Chapter 6

UML Diagrams (Dynamic View)

6.1 Introduction

Objects interact to implement behaviors. These interactions can be described in two complementary ways: one focuses on individual objects (using state-transition diagrams), while the other focuses on groups of cooperating objects (using interaction diagrams).

Interaction diagrams bridge the gap between use case diagrams and class diagrams by illustrating how objects—instances of classes—communicate to achieve specific functionalities. They introduce a dynamic aspect to system modeling.

When creating an interaction diagram, the focus should be on a subset of system elements to analyze how they interact in order to describe a particular behavior.

6.2 General Representation of an Interaction Diagram

An interaction diagram is represented by a rectangle. In the upper-left corner, a pentagon appears alongside a keyword:

- `sd` for a sequence diagram
- `com` for a communication diagram

This keyword is followed by the name of the interaction.

6.3 Communication Diagram

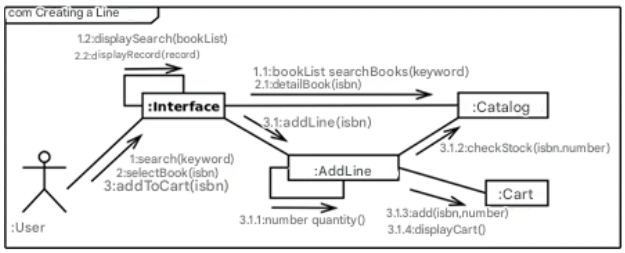

Figure 6.1: Communication diagram illustrating the search and addition of a book to a virtual basket during an online order.

Communication diagrams help validate associations in a class diagram by using them as conduits for message transmission.

Note: A communication diagram is a type of UML 2.0 interaction diagram (previously called a collaboration diagram in UML 1.x).

6.3.1 Representation of Lifelines

Lifelines are depicted as rectangles containing a label (the object name). **Note:** The class name of the object instance is preceded by a colon (:) and underlined. To distinguish objects of the same class, an identifier may be added before the class name. Example: `p1:Person` or `:Person` designates an object of the `Person` class.

6.3.2 Representation of Connectors

Connectors represent relationships between lifelines. They are drawn as solid lines connecting two lifelines, and their ends may be decorated with multiplicity indicators.

6.3.3 Representation of Messages

In a communication diagram, messages are typically ordered using an increasing sequence number.

6.4 Sequence Diagram

Sequence diagrams emphasize the chronological order of messages exchanged between lifelines. Unlike communication diagrams, time is explicitly represented along the vertical axis, flowing from top to bottom.

6.4.1 Representation of Lifelines

A lifeline is represented by a rectangle attached to a vertical dashed line, labeled with the object's name.

6.4.2 Representation of Messages

A message defines a specific type of communication between lifelines. Common message types include:

- Sending a signal
- Invoking an operation
- Creating or destroying an instance

Asynchronous Messages

Figure 6.2: Representation of an asynchronous message.

Asynchronous messages are suitable for modeling interrupts or events. They are represented by a solid arrow with an open arrowhead, pointing from the sender's lifeline to the receiver's lifeline.

Synchronous Messages

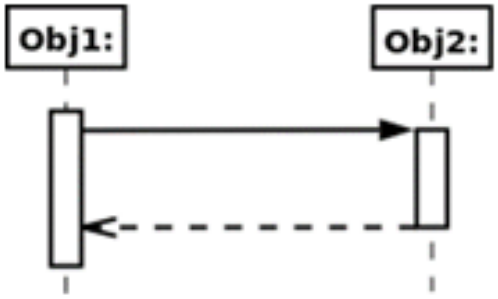

Figure 6.3: Representation of a synchronous message.

Invoking an operation is the most common message type in object-oriented programming. Graphically, a synchronous message is shown as a solid arrow with a filled arrowhead, pointing from the sender to the receiver. A reply may follow, represented by a dashed arrow.

