Object oriented analysis & design notes

Mikkel Helsing Andersen

September 24, 2023

Contents

Part I

Background

Chapter 1

Method

Concepets

Object	Entity with identity, state and behaviour
Class	Describes a collection of objects sharing structure, behavioural patterns and attributes
Problem domain	Part that is administrated, monitored or controlled by a system
Application domain	The organization that administrates the problem domain Where the user is
System	A collection components that implements modeling requirements, functions and interfaces
Context	Problem domain and application domain

Describe problem- and application domain better

Der er eksempler bag i bogen

Problem domain

Class structure and behaviour

Application domain

Usage functions and interfaces

Method

Purpose, concepts, principles and results.

1.1 Objects and classes

Objects - *Entity with identity, state and behaviour*

Each object serves as a seperate function. The object could be a customer, where specific people are treated as customers. The object contains that specific customers identity, state and behaviour

Class - *Describes a collection of objects sharing structure*

The class contains multiple objects, meaning a customer class will contain multiple data points. The class also contains multiple different customers and their data points.
When describing a class it's important to choose the right granularity. Gravel pit should not be described by the individual grains of sand, instead by the type, whereas a warehouse the individual packages should be described.

Analysis - outside the system

In analysis the object's behaviour is described by its events it performece and experiences that happens in definite points in time. Eg. customers ordering and shipping goods.

Design - inside the system

In design the object's behaviour is described by the operations it can perform and make available to other objects in the system. Eg. add order etc.
This allows the update of eg. the customers object state. The design object encapsulates the internal representation of the object state through its operations.

1.2 Principles

The 4 principles:

Model the context - Useful systems fit the context, so model both application and problem domain during analysis and design.

Emphasize the architecture - Understandable architecture makes collaboration between programmers and designers possible. Flexible architecture makes modifications and improvements affordable

Reuse patterns - Building on well-established ideas and pretested components

Tailor the method to suit specific project - Must be tailored to the specific needs of the analysis and design situation

Model

Model is a representation of the state in the problem domain. How often the model is updated is a design decision
Problem domain -¿ model -¿ application domain.

Chapter 2

System Choice

Define the system in its context problem domain + application domain.
Using the F.A.C.T.O.R system.
F: Functionality - System functions that support the AP tasks.

A: Application domain - The organization that administrates the problem domain. Where the user is

C: Condtion - The conditions under which the system will be developed

T: Technology - Both the technology used to develop the system and the technology which the system will run.

