HY-351:
Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων
Information Systems Analysis and Design

Πανεπιστήμιο Κρήτης, Φθινόπωρο 2005

Φροντιστήριο 5

Θέμα : Interaction Diagrams / State Machine Diagrams

Ημερομηνία : 21 Νοεμβρίου 2005

Outline

- Interaction Diagrams
  - Sequence diagrams
  - Communication diagrams
- State Machine Diagrams

- *For each Diagram*
  - *Definition*
  - *Elements*
  - *Demonstration*
  - *Tool tricks*

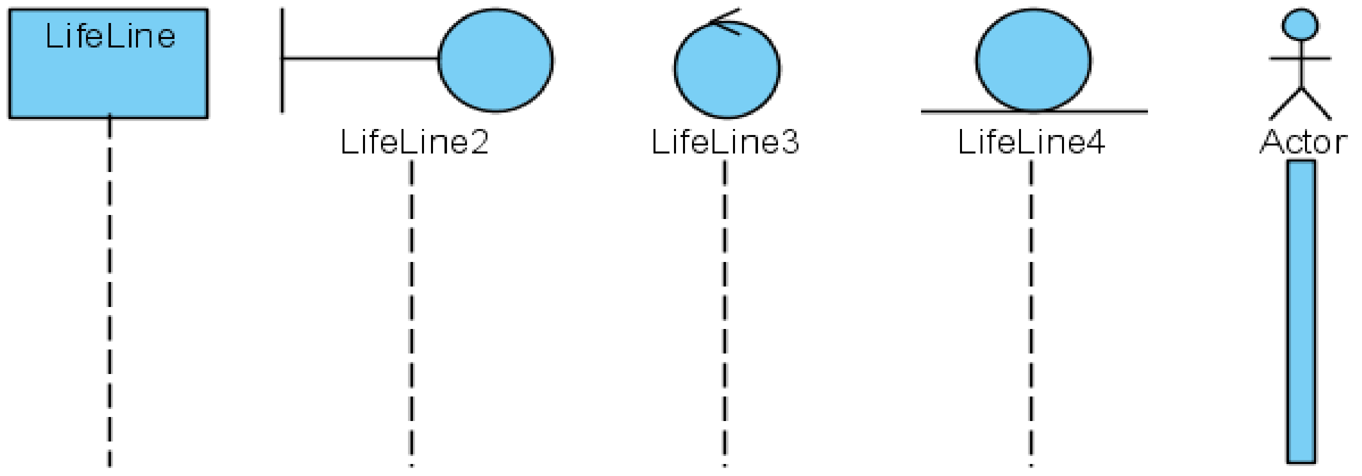
Sequence Diagrams

- **Definition** : An interaction diagram that represents objects as lifelines running down the page.
- The interactions between the objects are represented as messages drawn as arrows from the source lifeline to the target lifeline.
- Good at showing which objects communicate with which other objects and what messages trigger those communications.
- **NOT** intended for showing complex procedural logic.

Sequence Diagrams

- Lifelines
- Messages
  - Execution Occurrence
  - Self Message
  - Lost and Found Messages
- Lifeline Start and End
- Duration and Time Constraints
- Combined Fragments
- Gate
- Part Decomposition
- State Invariant / Continuations

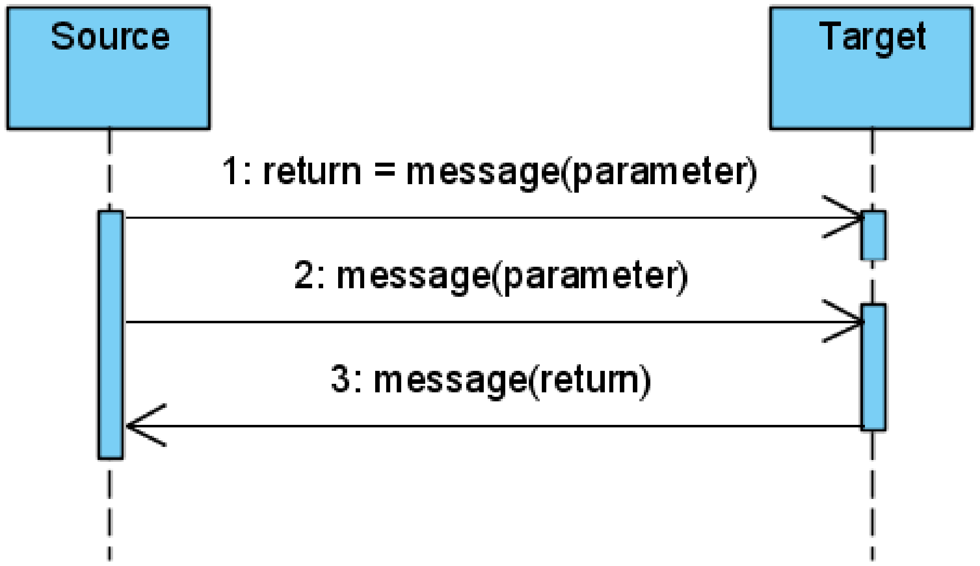
Sequence Diagrams - Lifelines



- A lifeline represents an individual participant. If its name is **self** then that indicates that the lifeline represents the **owner** of the sequence diagram.
- Sometimes a sequence diagram will have a lifeline with an actor element symbol at its head. This will usually be the case if the sequence diagram is owned by a use case.
- Boundary, control and entity elements from robustness diagrams can also own lifelines.

Sequence Diagrams - Messages

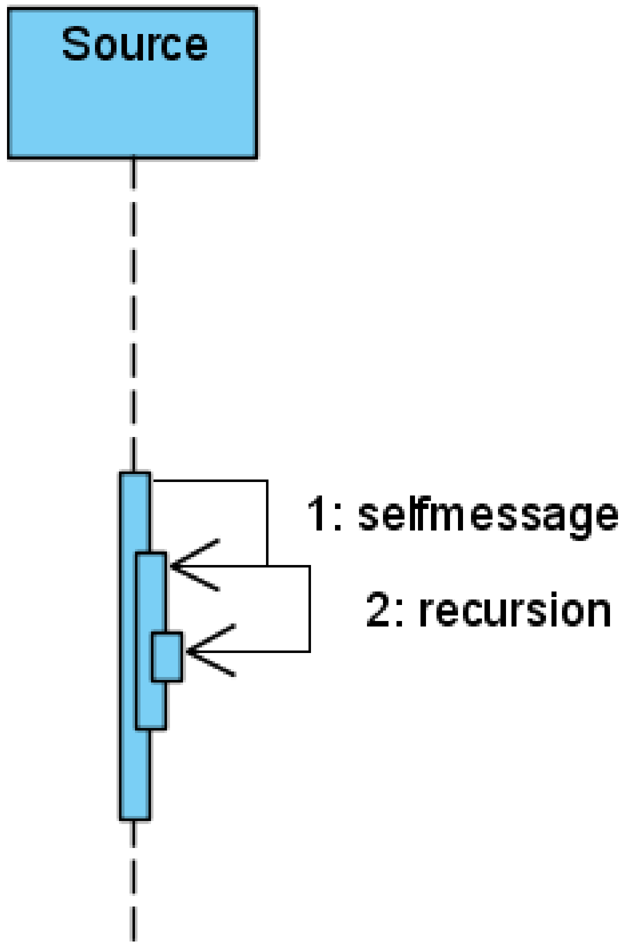
- complete
  - implicit return message
- lost or found
- synchronous
  - denoted by the solid arrowhead
- asynchronous
  - denoted by the dashed line
- call or signal



