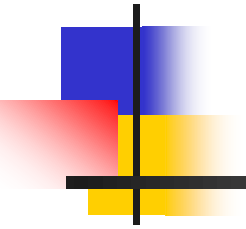


Use Cases

Object-Oriented Modeling and Design with UML (Second Edition)

Blaha & Rumbaugh

Sections 7.1, 8.1

Use Case Diagrams

Use case:

- A written description of one use of the system.
  - Who will use the system?
  - What will they be able to do with it?
  - No standard for format and content.

- Use case *diagram*
  - A graphical representation of a use case.
  - A diagram type defined in UML.

Use Case Diagrams

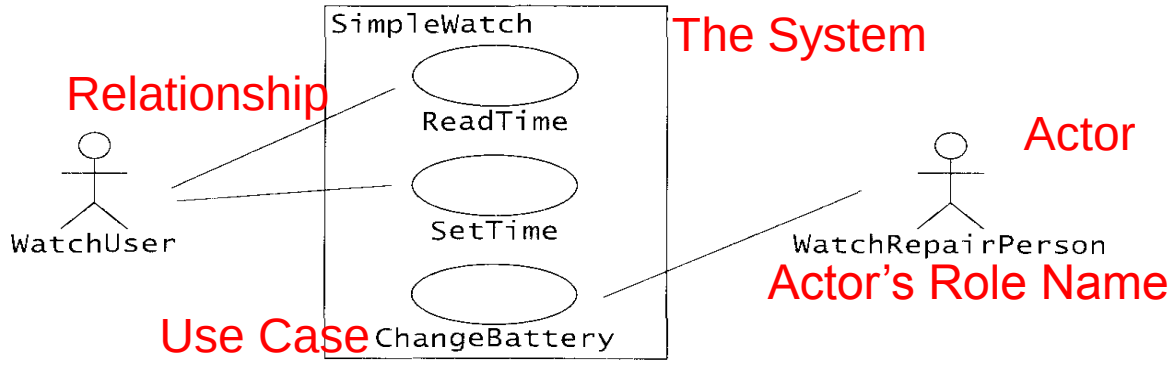


Figure 2-1 A UML use case diagram describing the functionality of a simple watch. The `WatchUser` actor may either consult the time on her watch (with the `ReadTime` use case) or set the time (with the `SetTime` use case). However, only the `WatchRepairPerson` actor can change the battery of the watch (with the `ChangeBattery` use case). Actors are represented with stick figures, use cases with ovals, and the boundary of the system with a box enclosing the use cases.

Componenets of a use case diagram:

- System
- Actors
- Use Cases
- Relationships

From *Object-Oriented Software Engineering using UML, Patterns, and Java*, Third Edition, by Bernd Bruegge and Allen H. Dutoit Prentice Hall/Pearson, 2010, page 31.

