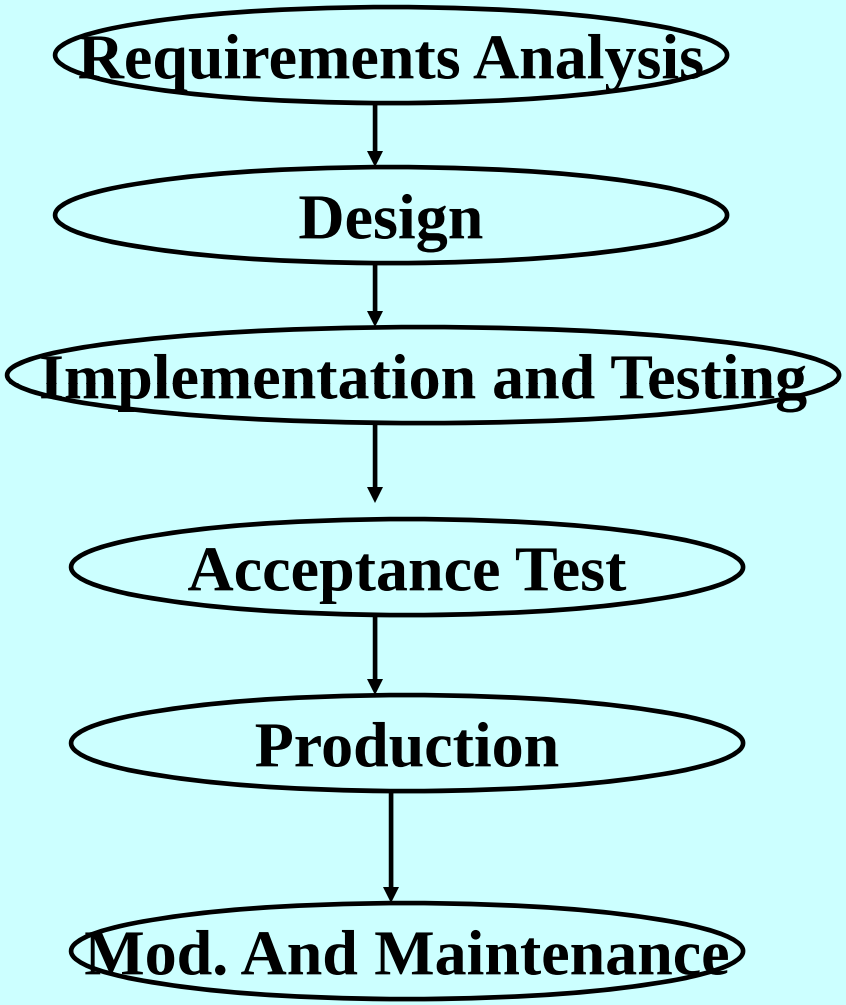
Modeling with UML

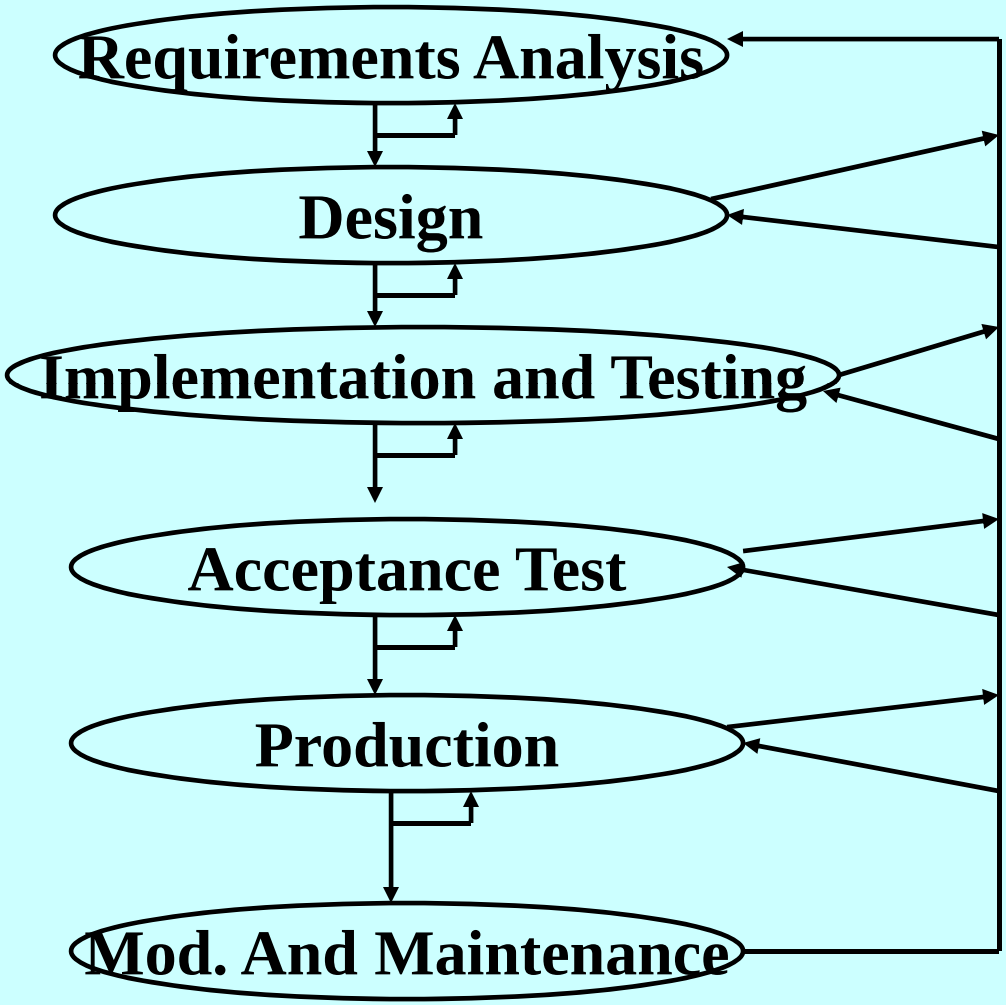
Software Life Cycle

- **Requirements Analysis**
- **Design**
- **Implementation and Testing**
- **Acceptance Test**
- **Production**
- **Modification and Maintenance**

Life cycle models: Waterfall model



Life cycle models: Spiral model



Modeling using UML

- Use Case Diagrams
- Class Diagrams
- Sequence diagrams
- State machine diagrams
- Activity diagrams

From book: Object-Oriented Software Engineering: Using UML, Patterns, and Java, by Bernd Bruegge and Allen H. Dutoit (Chapter 2)

Three different models:

- Functional model: describes the functionality of the system from user's point of view. (Use case diagrams)
- Object model: describes the structure of system in terms of objects, attributes, and operations and associations. (Class Diagrams)
- Dynamic model: describes the internal behavior of the system. (Sequence diagrams, state machine diagrams, activity diagrams).