Instance Creation and Destruction Messages

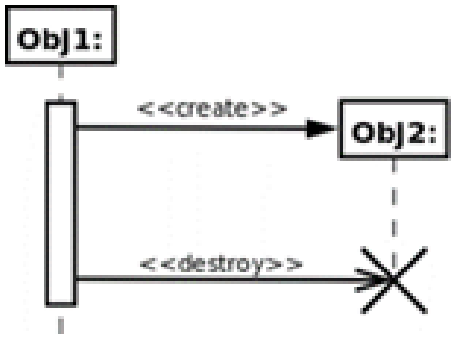

Figure 6.4: Representation of instance creation and destruction messages.

Object creation is represented by an arrow pointing to the top of a new lifeline. Object destruction is indicated by a cross (X) marking the end of the lifeline.

6.4.3 Events and Messages

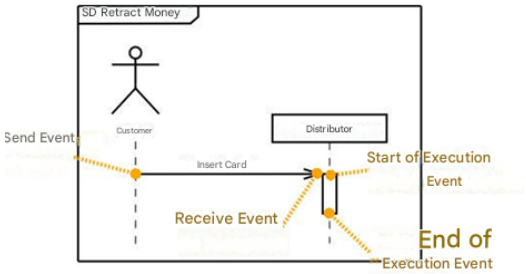

Figure 6.5: Different events corresponding to an asynchronous message.

Receiving a message usually triggers the execution of a class method. The message syntax allows arguments to be passed. Sequence numbers are often omitted since the vertical position already indicates order.

6.4.4 Combined Interaction Fragments

A combined fragment is represented by a rectangle with a pentagon in the upper-left corner containing an interaction operator.

Common operators include:

- `alt`, `opt`, `break`, `loop`
- `par`, `critical`
- `ignore`, `consider`, `assert`, `neg`
- `seq`, `strict`