Most messages look alike, pay attention to the dashed line, or the strong arrow at the end

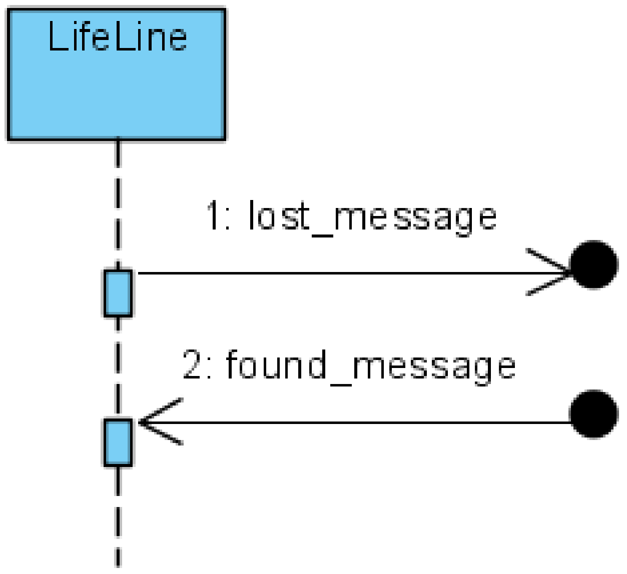
Sequence Diagrams – Self Message

- A self message can represent a recursive call of an operation, or one method calling another method belonging to the same object.
- It is shown as creating a nested focus of control in the lifeline's execution occurrence.



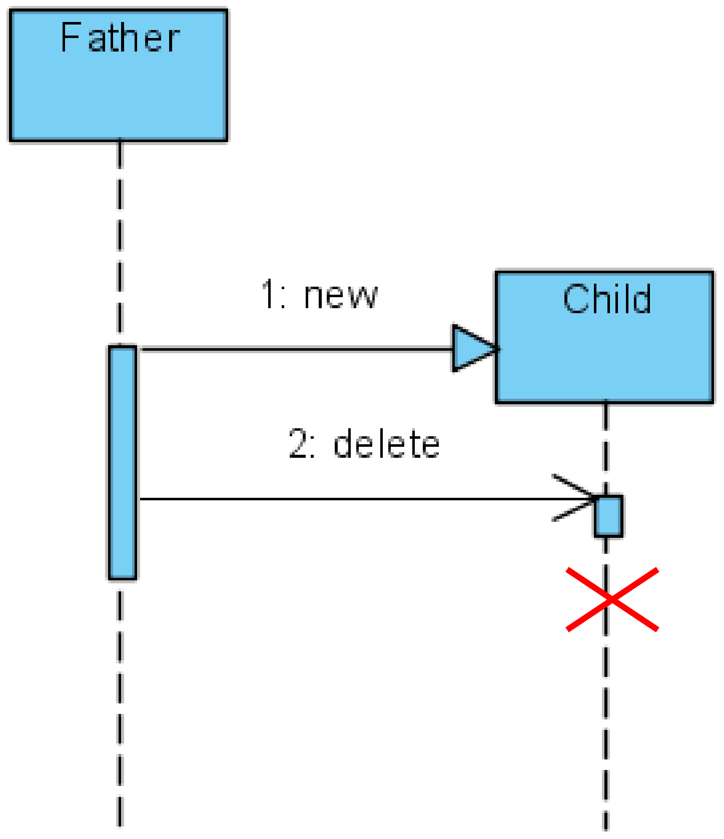
Sequence Diagrams – Lost and found Messages

- Lost messages are
  - those that are either sent but do not arrive at the intended recipient, or which go to a recipient not shown on the current diagram.
- Found messages are
  - those that arrive from an unknown sender, or from a sender not shown on the current diagram. They are denoted going to or coming from an endpoint element.

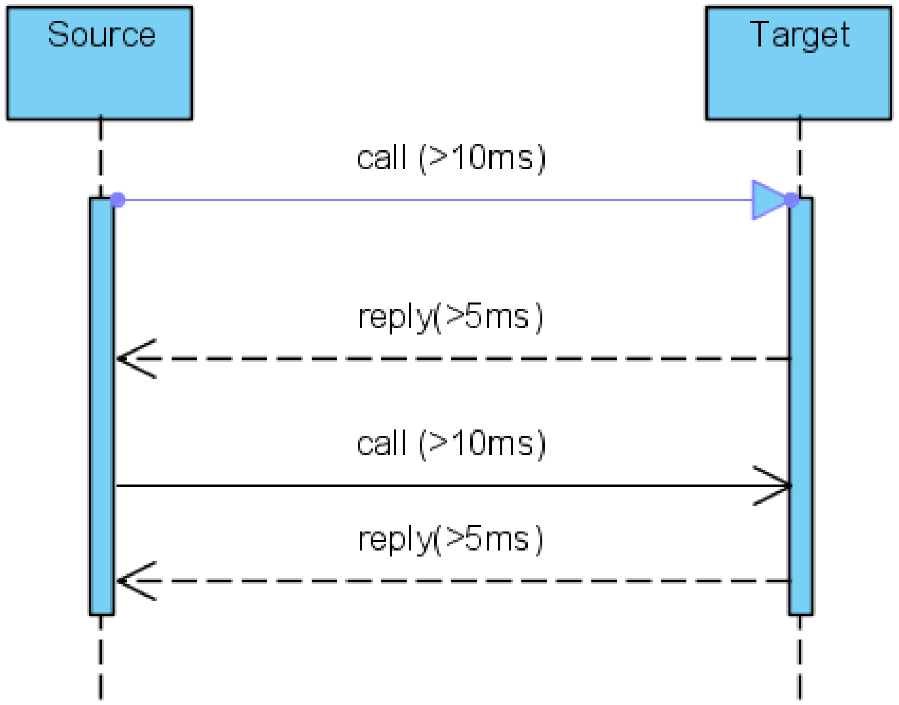


Sequence Diagrams – Lifeline start and end

- A lifeline may be created or destroyed during the timescale represented by a sequence diagram.
- Creation is denoted by the symbol at the head of the lifeline.
- Destruction is marked by an x-mark.