Use Case Diagrams

- **used during requirements analysis phase.**

- **represent the functionality of the system from a user's point of view.**

- **defines the boundaries of the system.**

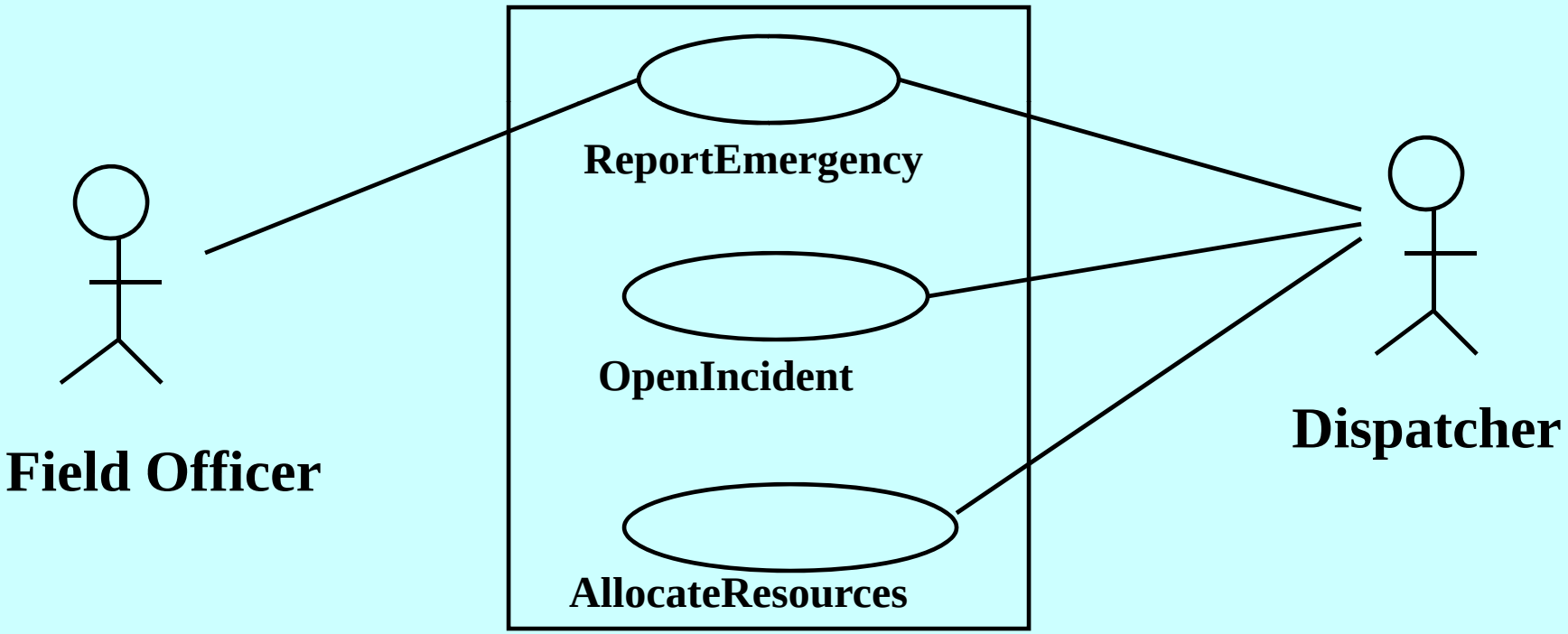
Use Case Diagrams (contd.)

Actors: external entities that interact with the system (e.g. a user, system admin, a bank customer…) or another system (central databse…).

Use cases: describe the behavior (external behavior) of the system as seen from an actor's point of view.

Use Case Diagrams (Example)

- **Accident Management System**



Template to describe a use case

- **Name of the use case**
- **Participating actors (actors interacting with the use case)**
- **Entry conditions: condns. that need to be satisfies before the use case is initiated.**
- **Flow of events (numbered): sequence of interactions of the use case.**
- **Exit conditions: condns. that need to be satisfies before the use case is initiated.**
- **Quality Reqts.: reqts. That are not related to the functionality of the system**

ReportEmergency Use case description

Use case name	`ReportEmergency`
Participating actors	Initiated by `FieldOfficer` Communicates with `Dispatcher`
Flow of events	1. The `FieldOfficer` activates the "Report Emergency" function of her terminal. 2. `FRIEND` responds by presenting a form to the `FieldOfficer`. 3. The `FieldOfficer` fills out the form by selecting the emergency level, type, location, and brief description of the situation. The `FieldOfficer` also describes possible responses to the emergency situation. Once the form is completed, the `FieldOfficer` submits the form. 4. `FRIEND` receives the form and notifies the `Dispatcher`. 5. The `Dispatcher` reviews the submitted information and creates an `Incident` in the database by invoking the `OpenIncident` use case. The `Dispatcher` selects a response and acknowledges the report. 6. `FRIEND` displays the acknowledgment and the selected response to the `FieldOfficer`.
Entry condition	• The `FieldOfficer` is logged into `FRIEND`.
Exit condition	• The `FieldOfficer` has received an acknowledgment and the selected response from the `Dispatcher`, OR • The `FieldOfficer` has received an explanation indicating why the transaction could not be processed.
Quality requirements	• The `FieldOfficer`'s report is acknowledged within 30 seconds. • The selected response arrives no later than 30 seconds after it is sent by the `Dispatcher`.

Scenarios

- **A use case is an abstraction that describes all possible scenarios involving the described functionality.**

- **Scenarios are used as examples for illustrating common cases: their focus is on understandibility.**

Template to describe a Scenario

- **Name of the scenario. The name is underlined.**
- **Participating actor instances (with underlined names)**
- **Flow of events (numbered): sequence of events step by step.**

