java, the message name will generally become the name of a method.

(3) A (possibly empty) list of parameters

(a) EXAMPLE: the « create » message from the ATM to Session has as its parameter the atm object (this) that is sending the message.
(b) **EXAMPLE:** The « create » message from the Session to Transaction has as parameters the ATM object (that created the session in the first place), the Session object (this), the customer’s card, and the pin the customer typed.

(c) **EXAMPLE:** Looking at the messages from and to the CardReader, we see that none have parameters - all parameter lists are empty.

(d) If the collaboration is implemented in Java, the parameters will become actual parameters to the method call.

e) Some messages have a return value. This is indicated by a dotted line going back from the recipient to the original sender.

**EXAMPLE:** readCard() returns a card to the Session; readPIN() returns a pin to the session; performTransaction() returns to the Session and indicator as to whether the customer wants to do another.

f) Sometimes a message or group of messages is sent repeatedly. This is indicated by enclosing the messages in a box, with a repetition condition specified.

**EXAMPLE:** The box indicating the possibility of multiple transactions.

If the interaction is implemented in Java, an iteration clause will be translated into an appropriate kind of loop statement.

g) Not shown in the example is the possibility than an object might send a message to itself. We will see an example of this in a moment.

3. **SHOW ADDITIONAL SEQUENCE DIAGRAMS ON THE WEB** - note messages sent by a Transaction object to itself (to take advantage of polymorphic methods)

E. Another form of interaction diagram in UML is the Collaboration Diagram.

NOTE two diagrams in Double-Play handout

HANDOUT: Collaboration Diagram for Session Use Case
1. Comparing the two types of diagram for the same use case is instructive.
   
a) The sequence diagram focuses on time sequence

b) The collaboration diagram focuses on object relationships

2. In a collaboration diagram, as in a sequence diagram, each object participating in an interaction is represented by a box.

3. The objects are connected by lines representing links. If a link has no arrows on it, it is taken as being bidirectional: the object at each end knows about the object at the other end. If there is an arrow, it is unidirectional: the object at the tail end knows about the object at the head end, but not vice versa. In the case of a bidirectional link, we must indicate the direction of the message with an arrow, as shown on the diagram.

   EXAMPLE: Who knows about whom in the example diagram?

4. Time sequence is specified by the use of sequence numbers. A number followed by a colon (e.g. 1: 3.2: etc.)

   a) The sequence numbers indicate the order in which the messages are sent. Message 1 is sent first, then 2, etc.

   b) Multiple numbers separated by dots represent nesting of messages.

   EXAMPLE: carrying out what is required by message 3 (performSession()) results in the sending of message 3.1 (readCard()) then 3.2 (getPIN()), etc.

   c) NOTE: To avoid excessive complexity, it is possible to draw layers of collaboration diagrams. A coarse grained diagram may show only the main level of messages (1, 2, 3). These may be refined in subsequent diagrams - e.g. there might be a separate diagram showing the messages associated with top-level message 2 (2.1, 2.2, etc.). We will not deal with examples complex enough to call for this.

5. If a message returns a value to its caller, this is denoted by an assignment operator - so that the message name looks like a function call. (The := operator is used for assignment in Pascal and Ada, and has been adopted by UML instead of =.)
F. As a general rule, the implementation of each use case will be described by an interaction diagram (of one type or the other) showing the objects that are needed to carry out that use case.

1. Thus, there will be at least one interaction diagram for each use case.

   **EXAMPLE:** Show links from Use Case page to Interactions on the web example

2. Interaction diagrams can be created at the same time as CRC cards, to detail the actual flow of messages that we discover doing the CRC role-playing. Or, they can be created later, during detailed design of the various classes.

3. It is also possible to use interaction diagrams for other purposes - e.g. if a class has an operation (method) that is sufficiently complex, that operation may be designed using its own interaction diagram.

G. CLASS EXERCISE: Do Interaction Diagrams for CS112 AddressBook problem.

1. **HANDOUT:** CRC cards - note that responsibilities from each use case have been annotated as to where they came from. This is not normally done, but is done here for clarity.

   Go over

2. Use same teams and do same use cases as for CRC card exercise.

3. Allow time to develop a sequence diagram for assigned use case.

4. Have one team for each use case draw their sequence diagram on the board - discuss.

5. If time, also do a collaboration diagram.
III. State Charts

A. Often, while we analyze a system, we discover that some objects undergo defined state transitions over time, which is an important part of the dynamic behavior of the system.

1. Example: we looked earlier at an example of this in the domain of traffic lights.

2. What objects in our ATM example fall into this category?

ASK

- a) The ATM itself goes through a series of states (off, on, serving a customer, etc.)
- b) Individual sessions pass through a series of states
- c) Individual transactions also pass through a series of states.