Sequence Diagrams - Duration and Time Constraints

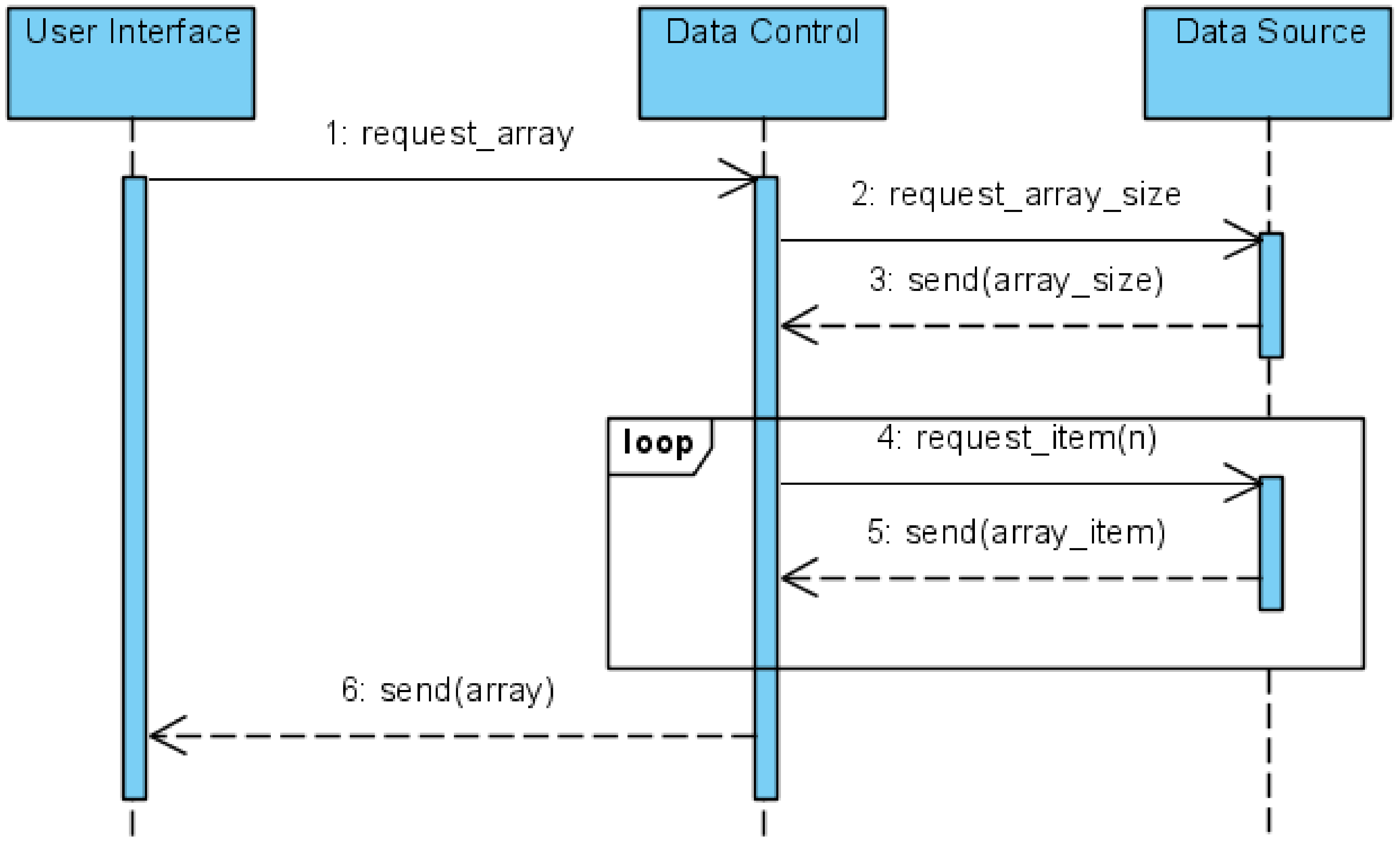


- The lifeline represents the passage of time down the screen.
- When modeling a real-time system, or even a time-bound business process, it can be important to consider the length of time it takes to perform actions. By setting a duration constraint for a message, the message will be shown as a sloping line.

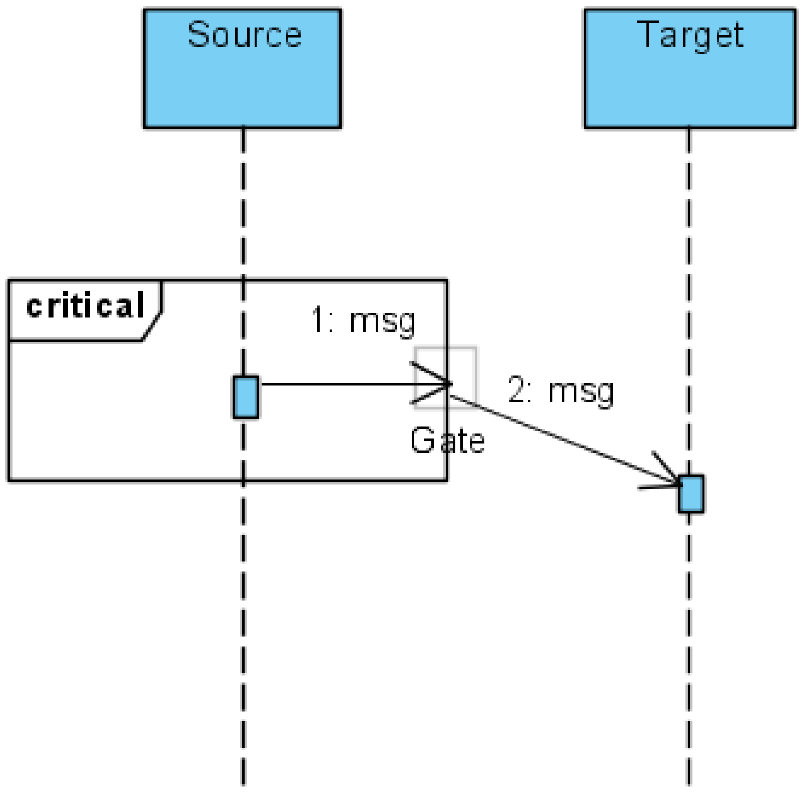
Sequence Diagrams - Operators

- “**Alt**” models if…then…else constructs.
- Option fragment (denoted “**opt**”) models switch constructs.
- **Break** fragment models an alternative sequence of events that is processed instead of the whole of the rest of the diagram.
- Parallel fragment (denoted “**par**”) models concurrent processing.
- Weak sequencing fragment (denoted “**seq**”) encloses a number of sequences for which all the messages must be processed in a preceding segment before the following segment can start, but which does not impose any sequencing within a segment on messages that don’t share a lifeline.
- Strict sequencing fragment (denoted “**strict**”) encloses a series of messages which must be processed in the given order.
- Negative fragment (denoted “**neg**”) encloses an invalid series of messages.
- Critical fragment encloses a **critical** section.
- **Ignore** fragment declares a message or message to be of no interest if it appears in the current context.
- **Consider** fragment is in effect the opposite of the ignore fragment: any message not included in the consider fragment should be ignored.
- Assertion fragment (denoted “**assert**”) designates that any sequence not shown as an operand of the assertion is invalid.
- **Loop** fragment encloses a series of messages which are repeated.