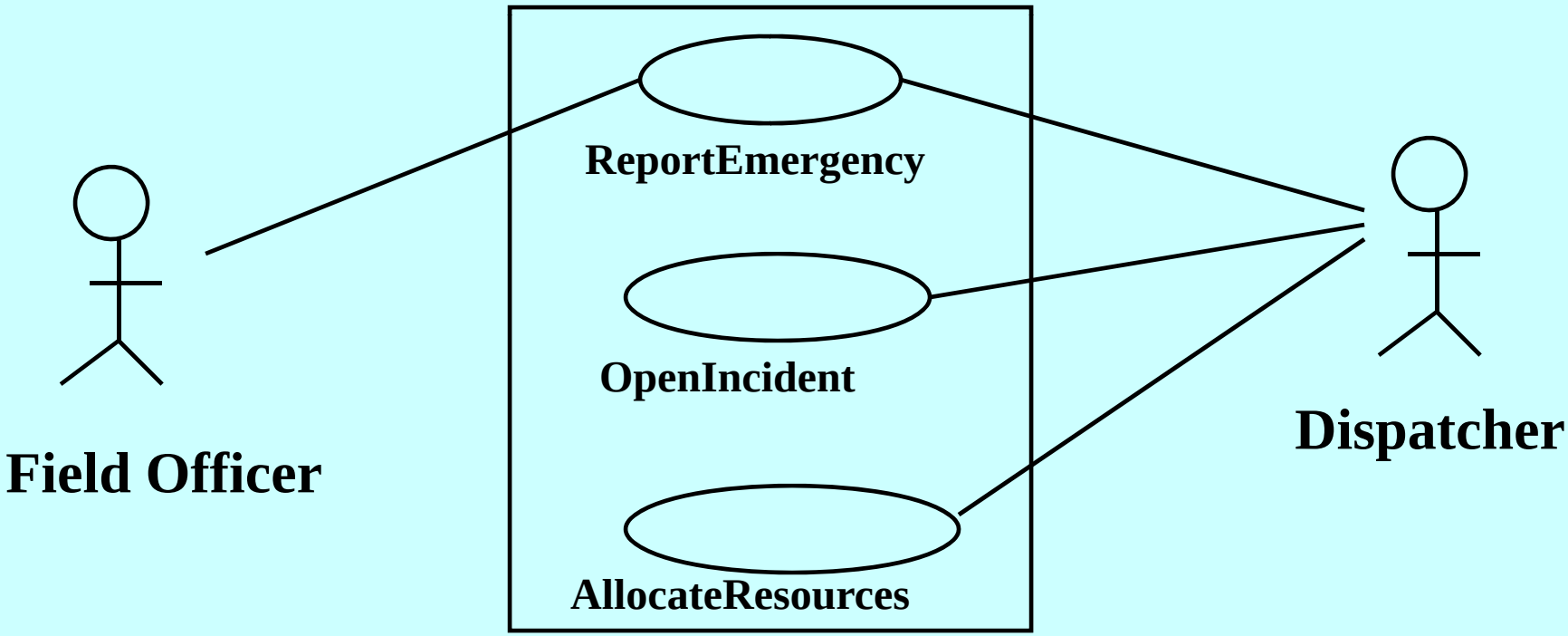
No Entry or Exit conditions

wareHouseOnFire scenario for ReportEmergency use case

Scenario name	warehouseOnFire
Participating actor instances	bob, alice:FieldOfficer john:Dispatcher
Flow of events	1. Bob, driving down main street in his patrol car, notices smoke coming out of a warehouse. His partner, Alice, activates the "Report Emergency" function from her FRIEND laptop. 2. Alice enters the address of the building, a brief description of its location (i.e., northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appears to be relatively busy. She confirms her input and waits for an acknowledgment. 3. John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice. 4. Alice receives the acknowledgment and the ETA.

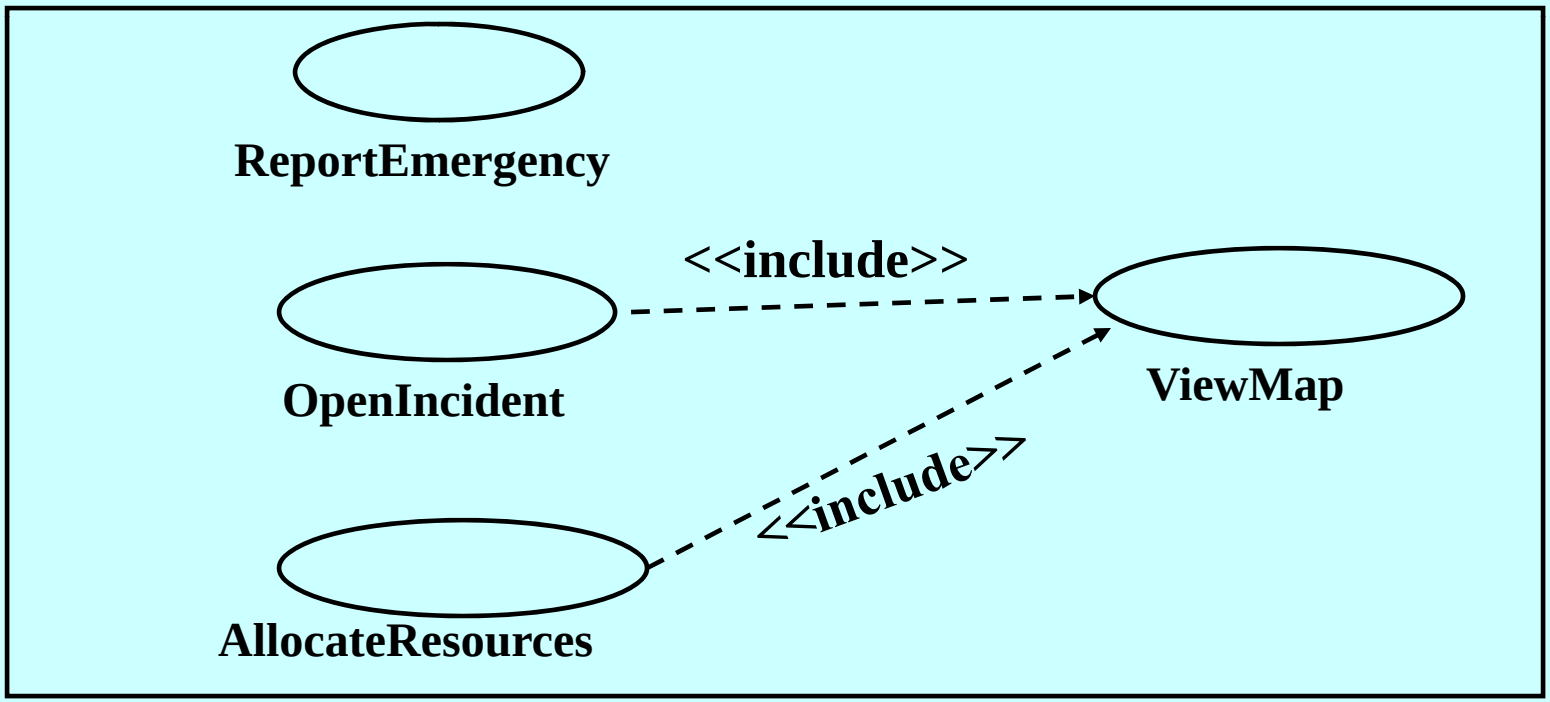
Use case diagrams: four types of relationships

- **Association: Actors and use cases communicate when information is exchanged between them. Depicted by solid line between actor and the use case.**



Use case diagrams: four types of relationships

- **Include: Depicted by dashed open arrow (labeled with <<include>> originating from the including use case.**