Use Case Example

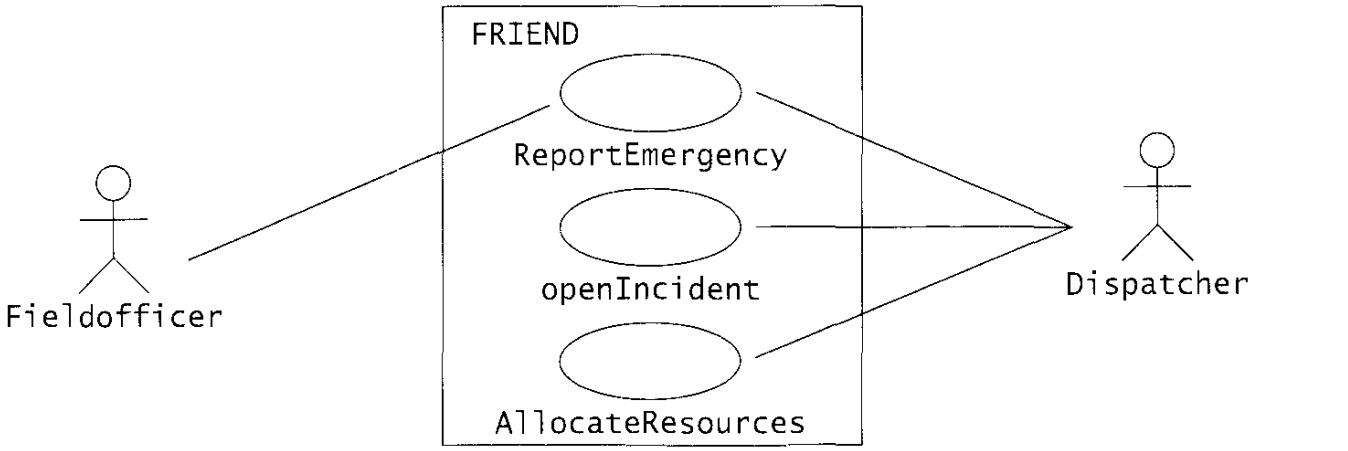


Figure 2-13 An example of a UML use case diagram for First Responder Interactive Emergency Navigational Database (`FRIEND`), an accident management system. Associations between actors and use cases denote information flows. These associations are bidirectional: they can represent the actor initiating a use case (`FieldOfficer` initiates `ReportEmergency`) or a use case providing information to an actor (`ReportEmergency` notifies `Dispatcher`). The box around the use cases represents the system boundary.

From *Object-Oriented Software Engineering using UML, Patterns, and Java*, Third Edition, by Bernd Bruegge and Allen H. Dutoit Prentice Hall/Pearson, 2010, page 44.

Use Case Diagrams

- Use case diagrams give a *high level* view of the system.
  - Outline
  - Table of Contents
- *Starting point* for saying what the system does.
  - Don't try to say too much.
  - Details will come later.

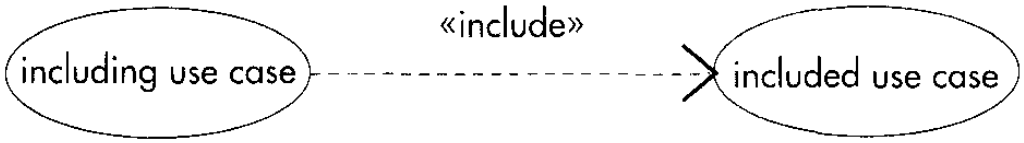
Generalization in Use Cases

Use case diagrams can show relationships among different use cases.

- Trade Bonds use case "is a" Make Trade use case.
- Generalization is used to indicate inheritance.
  - Factor out common behavior in related use cases.

Blaha and Rumbaugh, page 149.

The Include Relationship for Use Cases

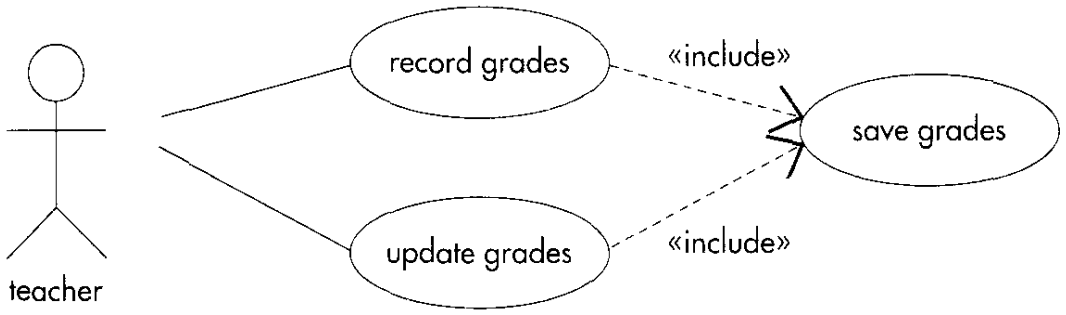


- "Subroutine" for use cases.
- Note "guillemets" around the word include.
- Means that the included use case is *always* a part of the including use case.

The Include Relationship for Use Cases

- Use the Include relationship when the same functionality appears in logically unrelated use cases.
  - Factor out common functionality.
  - Should be a well defined, meaningful activity.