Sequence Diagrams – Combined Fragments

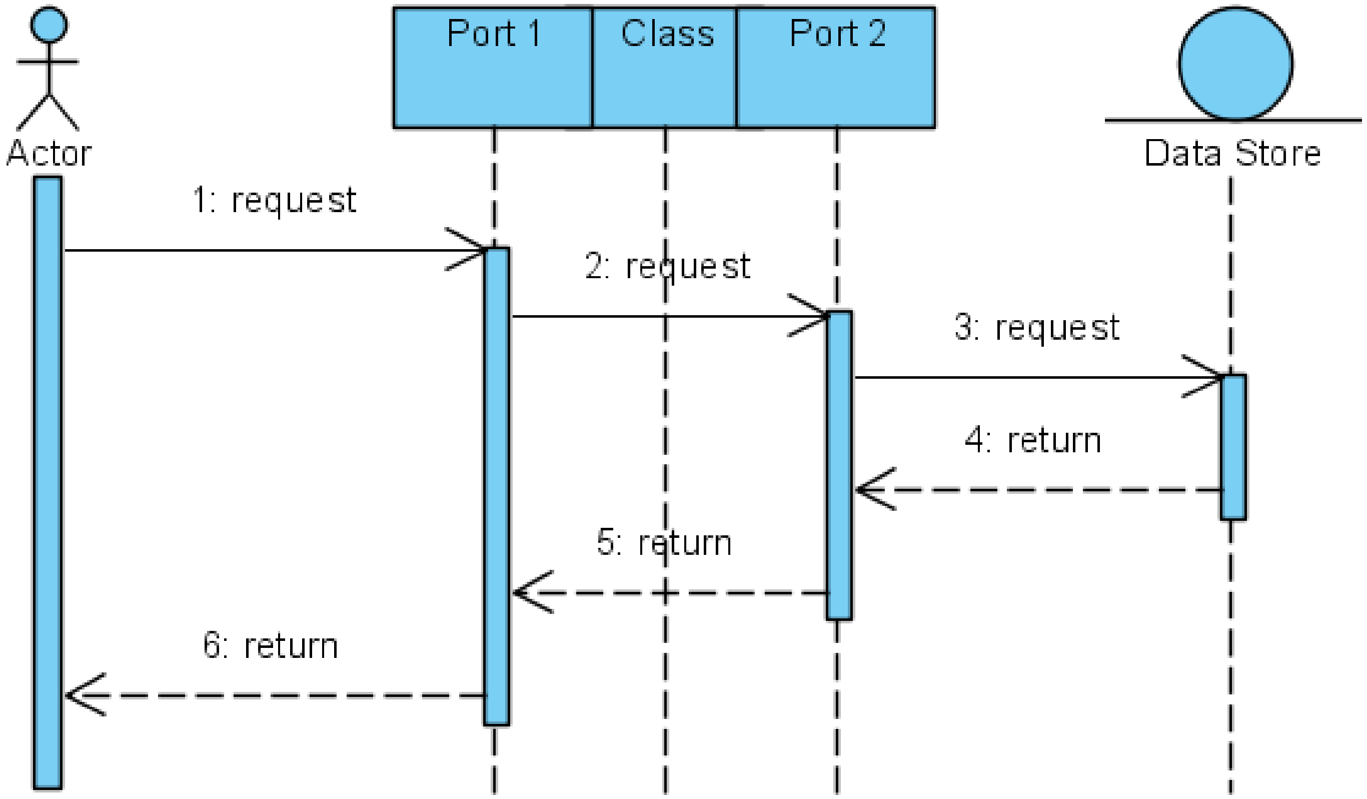


Sequence Diagrams - Gate



- A gate is a connection point for connecting a message inside a fragment with a message outside a fragment. EA shows a gate as a small square on a fragment frame.

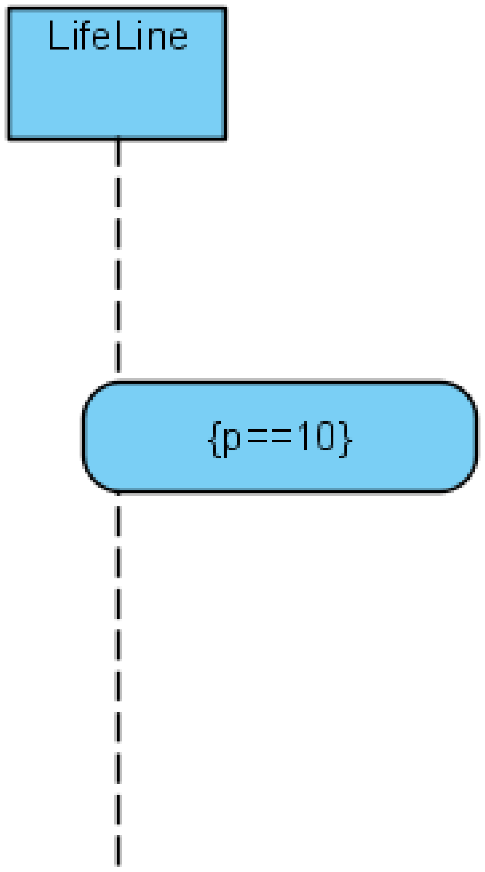
Sequence Diagrams – Part decomposition



- An object can have more than one lifeline coming from it. This allows for inter- and intra-object messages to be displayed on the same diagram.

Sequence Diagrams - State Invariant / Continuations

- A state invariant is a constraint placed on a lifeline that must be true at run-time. It is shown as a rectangle with semi-circular ends.
- A continuation has the same notation as a state invariant but is used in combined fragments and can stretch across more than one lifeline.



Sequence Diagrams – Tool Example

- Sequence diagrams re-use class diagrams
  - Create a class diagrams
    - Define the classes
  - Create a sequence diagrams
    - Right click on the sequence diagram and choose Select Class
    - When creating a message you may choose Select Operation
      - From the operation list it will show the operations (aka methods) the class supports

Outline

- Interaction diagrams
  - Sequence diagrams
  - Communication diagrams
- State Machine Diagrams