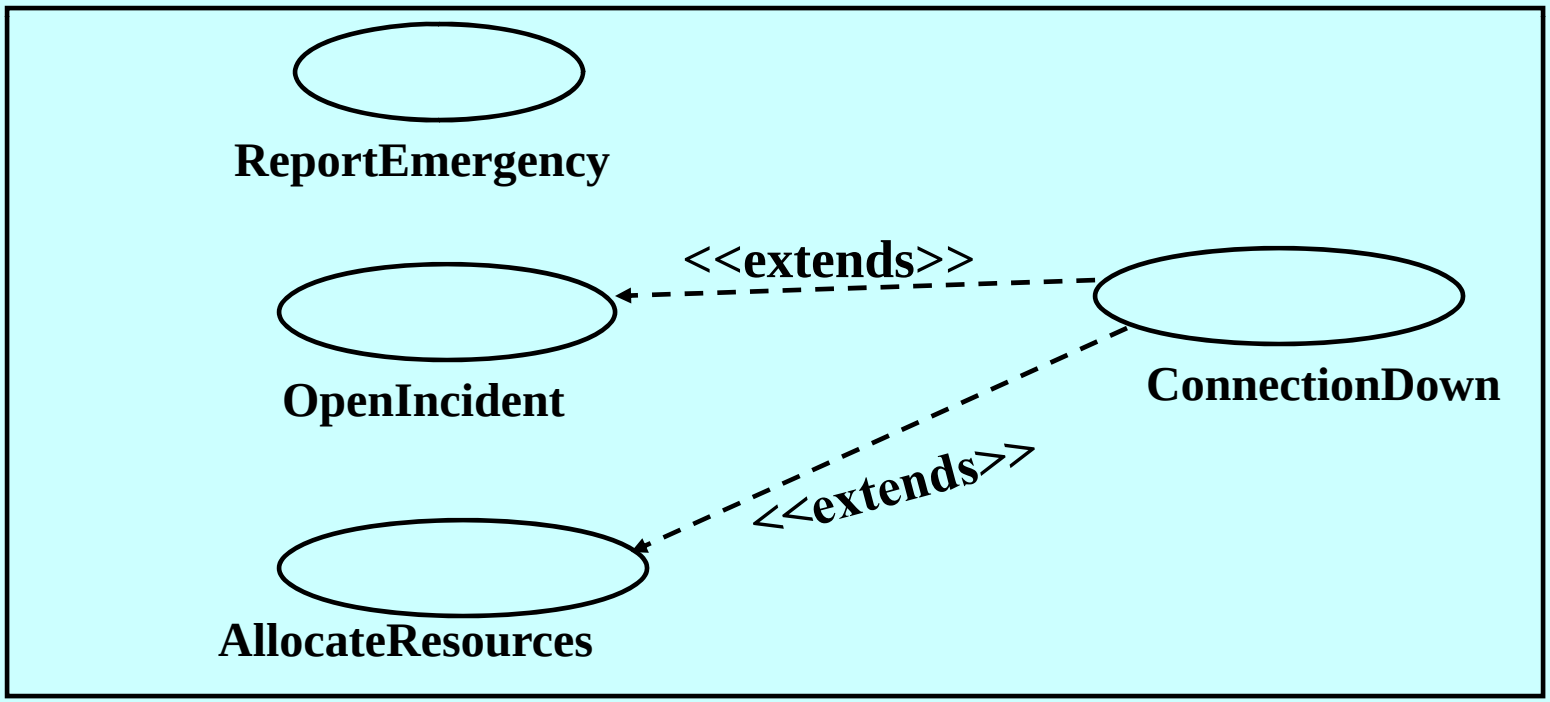
Textual description of include relationships

Figure 2-16 An example of an «include» relationship (UML use case diagram).

Use case name	AllocateResources
Participating actor	Initiated by Dispatcher
Flow of events	. . .
Entry condition	• The Dispatcher opens an Incident.
Exit condition	• Additional Resources are assigned to the Incident. • Resources receives notice about their new assignment. • FieldOfficer in charge of the Incident receives notice about the new Resources.
Quality requirements	At any point during the flow of events, this use case can **include** the ViewMap use case. The ViewMap use case is initiated when the Dispatcher invokes the "map" function. When invoked within this use case, the system scrolls the map so that location of the current Incident is visible to the Dispatcher.

Use case diagrams: four types of relationships

- **Extends: A use case can extend another use case by adding events.**



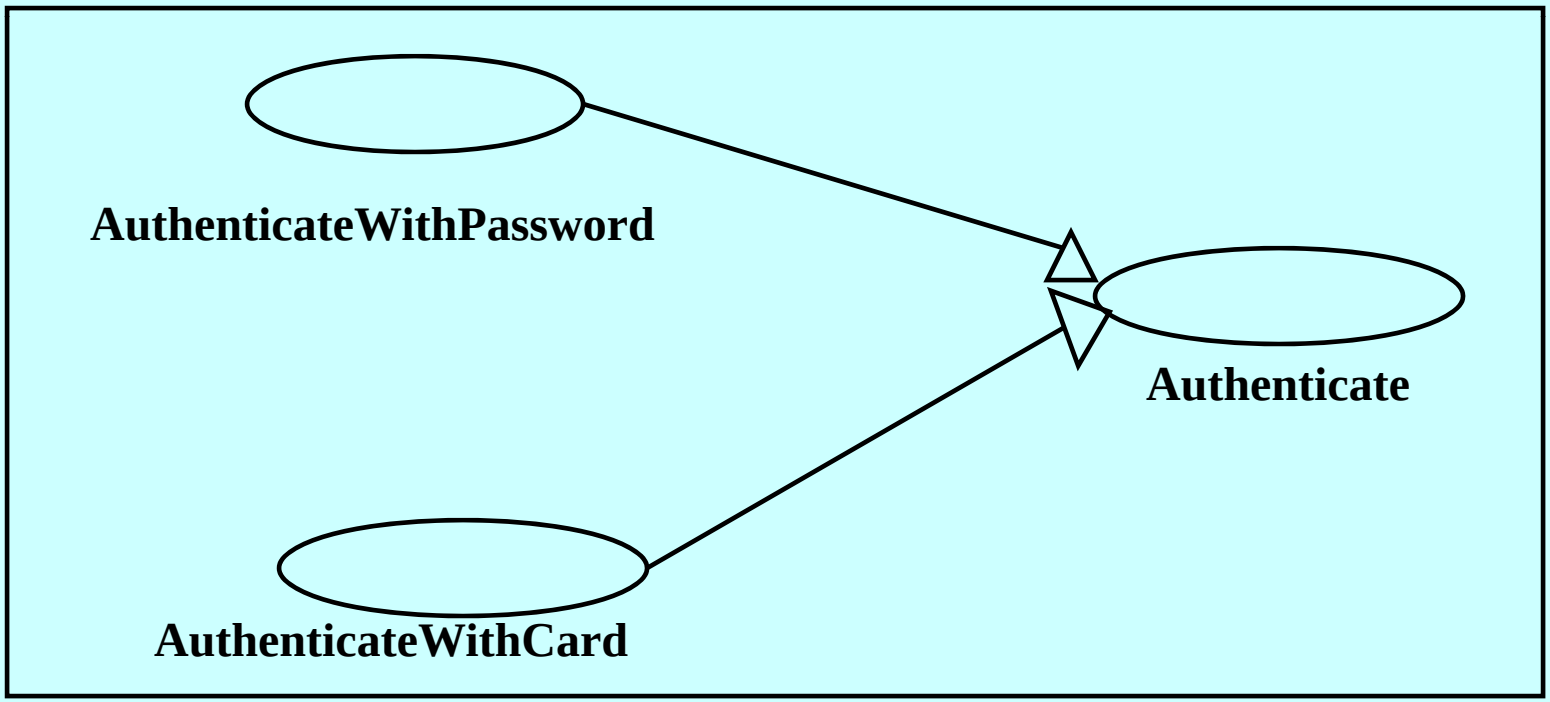
Textual description of extend relationships

Figure 2-18 An example of an «extend» relationship (UML use case diagram).

Use case name	`ConnectionDown`
Participating actor	Communicates with `FieldOfficer` and `Dispatcher`.
Flow of events	. . .
Entry condition	This use case **extends** the `OpenIncident` and the `AllocateResources` use cases. It is initiated by the system whenever the network connection between the `FieldOfficer` and `Dispatcher` is lost.
Exit condition	. . .

