Sem2.

- **The functional model**, represented in UML with use case diagrams, describes the functionality of the system from the user's point of view.
- **The object model**, represented in UML with class diagrams, describes the structure of the system in terms of objects, attributes, associations, and operations. During requirements and analysis, the object model starts as the analysis object model and describes the application concepts relevant to the system. During system design, the object model is refined into the system design object model and includes descriptions of the subsystem interfaces. During object design, the object model is refined into the object design model and includes detailed descriptions of solution objects.
- **The dynamic model**, represented in UML with interaction diagrams, state machine diagrams, and activity diagrams, describes the internal behaviour of the system. Interaction diagrams describe behaviour as a sequence of messages exchanged among a set of objects, whereas state machine diagrams describe behaviour in terms of states of an individual object and the possible transitions between states. Activity diagrams describe behaviour in terms control and data flows.

DIAGRAM TYPES:

• Use Case Diagrams

• Class Diagrams

• Interaction Diagrams

• State Machine Diagrams

• Activity Diagrams

Use Case Diagrams

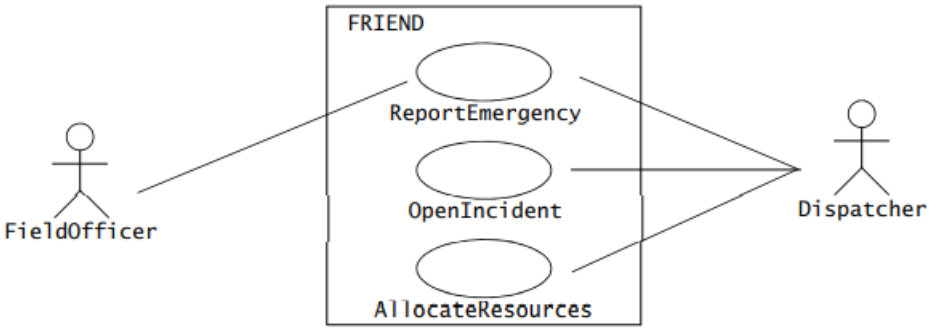
IDENTIFIING ACTORS and USE CASES

During requirements elicitation, the client and developers define the purpose of the system. The result of this activity is a description of the system in terms of actors and use cases. Actors represent the external entities that interact with the system. Actors include roles such as end users, other computers the system needs to deal with (e.g., a central bank computer, a network), and the environment (e.g., a chemical process). Use cases are general sequences of events that describe all the possible actions between an actor and the system for a given piece of functionality.

https://www.youtube.com/watch?v=zid-MVo7M-E

For example, create a use case diagram for a simple watch. The WatchUser actor may either consult the time on their 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).

Example 2, in an accident management system, field officers (such as a police officer or a fire fighter) have access to a wireless computer that enables them to interact with a dispatcher. The dispatcher in turn can visualize the current status of all its resources, such as police cars or trucks, on a computer screen and dispatch a resource by issuing commands from a workstation. In this example, field officer and dispatcher can be modeled as actors. Figure depicts the actor FieldOfficer who invokes the use case ReportEmergency to notify the actor Dispatcher of a new emergency. As a response, the Dispatcher invokes the OpenIncident use case to create an incident report and initiate the incident handling. The Dispatcher enters preliminary information from the FieldOfficer in the incident database (FRIEND) and orders additional units to the scene with the AllocateResources use case.



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

Precondition	The FieldOfficer is logged into FRIEND
PostCondition	• 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/Other req	• 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.

USE CASE PRIORITIZATION

USE CASE DETAILS

Preconditions / postconditions

USE CASE DIAGRAM - Conclusions

Systems (website, software component, business process, module, so on); represented using a rectangular, name inside

Actors (person, organization, other system, external device, so on)

- External objects (represented outside the system)
- Categorical objects (not specific)
- Type
    - Primary actors- initiate action, represented on the left side of the system
    - Secondary actors- respond to an action, represented on the right side of the system

Uses Cases (an action / flow within the system)

- Represented as an oval
- Described as verbs (+adverb - optionally)

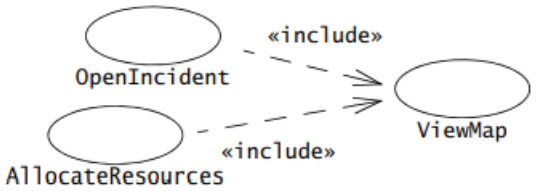
Relationships

- Association/communication -represented with simple line
- Inclusion – represented with dash line, it happens every time