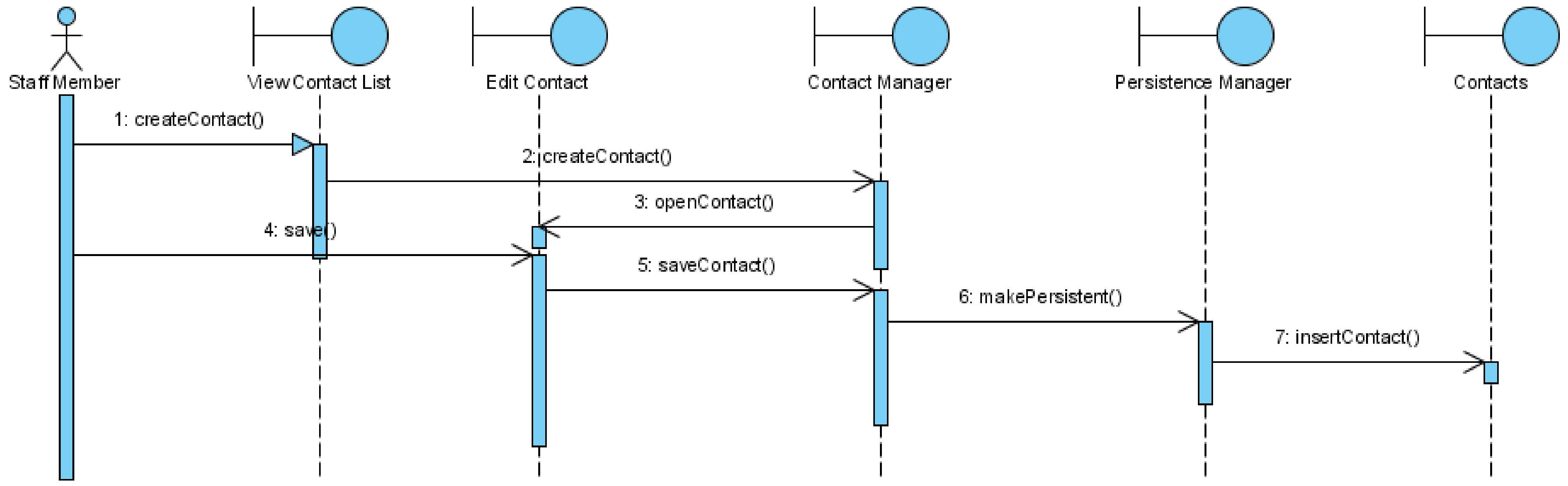
Communication Diagrams

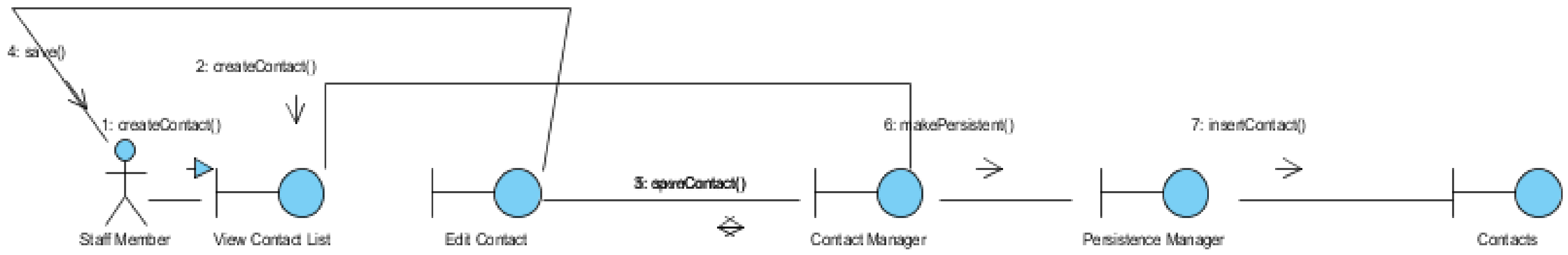
- Formerly called a collaboration diagram
- **Definition** : Interaction diagram that shows similar information to sequence diagrams but its primary focus is on object relationships.
- On communication diagrams, objects are shown with association connectors between them. Messages are added to the associations and show as short arrows pointing in the direction of the message flow. The sequence of messages is shown through a numbering scheme.

Communication Diagrams - Example

- The following two diagrams show a communication diagram and the sequence diagram that shows the same information.
- Although it is possible to derive the sequencing of messages in the communication diagram from the numbering scheme, it isn’t immediately visible.
- What the communication diagram does show quite clearly though is the full set of messages passed between adjacent objects.

Communication Diagrams - Example





Communication Diagram – Tool example

- How to derive a communication diagram from a sequence diagram
  - Synchronize Communication diagram
- It is also possible to derive a sequence diagram from a communication diagram
  - Synchronize Sequence diagram

Outline

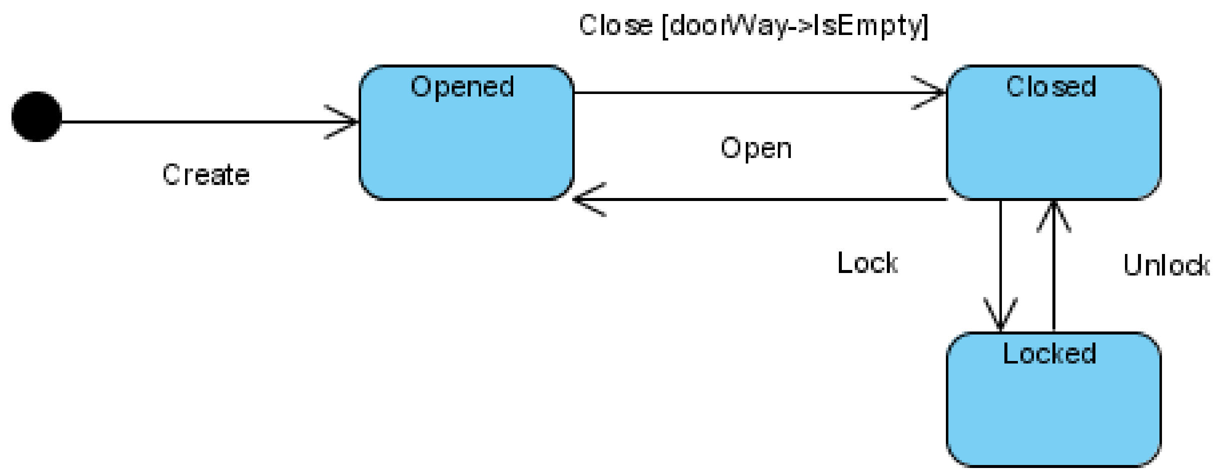
- Interaction diagrams
  - Sequence diagrams
  - Communication diagrams
- State Machine Diagrams

State Machine Diagrams

- **Definition** : A State Machine Diagram models the behavior of a single object, specifying the sequence of events that an object goes through during its lifetime in response to events.
- As an example, the following State Machine Diagram shows the states that a door goes through during its lifetime.

State Machine Diagrams - Example

- The door can be in one of three states:
  - Open, Closed, Locked
- It can respond to the events Open, Close, Lock and Unlock.
- Notice ALL events are valid.
  - The door can be locked until it is closed.
  - The door can be closed only if the doorway is empty

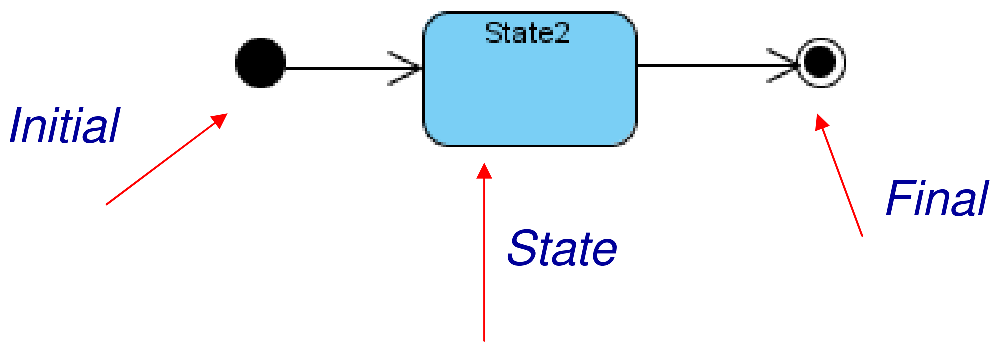


State Machine Diagrams

- States
  - Initial and final states
  - Transitions
  - State Actions
  - Self-transitions
- Compound states
- Entry Point
- Exit Point
- Choice Pseudo-State
- Junction Pseudo-State
- Terminate Pseudo-State
- History States
- Concurrent Regions