Use case diagrams: four types of relationships

- **Inheritance/Generalization: A use case can specialize another more general one by adding more detail.**



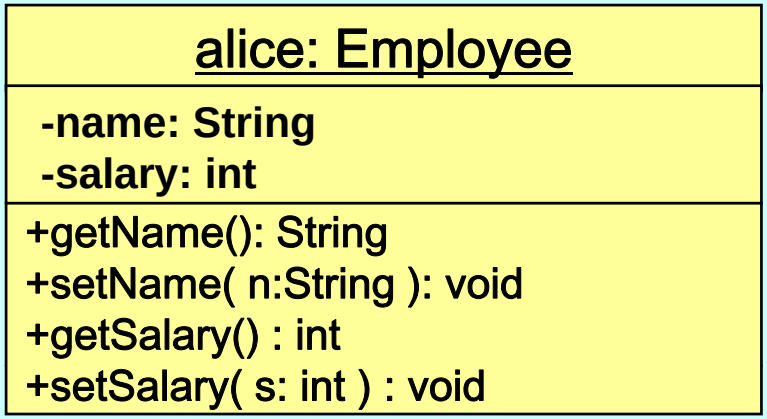
Textual description of inheritance relationships

Use case name	`AuthenticateWithCard`
Participating actor	**Inherited** from Authenticate use case.
Flow of events	1. The `BankCustomer` inserts her card into the ATM. 2. The `ATM` acknowledges the card and prompts the actor for her personal identification number (PIN). 3. The `BankCustomer` enters her PIN with the numeric keypad. 4. The `ATM` checks the entered PIN against the PIN stored on the card. If the PINs match, the `BankCustomer` is authenticated. Otherwise, the `ATM` rejects the authentication attempt.
Entry condition	**Inherited** from `Authenticate` use case.
Exit condition	**Inherited** from `Authenticate` use case.

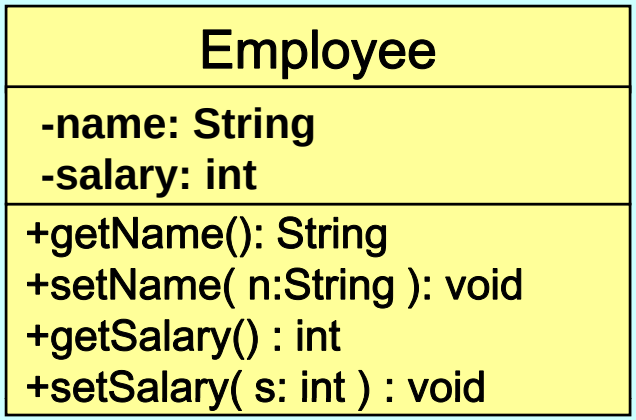
Class Diagrams

- **describe the structure of the system in terms of classes and objects.**

- **Relationships:**
  - **Aggregation**
  - **Inheritance**
  - **Dependency**



Object



Class

SequenceDiagrams

• describe patterns of communication among a set of interacting objects.

• An object interacts with another object by sending messages. The reception of a message by an object triggers the execution of a method, which may in turn send messages to other objects.

• represent the objects participating in the interaction horizontally and time vertically.

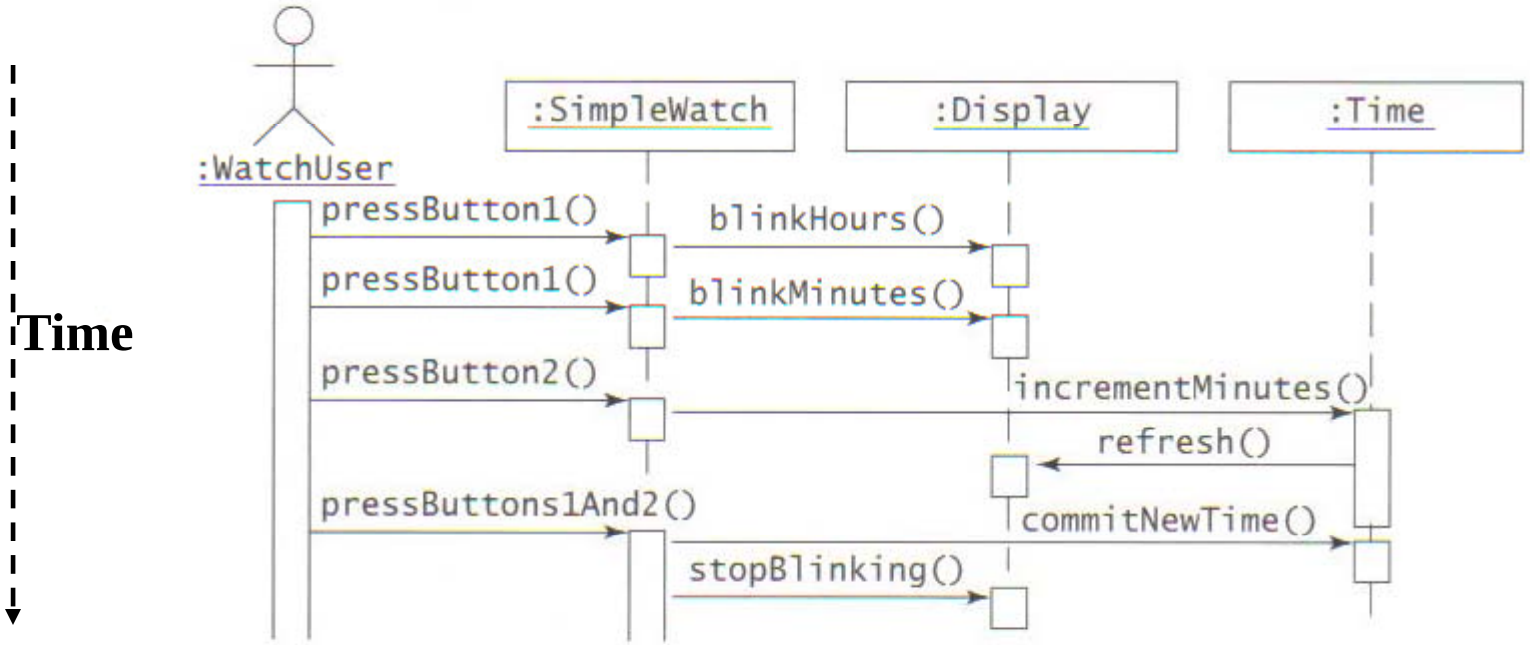


Figure 2-3 A UML sequence diagram for the `SimpleWatch`. The left-most column represents the timeline of the `WatchUser` actor who initiates the use case. The other columns represent the timeline of the objects that participate in this use case. Object names are underlined to denote that they are instances (as opposed to classes). Labeled arrows are stimuli that an actor or an object sends to other objects.