(Note that the objects that have this characteristic are often controller objects.)

B. For such objects, it may prove helpful to develop a Statechart Diagram, showing the states it passes through - especially if there is any complexity to the state transitions.

1. Example: We will develop a state transition diagram for a session.

   a) Referring to the use case for a session, what distinct states can we identify?

      ASK

      (1) Reading card
      (2) Asking for PIN
      (3) Asking for transaction choice
      (4) Performing transaction
      (5) Ejecting card at the end of the session
      (6) Session finished

   b) Further, the transitions between states for a session follow a fairly complex pattern - e.g.
(1) After reading a card, we go either to asking for PIN or to ejecting card, depending on whether the card was readable.

(2) After asking for PIN, we normally go to asking for a transaction choice; but we go to ejecting card if the customer presses cancel.

(3) After asking for a transaction choice, we normally go to performing the transaction, but we go to ejecting card if the customer presses cancel.

(4) After performing a transaction, we can go one of three ways:

   (a) To choosing transaction, if customer says he/she wants to do another.

   (b) To ejecting card, if customer says he/she doesn’t want to do another.

   (c) To session finished, if the card was retained due to too many invalid PINs.

*HANDOUT STATE CHART DIAGRAM FOR SESSION FROM WEB*

d) For what other objects would such a diagram prove useful?

*ASK*

*SHOW REMAINING STATECHARTS ON WEB*

2. In general, a statechart is a useful analysis tool for:

   a) An object that is responsible for a use case, provided that the use case has enough internal complexity to warrant doing a statechart. (If the use case consists of just a few steps, with no complex variation in sequencing, then a statechart may not be warranted.)

   b) An object has complex internal states, even though it is not directly responsible for a use case

   *Example:* In the case of the ATM simulation, the object representing the network connection to the bank would likely warrant a statechart, because network connections typically pass through a series of states as they dialog with a peer on the other
end. We have omitted that from the example, because we’ve
avoided getting into the details of how the network connection
works. A real system would, of course, have to deal with this.

C. Another place where statecharts are often useful is in the design of
graphical user interfaces, as we noted in the previous series of lectures.

1. The states correspond to the different visible states of the GUI - i.e.
what screen etc. is being displayed and what buttons/menus are active.

2. The transitions correspond to the various user gestures - i.e. options the
user may choose.

3. Your lab 3 included an example of this.
D. A Statechart diagram makes use of several symbols:

1. A rounded rectangle denotes a state.

2. Arrows connecting states denote state transitions. A transition is labeled with one or two pieces of information
   a) The situation under which the transition occurs (always)
   b) Action that takes place on the transition (optional)

   (in our examples, actions appear on the transitions for the ATM itself, but not for Session and Transaction.)

3. A filled in dot denotes the initial start - where the system starts.

4. A filled in dot in a circle denotes a final state. When a system reaches the final state, no further state transitions are possible.

   (Note that the diagrams for Session and Transaction have final states; but the ATM state diagram does not - turned off is one of its states, but there is no final state per se because it can always be turned back on.)

5. Additional notation is possible, but this is more advanced than what we want to talk about now.

E. It is also possible to incorporate information about specific actions to be performed during a state, or when entering or leaving it.

1. The statechart for the Overall ATM includes entry actions for 2 of the three states.

2. The jukebox example in the book (p. 281) shows an example of an action to be formed continuously throughout a state.

3. The tape-recorder example in the book (p. 282) shows an example of an exit action.

F. Finally, it is sometimes helpful to incorporate a state chart within a statechart, describing a complex internal structure for a single state.

1. This can be done by physically incorporating the nested statechart into the larger state on the diagram.

   EXAMPLE: Text p.282 TRANSPARENCY
2. Alternately, the sub chart can be drawn separately and referenced by an include in the containing state

EXAMPLE: The overall ATM chart’s SERVING CUSTOMER state explicitly includes the entire Session statechart; its PERFORMING TRANSACTION state explicitly includes the Transaction statechart.

3. Physical inclusion is necessary, as in the example in the book, when the sub chart can be entered at more than one place. Inclusion by the use of include makes an easier to read chart if the included statechart has only a single entry point.

G. A statechart can often be translated into code in a very straightforward way that preserves the clarity of the statechart

HANDOUT: Code for performSession () method of Session - compare to Statechart diagram.

H. Class Exercise: Develop a statechart for a single video in a video store.

1. What states can it be in?

   ASK

   On shelf (in the store)
   Checked out to a customer
   On hold (at the desk, waiting for a customer to pick it up)
   On rack to be sold.

2. Draw states, fill in transitions:- including initial state, and possibility of discarding or selling - each leading to final state.