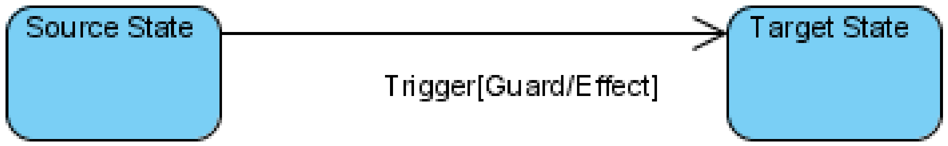
State Machine Diagrams

- A **State** is denoted by a round-cornered rectangle with the name of the state written inside it.
- The **Initial State** is denoted by a filled black circle and may be labeled with a name.
- The **Final State** is denoted by a circle with a dot inside and may also be labeled with a name.



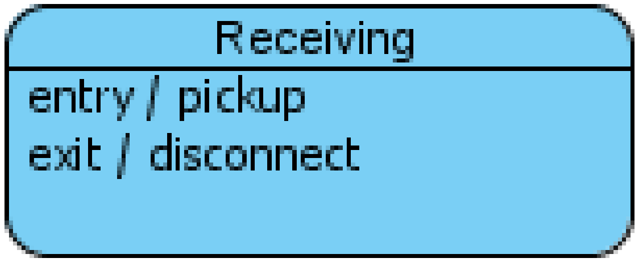
State Machine Diagrams - Transitions

- Transitions from one state to the next are denoted by lines with arrowheads.
- A transition may have a trigger, a guard and an effect, as below.
- **"Trigger"** is the cause of the transition.
- **"Guard"** is a condition which must be true in order for the trigger to cause the transition.
- **"Effect"** is an action which will be invoked directly on the object that owns the state machine as a result of the transition.

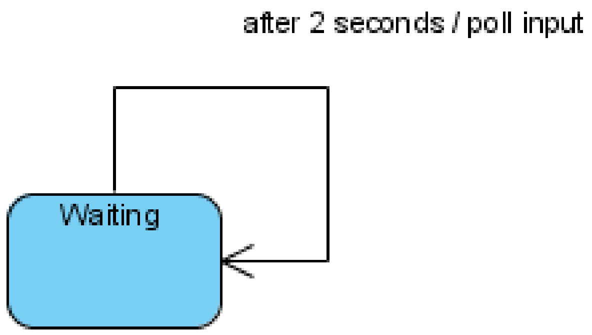


State Machine Diagrams – Entry and Exit

- In previous example, an Effect was associated with the transition.
- If the target state had many transitions arriving at it, and each transition had the same effect associated with it, it would be better to associate the effect with the target state rather than the transitions. This can be done by defining an entry action for the state. The diagram below shows a state with an entry action and an exit action.



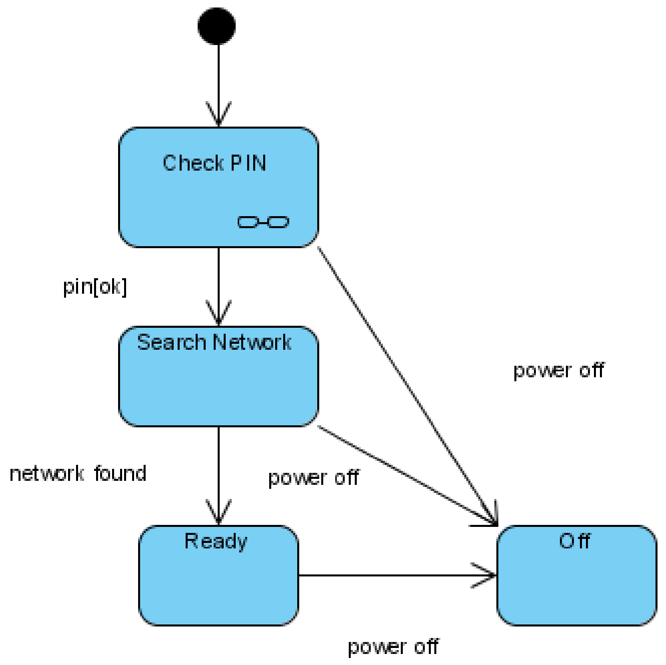
State Machine Diagrams – Self Transitions

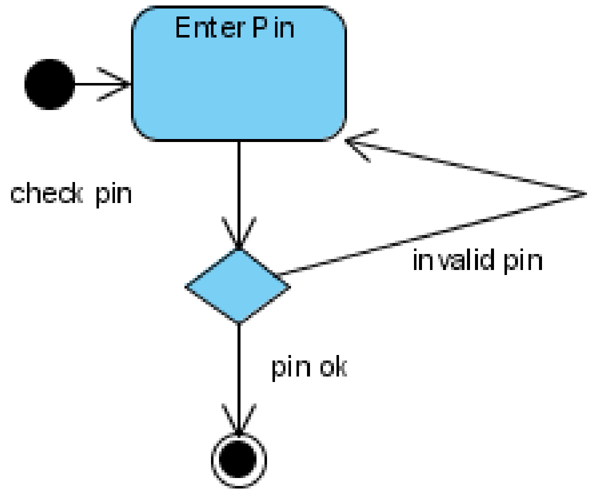


- A state can have a transition that returns to itself, as in the following diagram. This is most useful when an effect is associated with the transition.

State Machine Diagrams – Compound States

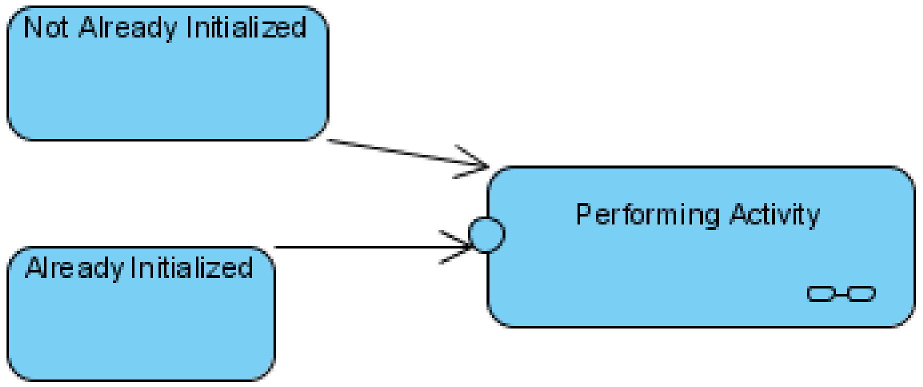
- A state machine diagram may include sub-machine diagrams, as in the example below.
- The notation in the above version indicates that the details of the Check PIN sub-machine are shown in a separate diagram.