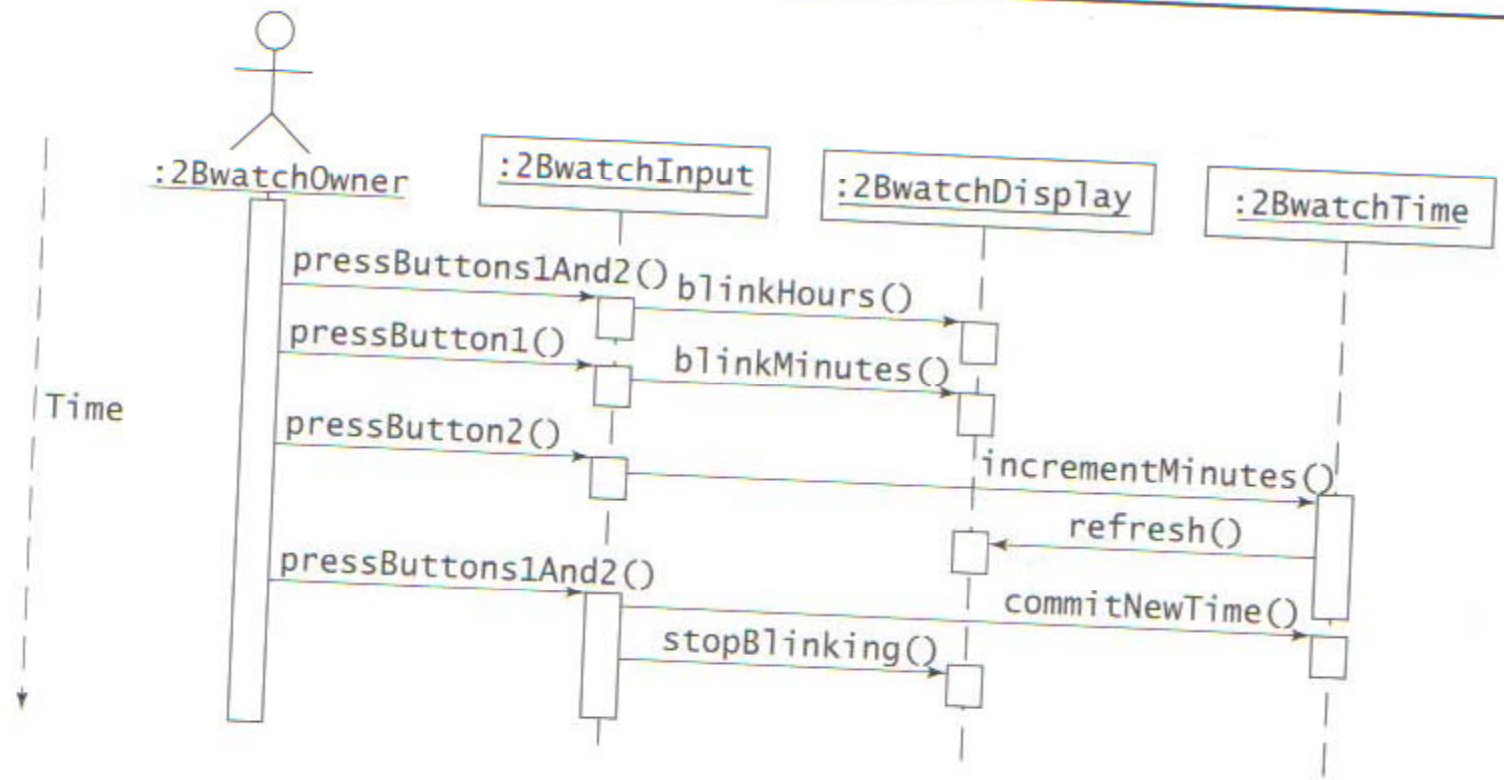


Figure 2-34 Example of a sequence diagram: setting the time on 2Bwatch.

State Machine Diagrams

- **describes a sequence of states an object goes through in response to external events.**

- **State: condn. satisfied by the attributes of an object.**
- **Transition: represents a change of state triggered by events, conditions or time.**
- **Actions: small atomic behaviors that are executed at specific points in the state machine. They take a short amount of time to execute and cannot be interrupted.**

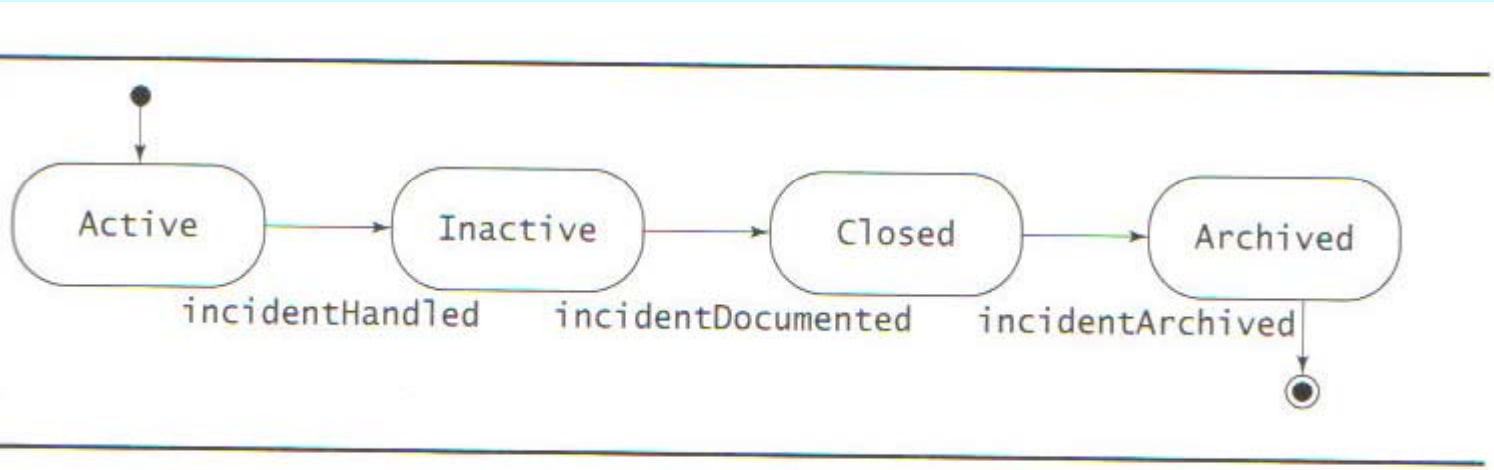


Figure 2-37 A UML statechart diagram for the `Incident` class.