The Include Relationship for Use Cases



- A teacher will record grades and update grades.
- In either case, grades will *always* be saved.
- Only describe "save grades" in one place.
- Indicates potential for factoring out and reusing functionality in the system that will be developed

The Extend Relationship for Use Cases

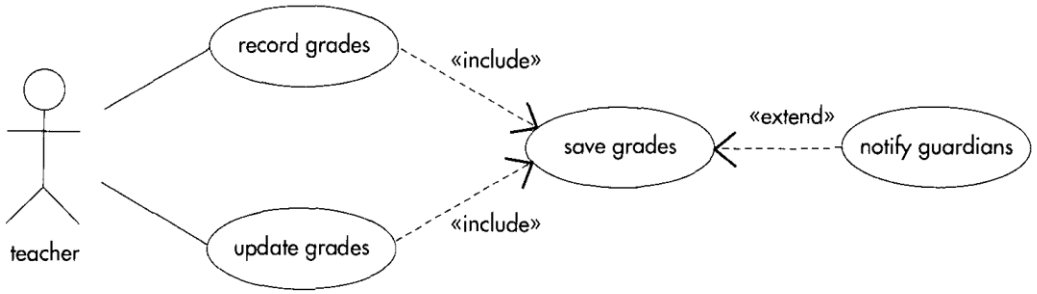
This means that the extending use case *may* be a part of the extended use case.

- Compare to *always* for the include relationship.

Note direction of arrow.

Use Extend for functionality that only applies in exceptional cases.

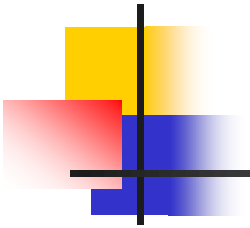
The Extend Relationship for Use Cases



- When the teacher either records grades or updates grades, the grades are always saved and guardians *may* be notified.

Includes vs. Extends

- By analogy with programming languages
  - Include is like a function
    - Must be explicitly "called" in the including use case.
    - Normally called from more than one place
      - multiple use cases.
  - Extend is like an exception
    - "Just happens"
    - Not mentioned in the extended use case.
    - Might be possible anywhere in the extended use case.
    - Might be possible only at a given point or range in the extended use case



How to Model Use Cases

- Identify the actors and use cases
- Prioritize the use cases.
- Detail each use case.
- Structure the case model.
- Prototype user interfaces.

Identify the Actors and Use Cases

- Talk to the stakeholders.
  - Product Manager
  - Customers
  - Users

Questions for Identifying Actors

- Who will use the system in their work?
- Who will perform administrative or maintenance functions?
- What external hardware or software will the system interact with?

Identifying Actors

- Who/What is an actor vs. part of the system?
- May depend on where you draw the boundary
- Example: Database System

Once actors are identified

Determine functionality accessible to each actor

Scenarios

- Narrative description of somebody doing something with the system.
- Specific, not abstract

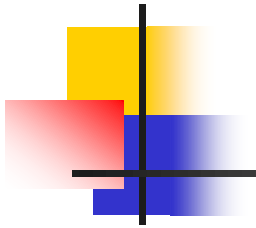
- Think of a movie scenario.
  - Describes exactly what happens thoughout a scene.

Scenario Example

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 the 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.

Bruegge and Dutoit, page 133

Use Cases



- What is the difference between a scenario and a use case?
- A scenario is *an instance of* a use case.
  - Scenario describes a *specific* case of someone making use of some piece of functionality.
  - Use case describes the functionality in general terms.
    - Covers all possible scenarios for this functionality.

Where do we start?

- Suggestion:
  - Sketch use case diagrams.
  - Write scenarios
  - Derive use case descriptions
  - Iterate

Detail each use case.

- Describe interactions in detail.
  - Remember: Still from external point of view
- Look for gaps and inconsistencies.
- Dig for details.
- May need to do additional interviews to answer questions and fill in details.

Structure the Use Case Model.

- Look for relationships
  - Generalizations
  - Includes
  - Extends
- Draw the use case diagrams.
- Write use case descriptions.

Use case name	`ReportEmergency`
Participating actors	
Flow of events	
Entry condition	
Exit condition	
Quality requirements	

Use case name	`ReportEmergency`
Participating actors	Initiated by `FieldOfficer` Communicates with `Dispatcher`
Flow of events	
Entry condition	
Exit condition	
Quality requirements	

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	
Exit condition	
Quality requirements	

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	
Quality requirements	

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	

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`.