- Extending – represented by a dash line, arrow towards “base” case, it happens when some criteria’s are meet
- Generalization (or inheritance), represented by straight line (90C corners, area arrow)

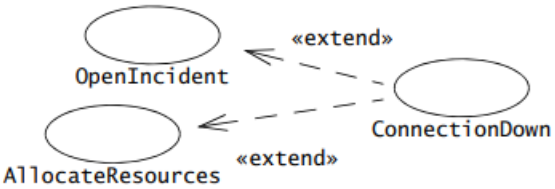
Include relationships

When describing a complex system, its use case model can become quite complex and can contain redundancy. The complexity of the model can be reduced by identifying commonalities in different use cases. For example, assume that the Dispatcher can press at any time a key to access a street map. This can be modeled by a use case ViewMap that is included by the use cases OpenIncident and AllocateResources (and any other use cases accessible by the Dispatcher). The resulting model only describes the ViewMap functionality once, thus reducing complexity of the overall use case model. Two use cases are related by an include relationship if one of them includes the second one in its flow of events. In use case diagrams, include relationships are depicted by a dashed open arrow originating from the including use case. Include relationships are labelled with the string «include».



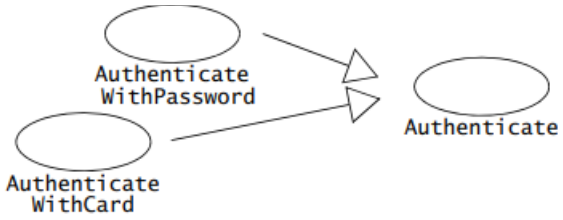
Extend relationships

They are an alternate means for reducing complexity in the use case model. A use case can extend another use case by adding events. An extend relationship indicates that an instance of an extended use case may include (under certain conditions) the behaviour specified by the extending use case. A typical application of extend relationships is the specification of exceptional behaviour. For example, next figure assume that the network connection between the Dispatcher and the FieldOfficer can be interrupted at any time. (e.g., if the FieldOfficer enters a tunnel). The use case ConnectionDown describes the set of events taken by the system and the actors while the connection is lost. ConnectionDown extends the use cases OpenIncident and AllocateResources. Separating exceptional behaviour from common behaviour enables us to write shorter and more focused use cases. In the textual representation of a use case, we represent extend relationships as entry conditions of the extending use case. For example, the extend relationships depicted in next figure are represented as an entry condition of the ConnectionDown use case



Inheritance relationships

An inheritance relationship is a third mechanism for reducing the complexity of a model. One use case can specialize another more general one by adding more detail. For example, FieldOfficers are required to authenticate before they can use FRIEND. During early stages of requirements elicitation, authentication is modeled as a high-level Authenticate use case. Later, developers describe the Authenticate use case in more detail and allow for several different hardware platforms. This refinement activity results in two more use cases: AuthenticateWithPassword which enables FieldOfficers to login without any specific hardware, and AuthenticateWithCard which enables FieldOfficers to login using a smart card. The two new use cases are represented as specializations of the Authenticate use case. In the textual representation, specialized use cases inherit the initiating actor and the entry and exit conditions from the general use case.



HOMEWORK

1) Restaurant: Create a business use case diagram for a Restaurant. A customer can order a meal, the restaurant must have suppliers, employees, contractual personal,
2) Online shopping: Provide top level use cases for a web customer making purchases online. Web customer actor uses some web site to make purchases online. Top level use cases are View Items, Make Purchase and Client Register.
3) Credit card processing system: Define major use cases for a credit card processing system (credit card payment gateway). The merchant submits a credit card transaction request to the credit card payment gateway on behalf of a customer. Bank which issued customer's credit card is actor which could approve or reject the transaction. If transaction is approved, funds will be transferred to merchant's bank account.

Class diagrams

https://www.youtube.com/watch?v=UI6lqHOVHic&t=3s

Class diagrams describe the structure of the system in terms of classes and objects. Classes are abstractions that specify the attributes and behaviour of a set of objects. A class is a collection of objects that share a set of attributes that distinguish the objects as members of the collection. Objects are entities that encapsulate state and behaviour. Each object has an identity: it can be referred individually and is distinguishable from other objects.

Scenarios

A use case is an abstraction that describes all possible scenarios involving the described functionality. A scenario is an instance of a use case describing a concrete set of actions. Scenarios are used as examples for illustrating common cases; their focus is on understandability. Use cases are used to describe all possible cases; their focus is on completeness. We describe a scenario using a template with three fields:

• The name of the scenario enables us to refer to it unambiguously. The name of a scenario is underlined to indicate that it is an instance.