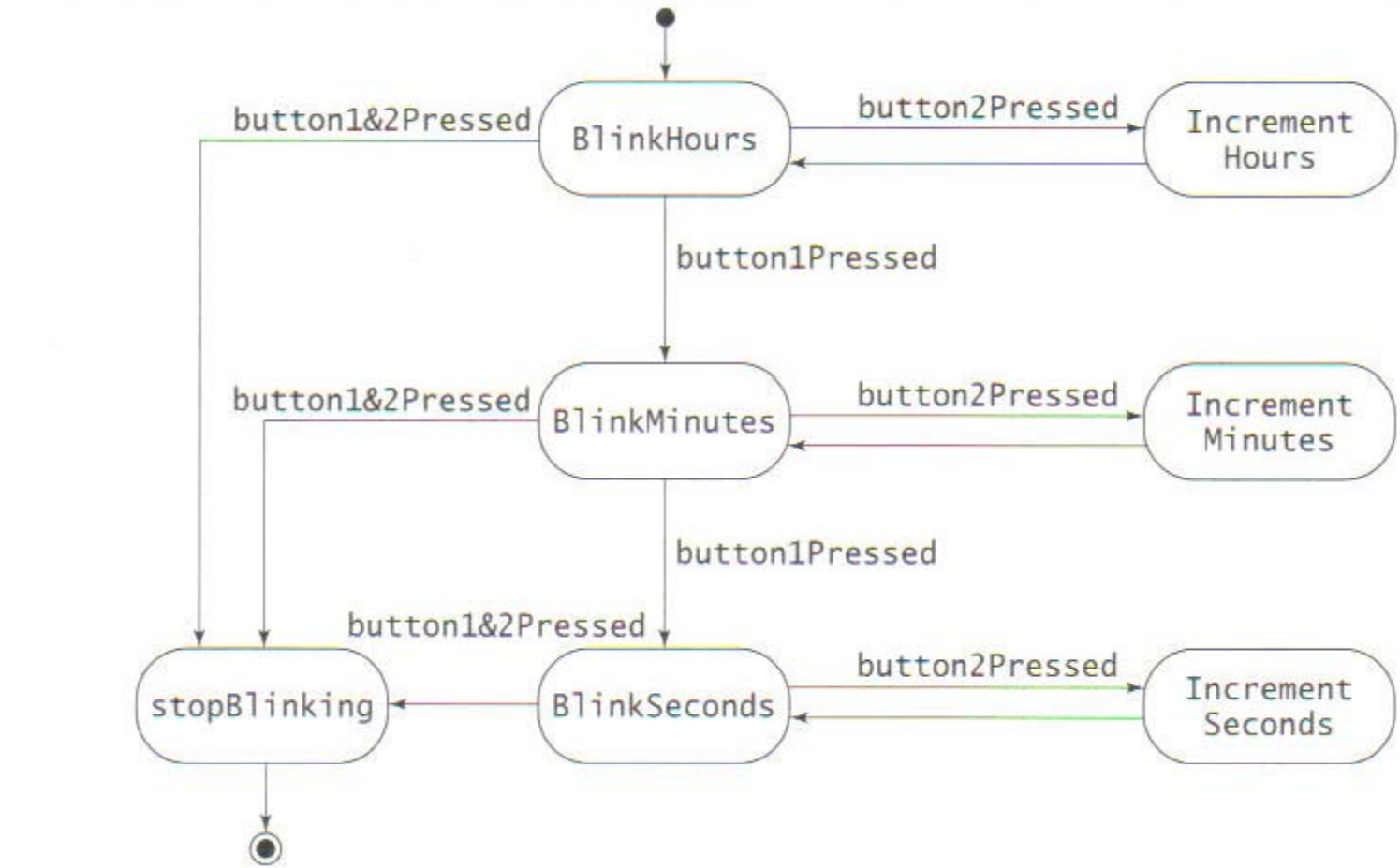


Figure 2-4 A UML statechart diagram for `SetTime` use case of the `SimpleWatch`.

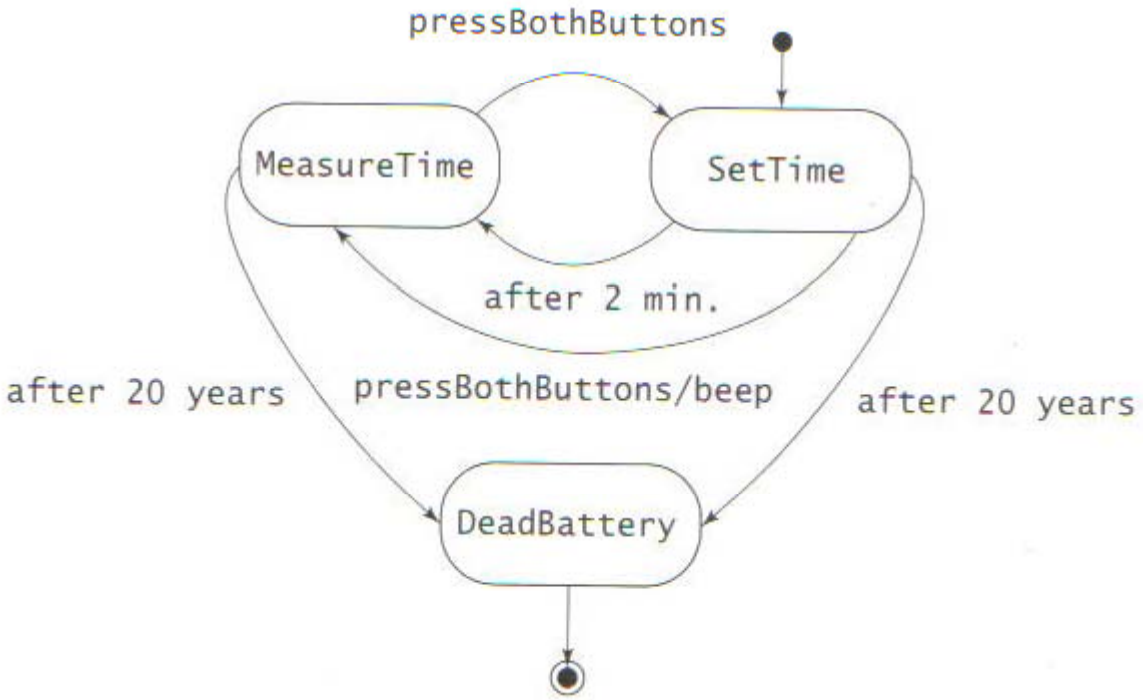


Figure 2-38 Statechart diagram for 2Bwatch set time function.

State Machine Diagrams

•Actions: small atomic behaviors that are executed at specific points in the state machine. They take a short amount of time to execute and cannot be interrupted.

- **Can occur in 3 places:**
  - **when a transition is taken**
  - **when a state is entered**
  - **when a state is exited.**

State Machine Diagrams

•Activity: is a behavior that is executed as long as an object resides in some state.

• an activity can take a substantial amount of time and interrupted when a transition exiting the state is fired.

• represented with the do label and are placed inside the state where they are executed.

Activity Diagrams

- **a task-centric view of the behavior of a set of objects.**
- **contains action states. Each action state corresponds to an activity in that state.**
- **open arrows are interpreted as sequential constraints between activities.**

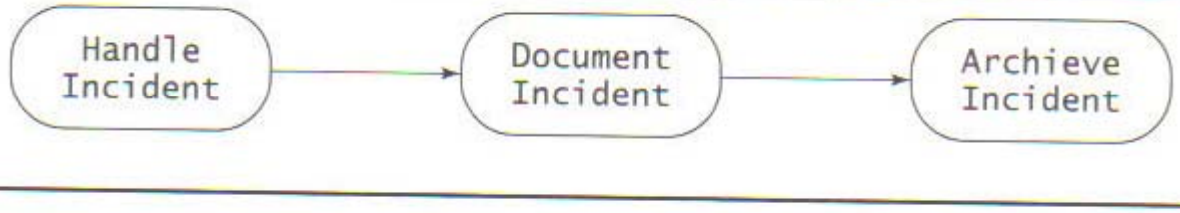


Figure 2-41 A UML activity diagram for `Incident`. During the action state `HandleIncident`, the `Dispatcher` receives reports and allocates resources. Once the `Incident` is closed, the `Incident` moves to the `DocumentIncident` activity during which all participating `FieldOfficers` and `Dispatchers` document the `Incident`. Finally, the `ArchiveIncident` activity represents the archival of the `Incident` related information onto slow access medium.

