# Modeling - Classes

When making paper prototypes, you’re **supposed** to have some aural explanation with the video of the paper prototype walkthrough. It doesn’t help much if you don’t explain what you’re doing while going through the prototype.

**Model**

A simplification of something in the real world. The purpose of the model is not necessarily to predict how it behaves, but also to understand it.

After creating a system definition, we need to get a deeper understanding of the problem we are trying to solve. The system definition is an important part of doing that. So are the users and people involved. This will be concerning the class activity.

## Problem-domain analysis

The purpose is to identify and model a problem domain.

We already know what a problem domain is, so we will regard it as we are used to. We will regard the **model** as a description of classes, objects, structures, and behaviour in a problem domain.

We need to model the real world as *future* users will see it.

We also need to get an overview first, and then supply with details.

The output of this activity is a coherent model of a problem domain.

To do all this, we need to understand the users mental models. We need to create a tangible and visible model of these mental models. Because users are different, we need to include mental models and ideas, and systematically choose which one is the best fitting one.

Problem domain analysis is parted into three main activities:

*To find relevant elements and events to model in our system.*

*To build the model and find the structural elements in our model; how do the different objects and events relate to each other?*

*Behaviour; how do the objects behave, and what triggers the events we have created?*

When finding relevant elements to model, we need to consider whether or not they are static or not. A CPR-number is usually very static. Color can be static or non-static, depending on the context. Always understand the context before trying to model the environment.

## Model the context

Problem domain: What are the important elements we need to model?

Application domain: What are the important parts of the context of the use we need to take care of?

## Model of the problem domain

We need to design our model so that when something changes in the real world, that change should be reflected in our model. We need to be able to show change from the real world in our model to be able to show the same change in our application domain.This gives us an idea of what entities and events we need to model in order to be able to do this.

As mentioned before, we distinguish between three activities: Classes, structure, and behaviour. This will only be regarding classes.

During the class activity, we are looking for objects and events that are relevant to our system.

In the structure activity, we see how the classes and objects are conceptually linked together.

In the behaviour activity, we see which dynamic properties the objects have. What triggered the events, and what is the result?

Main concepts when analyzing the problem domain:

Objects are entities with identity, state and behavior.

Classes are descriptions of collections of objects sharing structure behavioral pattern, and attributes.

Events are instantaneous incidents involving one or more objects.

Main principles when analyzing the problem domain:

Classify objects by classifying them in the problem domain. We then characterize the objects **through their events**.

We need to have an open mind when coming up with candidates, but also have to select critically thereafter.

The result should be an event table with classes and relevant events.

## 

## 

## Classify objects and events

We try to identify different objects that share the important characteristics and group them together in classes. Similarly with events, there are a lot of different events happening. However, some of these events are expressions of the same thing. That is, we generalize and make abstractions.

Examples of classes:

Customer

Assistant

Agreement

Example of events:

Agreement made

Customer serviced

See slide 12 for an example of an event table. You can see the different classes and what events are tied to them. That is, their behaviour.

When a customer is reserved, it’s the assistant’s job to make the reservation. The result of that event is an appointment, which is why that is also included.

It’s important to remember that the purpose of this activity is to get an understanding of the problem domain that allows of to make a model of it.

## Activities in “classes”

Find candidates for classes: Find potential classes that could be included in the system. Be open minded.

Find candidates for events, same thing.

Then we evaluate and select systematically. The selection is based on the criteria described in the book, but they are pretty intuitive on their own.

### Find classes

Make a list of all potentially relevant classes. Consider a lot of sources like your own perception, existing descriptions and like rich pictures and system definitions and stuff like that. Or, you could collaborate with prospective users.

The *names* of the classes must be **simple and reachable, originate in the problem domain** and should be able to **describe a single instance** of that class**.**

Based on the first list of potential classes, we can start reducing it.

Generate a similar list of candidates for events.

Here, we can use our own perception or existing descriptions as sources.

Eliminate verbs related to the way users carry out their job, because that belongs to the application domain. The naming conventions look a lot like the ones for classes.

## Evaluate and select systematically

**General evaluation criteria**

Is the class or event within the system definition?

Is the class or event relevant for the problem-domain model?

If either is a no, discard the candidate.

**Evaluation criteria for classes**

**Can you identify objects from the class?** If you can’t you probably don’t need the class.

**Does the class contain unique information?** If it doesn’t, you are at the risk of creating redundancy in your system. If your class contains the same information as another class, two classes need to change at some event, which creates a lot of overhead.

**Does the class include multiple objects?** Unless we can identify multiple instances of this class, it might not be necessary to model it. Classes are a collection of shared properties and behaviors across objects. Therefore, if there is only one instance of it necessary in the system, the abstraction level is nowhere to be found. Remember that this is USUALLY how it’s done. A boss, for example, can be modeled because it can handle events that customer classes cannot, even though there is only one boss.

**Does the class have a suitable and manageable number of events?** If your class has too many events, you might need to split that class into different subclasses.

***Side note:*** Usually, we don’t want to have classes that are just lists of other classes in our system. For example, a list of customers in a banking system should not be a valid candidate, because there is no abstraction level. However, if the list of customers is an explicit part of the problem domain (fx. Mentioned in the system definition), it should be included as a candidate regardless, even though it doesn’t satisfy some of the criteria.

As an example, a system definition for a mobile phone system may mention that the system must contain a contact list, and that this must be updateable. This means that the class “Contact List” should be a central class, even though it’s just a list of the class “Contact”.