O: Objects

R: Responsibility

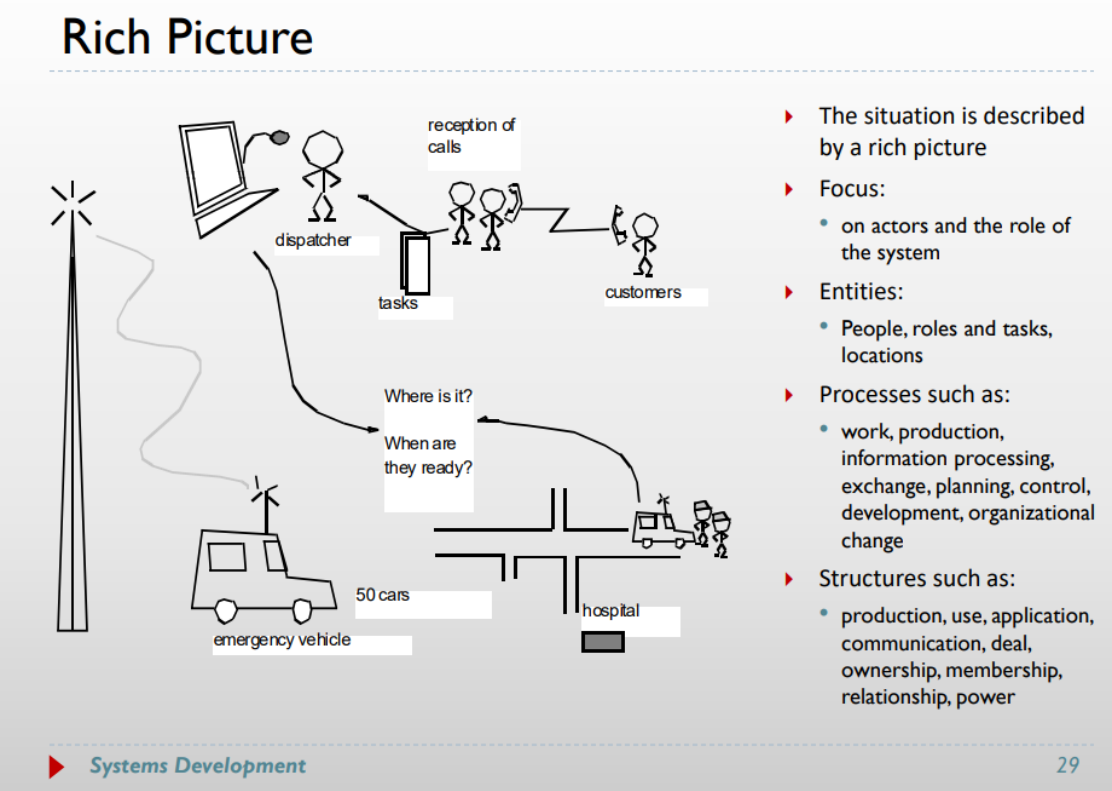
2.1 System definition

Situation

Describe the situation based on the context of the system. Described in a rich picture. Which is the described by its focus, entities, procceses and structure.

Ideas

Examples - eg. Study preexisting systems.



System definition

Not to be confused with *system*. A concise description of a computerrized system expressed in natural language.

Context

2.2 Problem Domain Analysis

The result of a problem domain analysis is a class diagram describing classes and structure.

Event

An event in the problem analysis is:
- Atomic
- An incident involving one or more objects

- Instantaneous

System Model

The system's model of the problem domain
- Represents the state of the problem domain
- Provides information to users in the application domain about the problem domain

Part II

Problem Domain Analysis

Chapter 3

Classes (OOAD P. 51)

To model the problem domain, starting with class activity.

3.1 Class Activity

Abstraction, classification and selection are the primary tasks in the class activity.

Abstraction

Abstracting the problem domain phenomena by seeing them as objects and events.

Classification

Then classifying the objects and events.

Principle - *Classify objects in the problem domain*

Step 1: Abstraction and classification should lead to identifying all relevant objects, which should be develop a rich list of potentially relevant classes for the problem-domain model. In a parallel activity, identify and develop a similar list of events.

Selection

Then selection which classes and events the system will maintain information on. Each class is characterized by a set of specific events.

Step 2: Systematically evaluate the candidates and select relevant classes and events to be included in the problem-domain model. Finally relate events to classes.

Event table (OOAD P. 52)

Classes / **Events**	Customer	Assistant	Apprentice	Appointment	Plan
Reserved	✓	✓		✓	✓
Cancelled	✓	✓		✓	
Treated	✓			✓	
Employed		✓	✓		
Graduated			✓		
Agreed		✓	✓		✓

Figure 3.1: Event table for Hair Salon System (P. 52)

3.2 Classification of Objects and Events (OOAD P. 52)

Object

During the problem analysis an object is an abstraction of a phenomena in that problem domain. An object should be indetifiable be an independent entity, which is delimited. Using events is emphasized by:

Principle - *Characterize objects through their events*

Event

Events specify the qualities of problem-domain objects. An event is defined as:

Event - *An instantaneous incident involving one or more objects*

An event is an abstraction of a problem-domain activity or process that is perfomed or experienced by one or more objects.

Class

Classes contain objects and event. These are identify all the objects and events to be included in a relevant problem-domain model. The class concept refers and describse all the objects in a specific class:

Class - *A secription of a collection of objects sharing structure, behavioral pattern, and attributes*

Object's structure, behavioral pattern, and attributes are described in general terms by the appopriate class deiniftion. Where all classes are different.

Find Classes (OOAD p. 57)

Class selection is the first and most basic for building problem-domain model. It's important to write down all potentially relevant classes, without evaluating them in detail. Do this using own perception, pre-existing decriptions and definitions, including prospective users by interviewing and observing them work. This can be further expanded by using pre-existing system, and using the experience as an advantage. This also include regulations in the are of the operation of the system. This should in a list of class candidates, with easy to understand name, that references in the problem domain.

Event

Find Events (OOAD p. 59)

Using the same fundamental principle as the **find classes section**.