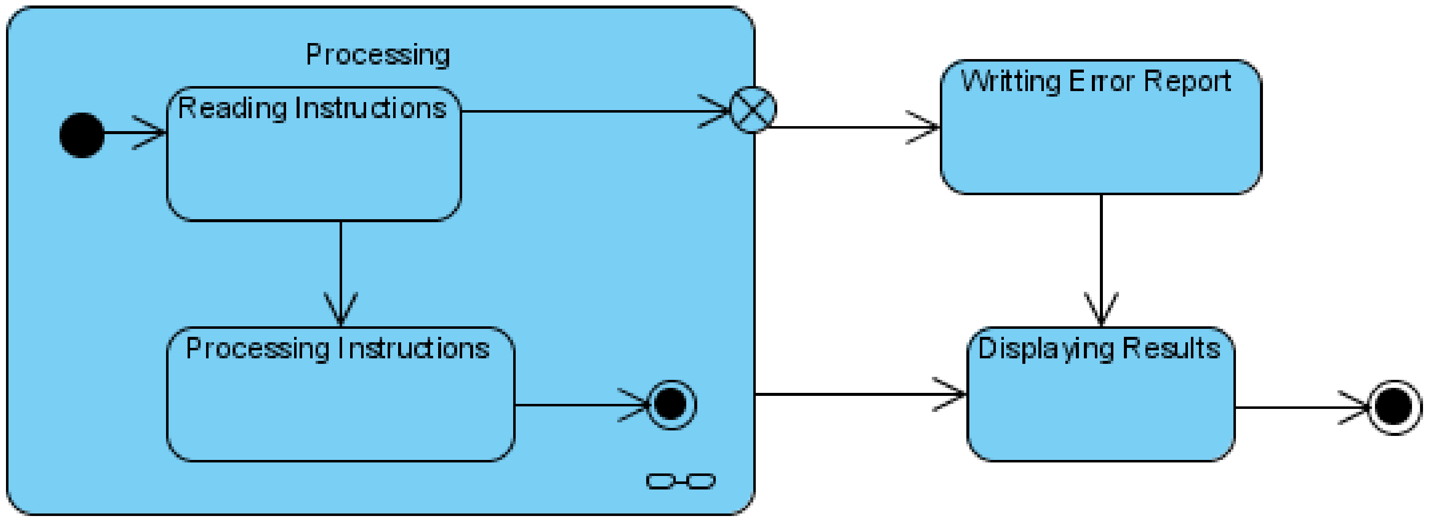
State Machine Diagrams – Entry Point

- Sometimes you won't want to enter a sub-machine at the normal Initial State. For example, in the following sub-machine it would be normal to begin in the Initializing state, but if for some reason it wasn't necessary to perform the initialization, it would be possible to begin in the Ready state by transitioning to the named Entry Point.



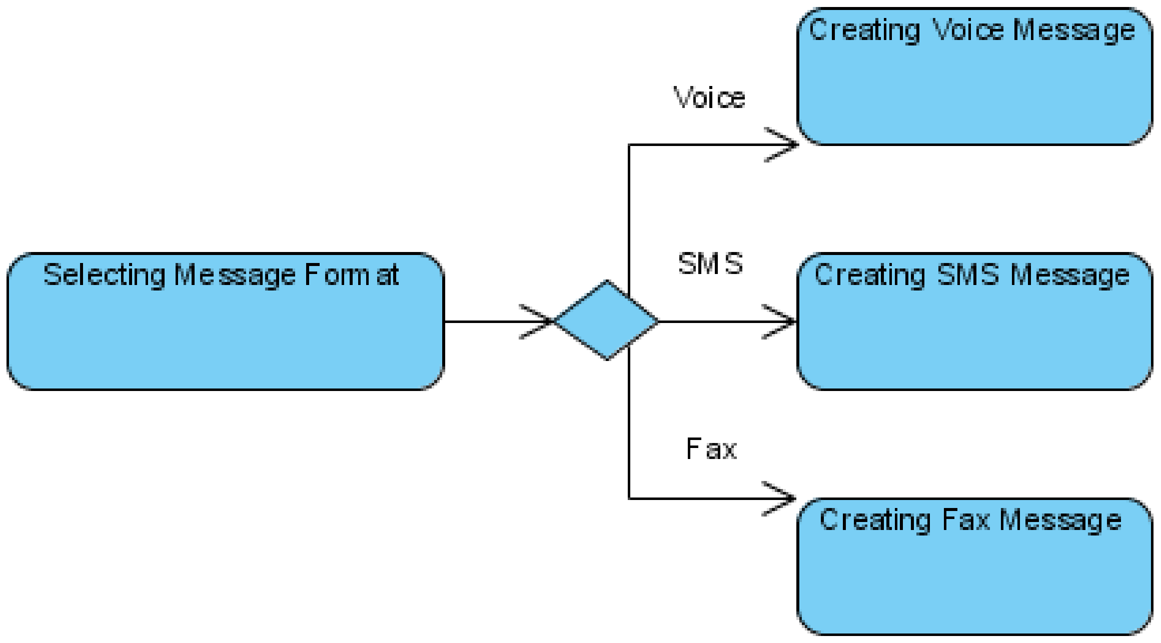
State Machine Diagrams – Exit Points

- In a similar manner to Entry Points, it is possible to have named alternative Exit Points. The following diagram gives an example where the state executed after the main processing state depends on which route is used to transition out of the state.



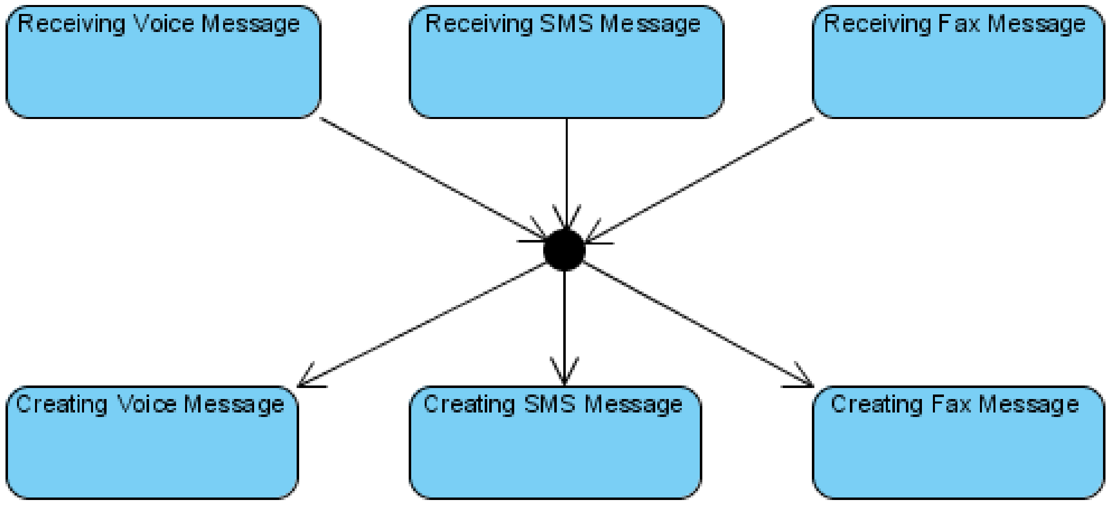
State Machine Diagrams – Choice Pseudo State

- A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving. The following diagram shows that whichever state is arrived at after the choice pseudo-state is dependent on the message format selected during execution of the previous state.