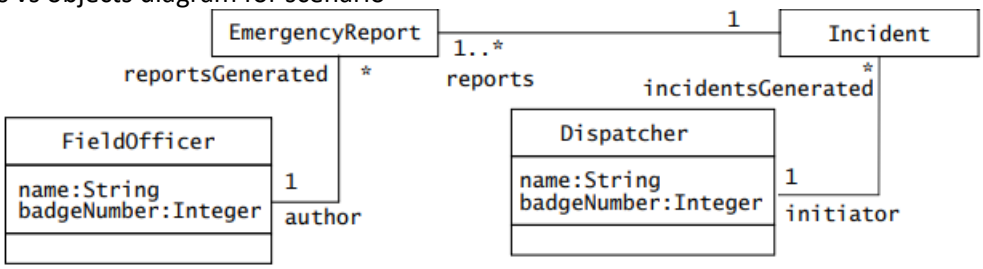
• The participating actor instances field indicates which actor instances are involved in this scenario. Actor instances also have underlined names.

• The flow of events of a scenario describes the sequence of events step by step.

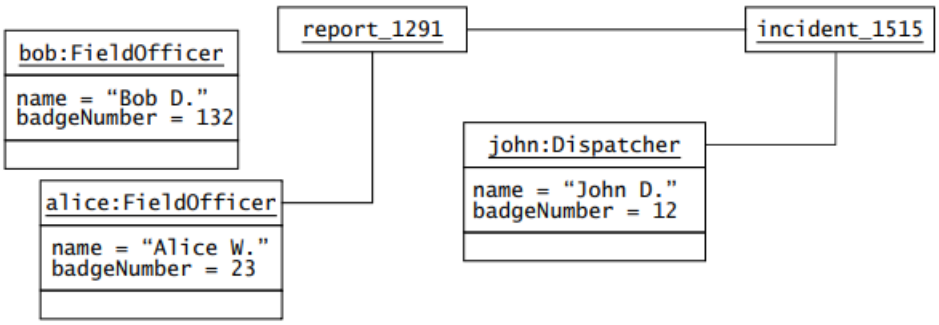
Scenario name: warehouseOnFire
Participating actor instances: bob, alice – FieldOfficer, john – Dispatcher
Flow of events

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.

Classes vs objects diagram for scenario



Classes diagram for scenario



Objects Diagram for scenario

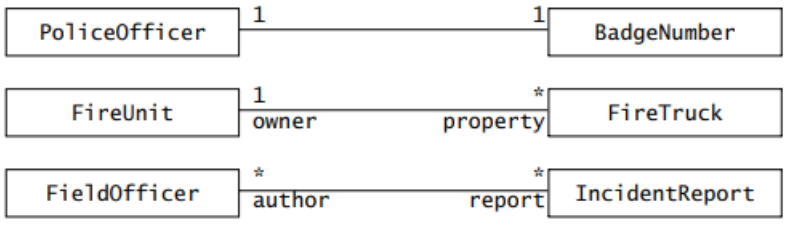
Multiplicity

- A one-to-one association has a multiplicity 1 on each end. A one-to-one association between two classes (e.g., PoliceOfficer and BadgeNumber) means that exactly one link exists between instances of each class (e.g., a PoliceOfficer has exactly one BadgeNumber, and a BadgeNumber denotes exactly one PoliceOfficer).

- A one-to-many association has a multiplicity 1 on one end and 0..n (also represented by a star) or 1..n on the other. A one-to-many association between two classes (e.g., FireUnit and FireTruck) denotes composition (e.g., a FireUnit owns one or more FireTrucks, a FireTruck is owned exactly by one FireUnit).

- A many-to-many association has a multiplicity 0..n or 1..n on both ends. A manyto-many association between two classes (e.g., FieldOfficer and IncidentReport) denotes that an arbitrary number of links can exist between instances of the two classes (e.g., a FieldOfficer can write many IncidentReports, an IncidentReport can be written by many FieldOfficers). This is the most complex type of association.

Example:



Association / Aggregation / Qualification / Inheritance

....

Mr prof Chiorean recommended for understanding how to determine the pre/post conditions in a problem the following youtube films:

https://www.youtube.com/watch?v=KeMgPqLCkuo&t=305s
https://www.youtube.com/watch?v=8qyHeqInnFU&t=7s
https://www.youtube.com/watch?v=iex3QbSo61c&t=33s
https://www.youtube.com/watch?v=3OWNQddbrOs&t=7s
https://www.youtube.com/watch?v=FrdCX9Gcc4g&t=28s

Homework: Create a class diagram for the exercise 1-3 chosen from the first part of the seminar.