Activity Diagrams

- **Decision: branches in the control flow.**
  - **depicted by a diamond with one or more incoming open arrows and two or more outgoing arrows.**

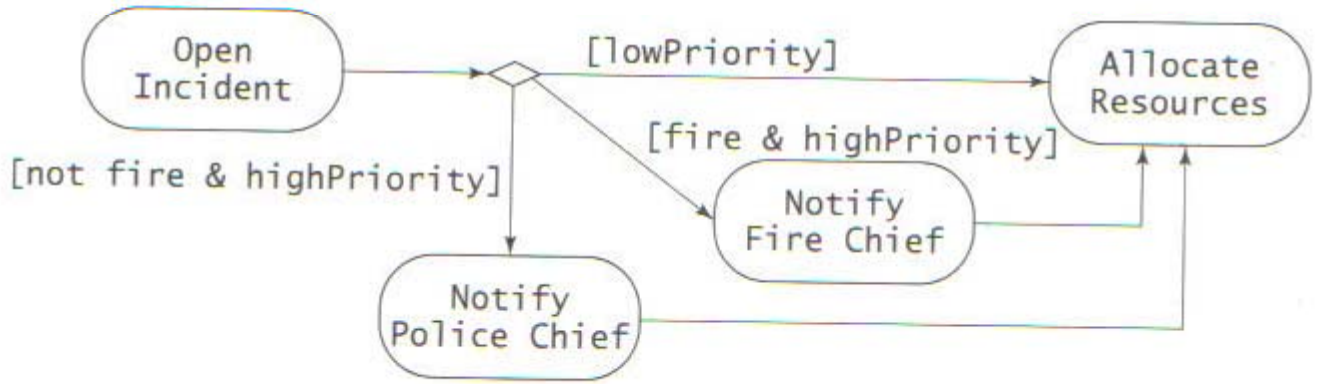


Figure 2-42 Example of decision in the `OpenIncident` process. If the `Incident` is a fire and is of high priority, the `Dispatcher` notifies the `FireChief`. If it is a high-priority `Incident` that is not a fire, the `PoliceChief` is notified. In all cases, the `Dispatcher` allocates resources to deal with the `Incident`.

Activity Diagrams

• **Complex transitions: transitions with multiple source states or multiple target states.**

- denotes the synchronization of multiple concurrent activities. (in case of multiple sources)

- denotes splitting of flow of control into multiple threads. (in case of multiple targets)

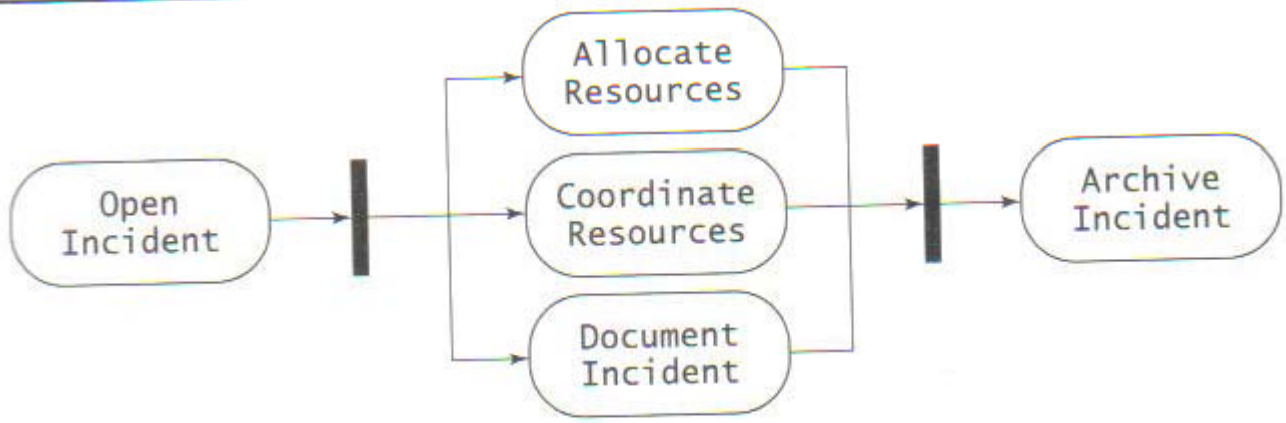


Figure 2-43 An example of complex transitions in a UML activity diagram.

Activity Diagrams

- **Swimlanes:actions may be grouped into swimlanes to denote object or subsystem that implements the actions.**
  - **represented as rectangles enclosing a group of actions.**
  - **transitions may cross swimlanes.**

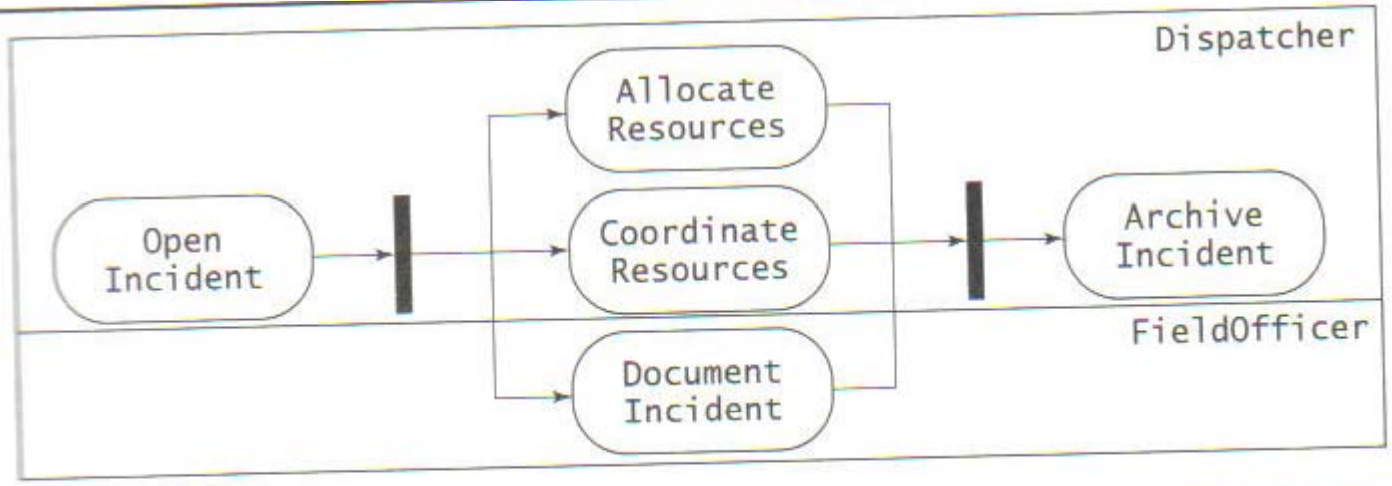


Figure 2-44 An example of swimlanes in a UML activity diagram.