Example: alt Operator

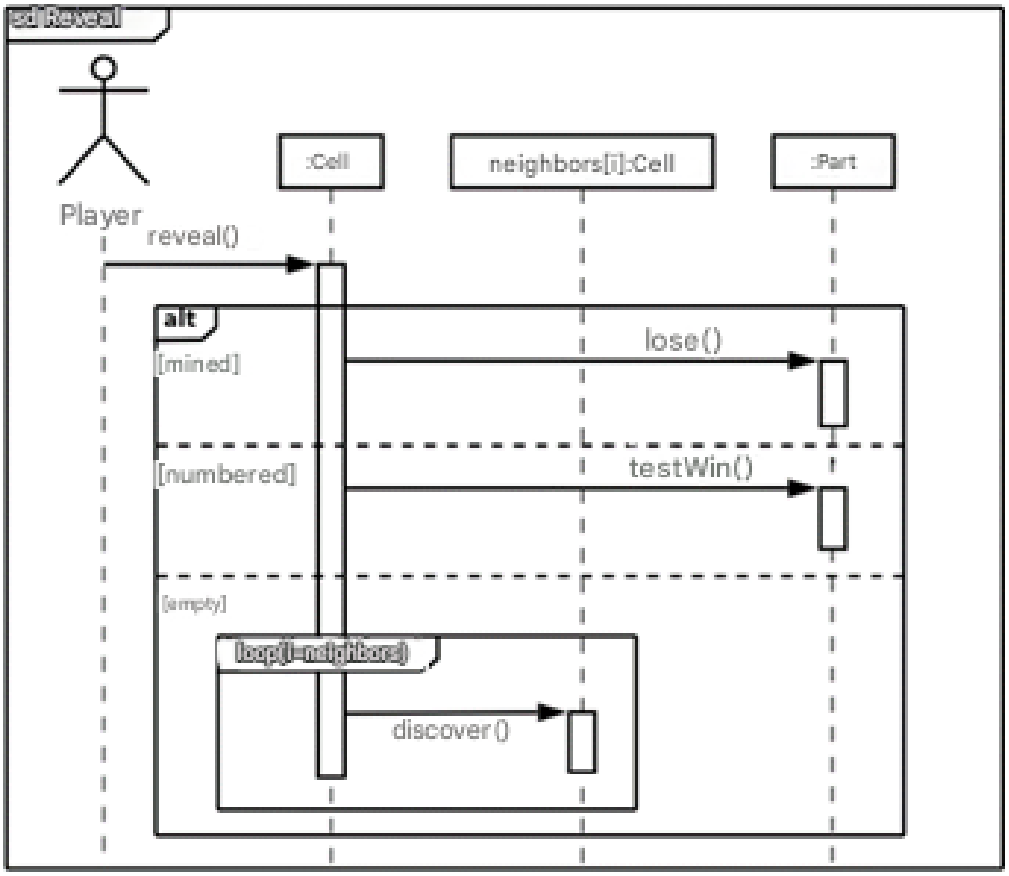

Figure 6.6: Representation of a choice in a sequence diagram illustrating the discovery of a square in the Minesweeper game.

The `alt` operator is a conditional operator with multiple operands, each having a guard condition. If multiple conditions are true, the choice is non-deterministic.

6.5 Activity Diagram

Activity diagrams focus on processes and are well-suited for modeling control and data flow.

6.5.1 Activity and Transition

Action

An action is the smallest operation expressible in UML. Examples include assigning a value, accessing a property, creating an object, or sending a signal.

Activity

An activity defines behavior through an organized sequence of units (actions). Execution flow is modeled using nodes connected by transitions.

Activity Node

Figure 6.7: Graphical representation of activity nodes.

Three families exist: executable nodes, object nodes, and control nodes.

Transition

Figure 6.8: Graphical representation of a transition.

Transitions are represented by solid arrows connecting activities and defining control flow.

6.5.2 Example: Activity Diagram with Control Nodes

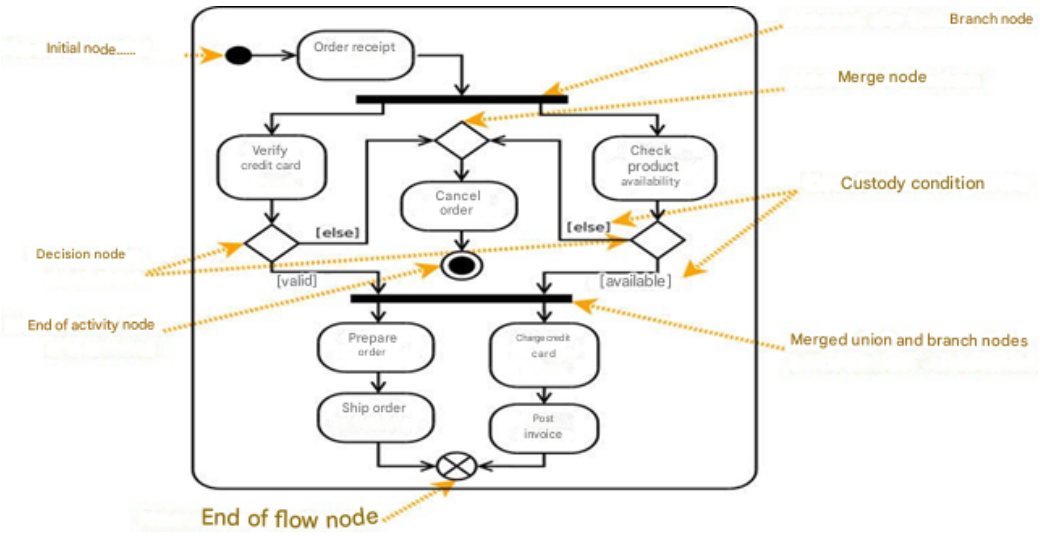

Figure 6.9: Example of an activity diagram illustrating the use of control nodes to describe order processing.

6.6 State Machine Diagram

6.6.1 Introduction

UML state machine diagrams describe the internal behavior of an object using finite state automata.

6.6.2 State Machine Diagrams

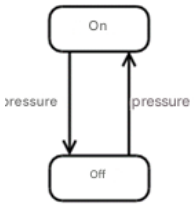

Figure 6.10: A simple state machine diagram.