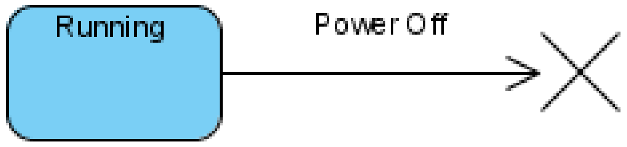
State Machine Diagrams – Junction Pseudo State

- Junction pseudo-states are used to chain together multiple transitions. A single junction can have one or more incoming and one or more outgoing transitions and a guard can be applied to each transition. Junctions are semantic-free; a junction which splits an incoming transition into multiple outgoing transitions realizes a static conditional branch as opposed to a choice pseudo-state which realizes a dynamic conditional branch.



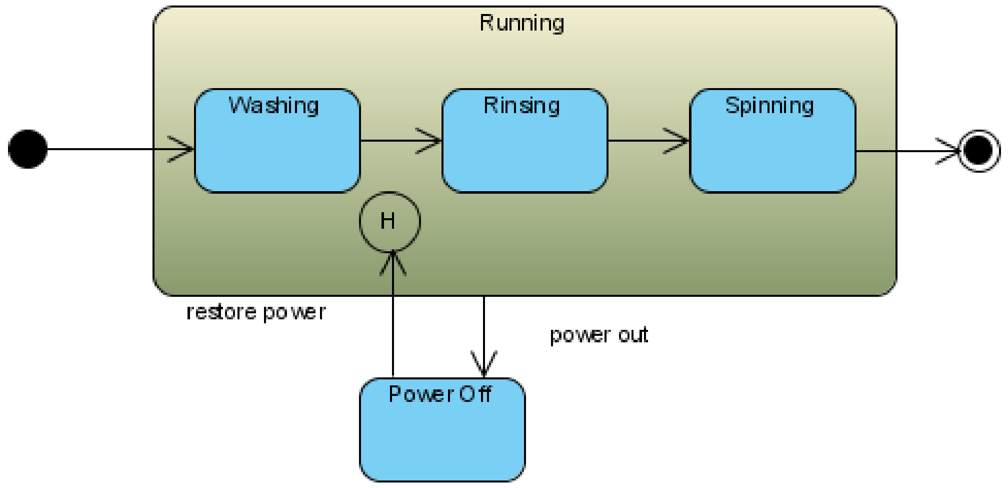
State Machine Diagrams – Terminate Pseudo State

- Entering a terminate pseudo-state indicates that the lifeline of the state machine has ended. A terminate pseudo-state is notated as a cross.



State Machine Diagrams – History State

- A History State is used to remember the previous state of a state machine when it was interrupted. The following diagram illustrates the use of history states. The example is a state machine belonging to a washing machine.
- In this state machine, when a washing machine is running it will progress from Washing through Rinsing to Spinning. If there is a power cut, the washing machine will stop running and will go to the Power Off state. Then when the power is restored, the Running state is entered at the History State symbol meaning that it should resume where it last left-off.

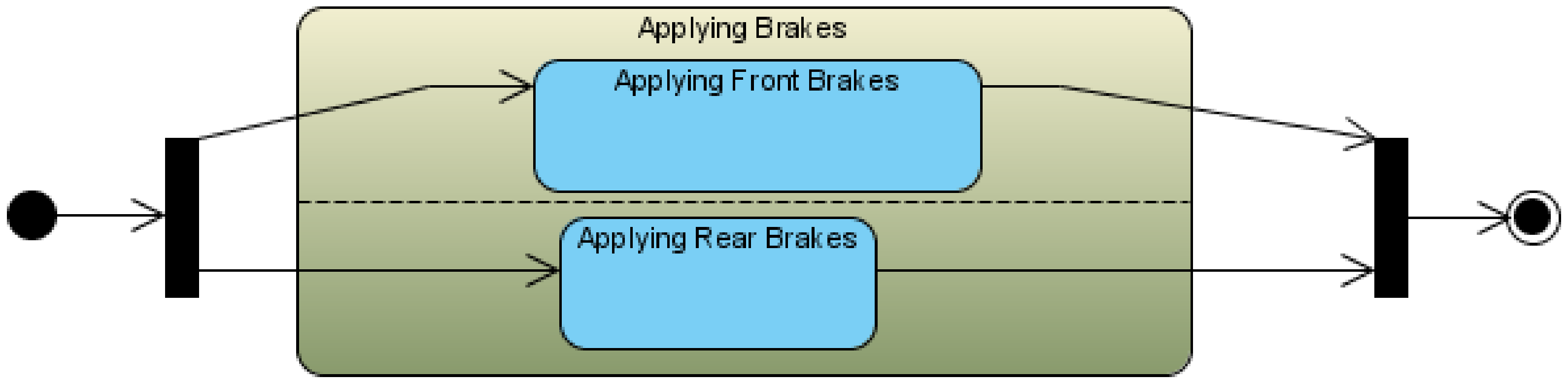


State Machine Diagrams – History State

- Shallow history is denoted by H in a circle, and means that when you transition into a hierarchical composite state, the history remembers just the top-level substate
- There is also deep history, denoted by H* is a circle, and this means that when you transition into a hierarchical composite state, the history remembers the lowest level substate and goes to that

State Machine Diagrams – Concurrent Regions

- A state may be divided into regions containing sub-states that exist and execute concurrently. The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently. Notice the use of fork and join pseudo-states rather than choice and merge pseudo-states. These symbols are used to synchronize the concurrent threads.



Questions

References

- Sequence Diagrams
  - http://www.sparxsystems.com/resources/uml2_tutorial/uml2_sequencediagram.html
  - http://www.agilemodeling.com/artifacts/sequenceDiagram.htm
  - http://www.visual-paradigm.com/VPGallery/diagrams/Sequence.html
- Communication Diagrams
  - http://sparxsystems.com.au/resources/uml2_tutorial/uml2_communicationdiagram.html
  - http://www.agilemodeling.com/artifacts/communicationDiagram.htm
  - http://www.visual-paradigm.com/VPGallery/diagrams/Collaboration.html
- State Machine Diagrams
  - http://sparxsystems.com.au/resources/uml2_tutorial/uml2_statediagram.html
  - http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm
  - http://www.visual-paradigm.com/VPGallery/diagrams/State.html
- UML 2.0 Tutorial
  - http://www.visual-paradigm.com/product/vpuml/tutorials/uml.jsp (Interactive)
  - http://isds.bus.lsu.edu/cvoc/learn/bpr/cprojects/spring1998/modeling/interaction.html