**Evaluation criteria for events**

**Is the event instantaneous?** It can be hard to determine when a continuous activity happens. The event must be instantaneous.

**Is the event atomic?** Can it be divided into subevents? If we can, we need to do that. An event must not be atomic.

**Can the event be identified when it occurs?** If we can’t, we need to look for other events.

Look at slide 22 for the selection of different class candidates.

We don’t need a customer database, as we can get this from just grouping customers. We don’t need a separate class for that. Same thing for appointment book.

We don’t need a cash register class, as it is not a part of the system definition. Treatment performed is an event, not a class. Desired vacation should not be a class since it is not a part of the system description. The work schedule would be covered by the plan.

Boss, assistant and receptionists need to be there.

The system definition says that every employee has their own chair, which is why the chair class becomes redundant.

Look at slide 23 for the selection of different event candidates.

We probably don’t need customer arrived, since it is pretty evident if the customer is there or not.

Note the “agree” that is a part of the events. This is a good example of bad naming. We are talking about a specific type of agreement, and that is not clear from the naming of the event.

# Lecture 4 exercises

System definitions for the airport traffic control system

We need to make sure that our system definitions satisfy all the criteria in the FACTOR (or BATOFF) criteria set.

Notes from Esben: *In any system, the system itself is actually part of the application domain. As long as something can affect the use situation, it should probably be part of the application domain. However, it’s up to you how much detail you want to get into, and whether or not the repair man should be included in the application domain because when he repairs the system it goes down, and that affects the traffic control.*

### S1

**F**unctionality: The system should allow the user to communicate with all vehicles, allow take-off and landing, and supply air vehicles with routes.   
**A**pplication domain: The system should be designed for use by air traffic generals relevant personnel in the radio tower.   
**C**onditions: The system should not allow for downtime or erroneous information. The system should be robust, safe, and completely tested. The system cannot undergo live-testing.   
**T**echnology: Should be developed for a Windows PC. Development must include interfacing for radar and whatever communication hardware they currently apply for radio tower/airplane communication.   
**O**bjects: The system should model all air and ground vehicles and the map of roads, including runways and taxi-roads. Furthermore, the system should contain a timed schedule for when planes should arrive and leave the airport.  
**R**esponsibility: Flight monitoring and communication medium.

The system should be responsible for monitoring and facilitating communication between radio towers and planes. It should be designed for use by several air traffic controllers and other relevant personnel in the radio tower. The system should allow the user to communicate with all vehicles, change the status of roads, allow for take-off and landing, and provide active airplanes with routes. Following this, the system should contain information about all vehicles, road maps, timed schedules, and airspace. The system should not allow for downtime or erroneous information. The system should be robust, safe, and completely tested. During development, the system cannot undergo live-testing.

### S2

**F**unctionality: Warn when a collision course is entered, divert from course, communicate with other planes,   
**A**pplication domain: Pilots  
**C**onditions: The system should not allow for downtime or erroneous information. The system should be robust, safe, and completely tested. The system cannot undergo live-testing.   
**T**echnology: Whatever hardware airplanes contain  
**O**bjects: All planes in the same airspace and itself. Positions, current routes, velocity, size  
**R**esponsibility: Collision prevention

The system should automatically handle collision detection and prevention in airplanes. The system should be designed for use by pilots, and should be developed for implementation in the proprietary airplane hardware. The system should allow for the pilot to communicate with other planes, as well as alarming the pilot when a collision course is entered. The system must be able to represent information regarding the position, course, velocity, and size of itself and all other planes in the same airspace. The system should not allow for downtime or erroneous information. The system should be robust, safe, and completely tested. During development, the system cannot undergo live-testing.

### S3

**F**unctionality: The system should automatically communicate with all vehicles, allow take-off and landing, and supply air vehicles with routes based on information received from all vehicles and road maps.  
**A**pplication domain: Sys\_admin, root, Zezima, ReturnOfWilderness, Pilots  
**C**onditions: The system should not allow for downtime or erroneous information. The system should be robust, safe, and completely tested. The system cannot undergo live-testing.   
**T**echnology: Hardware with the ability to receive information from all airplanes, a central server, and a visual interface.   
**O**bjects: The system should model all air and ground vehicles and the map of roads, including runways and taxi-roads. Furthermore, the system should contain a timed schedule for when planes should arrive and leave the airport.  
**R**esponsibility: Monitoring flight control as well as making decisions regarding routing and permissions, and communicating these decisions to the pilots.

The system should monitor flights and make decisions based on the best possible routings and communicate these decisions to the pilots. The system will be in contact with all vehicles and convey information to the pilots regarding landing and takeoff as well as other relevant information. The system will be monitored by a system admin as well as a flight controller. The pilots themselves will be in contact with the system, and can impact the system by not following the instructions given.

# Classes

Profile

Component

Packaged Solution

Energy Label

# Events

ProfileCreated

ComponentCreated

PackagedSolutionCreated

ComponentAddedToPackagedSolution

ComponentRemovedFromPackagedSolution

EnergyLabelCalculated

PDFCreated

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Profile | Component | Packaged Solution | Energy Label | Database |
| Profile Created | X |  |  |  |  |
| Component Created | X | X |  |  | X |
| Packaged Solution Created | X |  | X |  | X |
| Component Added To Packaged Solution |  | X | X |  |  |
| Component Removed From Packaged Solution |  | X | X |  |  |
| Energy Label Calculated |  |  | X | X |  |
| Energy Label Formatted |  |  | X | X |  |
| Energy Label Exported | X |  | X | X |  |