6.6.3 States in a State Machine Diagram

Figure 6.11: Simple state example.

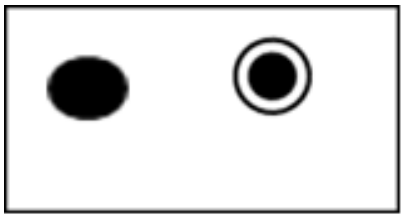

Figure 6.12: Graphical representation of initial and final states.

6.6.4 Events

Events occur at a specific point in time and may trigger transitions. Event types include signal, call, change, and time events.

6.6.5 Transition

Figure 6.13: Graphical representation of a transition between two states.

6.6.6 Choice Modeling

Junction Point

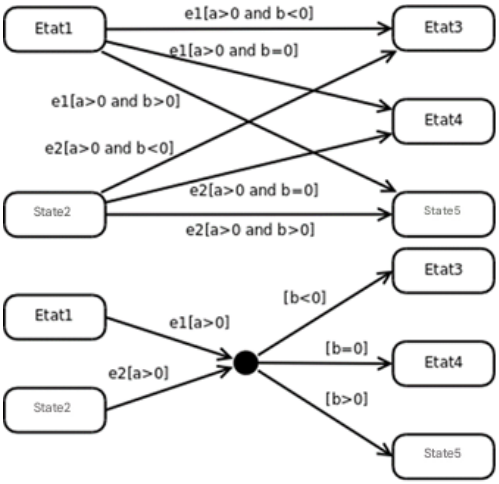

Figure 6.14: Diagram representing a junction point.

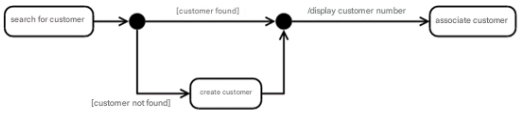

Figure 6.15: Example of using two junction points to represent an alternative.

Decision Point

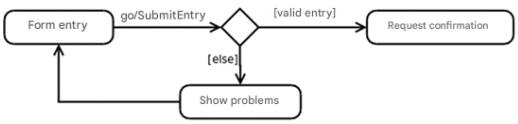

Figure 6.16: Example of using a decision point.

6.6.7 Composite States

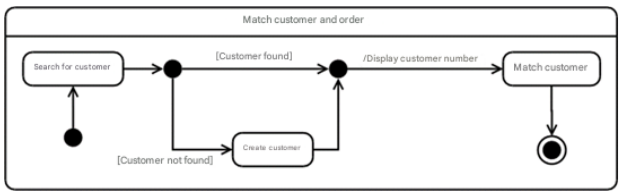

Figure 6.17: Example of a composite state modeling the association of an order with a customer.

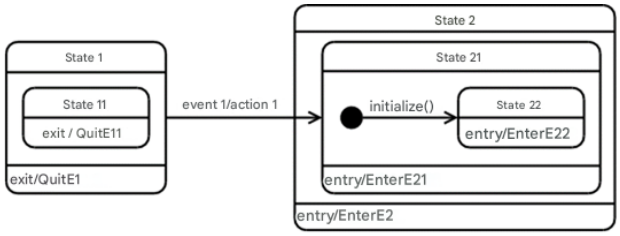

Figure 6.18: Example of a complex transition configuration.

Exercise

1. Model cash withdrawals using a VISA card with an activity diagram, considering:
   - The card may be invalid.
   - If valid, the customer must enter their PIN.
   - The card is retained after three unsuccessful attempts.
   - The system authorizes or refuses withdrawals.
   - An unclaimed card is retained.
   - A receipt is always printed.
2. Model the nominal (success) scenario using a sequence diagram.

References

- G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.
- B. Charroux, A. Osmani, Y. Therry-Mieg, *UML2 Synthesis and Exercises*, Pearson France, 2005.
- G. Booch et al., *Object-Oriented Analysis and Design with Applications*, Addison-Wesley, 2007.
- UML 2.0 course by Laurent Audibert, available at: `http://www.developpez.com`