Evalute Systematically (OOAD p. 62)

Fundamental evaluation rule: a class or event should be included in the problem-domain model *iff* system functions use information about it. Basic criteria rules:

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

Only classes and events within the problem-domain should be selected. These classes and events should refer to phenomena that will be administrated, monitored and controlled by future users in their work.

Users are usually not part of the problem domain, unless the users are being registered within the system, in eg. the case of restricted access.

Evaluation Criteria for Classes (OOAD p. 63)

As a rule, these questions should be answered when evaluating classes:

- Can you identify objects from the classes?
- Does the class contain unique information?
- Does the class encompass multiple objects?

- Does the class have a suitable and manageable number of events?

An object should be unambiguously be indetifyalbe from a class. Typically a class will contain multiple objects and should contain an appropriate amount of unique information. Events are realated to classes in order to characterize them. This is expressed through an event table. Classes can be further specified than their name by descriping their responsibilities, this limits confusion when the system is developed. This is done through plain text and can specify why it differs from other classes.

Evaluation Criteria for Events (OOAD P. 65)

As a rule, these questions should be answered when evaluating events:

- Is the event instantaneuou?
- Is the event atomic?
- Can the event be identified when it happens?

It should be defined as instantaneous to make it clear when it happens. If the incident happens over a period a stop and start event can be used. Defining and naming an event determines the granularity of the time model. Each event should be identifiable when it happens.

Relating Classes and Event (OOAD P. 66)

When selecting a class, one also has to define the events that the class objects are involved in. From event to objects should also be defined. As a rule, these questions should be answered when relating:

- Which events is this class involved in?
- Which classes are involved in this event?

These should be summarized in an event table. Which can also be used to evluate the quality of class and event candidates.

Summarized principles can be seen on OOAD P. 66

Chapter 4

Structure (OOAD p. 71)

In the structure activity an extension of the problem-domain description to include structural relations between classes. Including relations between objects. The result of the structure activity is a class diagram.

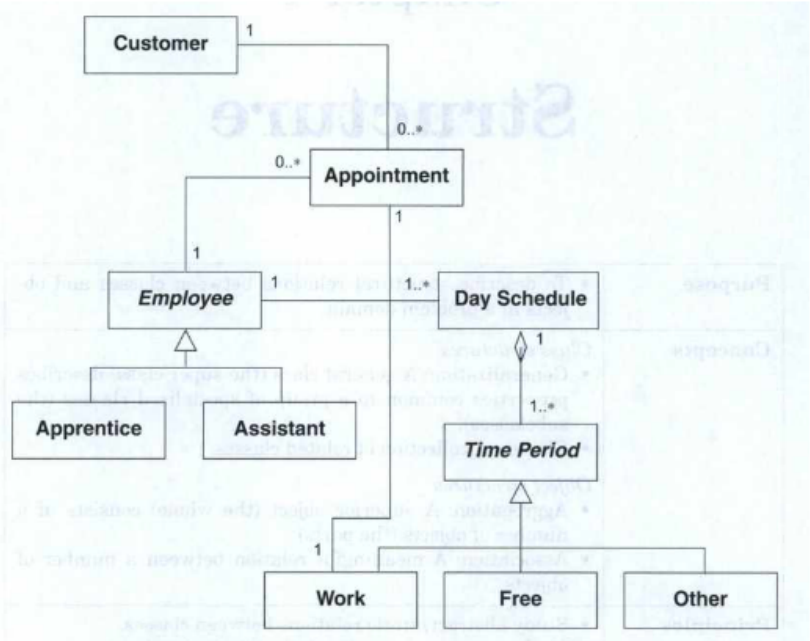


Figure 4.1: Class diagram example (OOAD p. 72)

4.1 Class Structure

Class structures express static, conceptual relations between classes. They connect classes, and the relationship doesn't change unless the description itself changes.

Generalization

A general class(super-class) that describes properties common to a group of specialized classes(sub-classes). It's a relation between two or more specialized classes(sub-class) and a general class(super-class). Everything that holds true for the super-class also hold for any sub-classes. Programmingvise its inheritance/extending a super class.
This can be tested with "is-a", eg. "doctor is a person" etc.

Cluster

A collection of related classes. Classes within a structure is usually connected by a generalization class, eg. cars like in the example below. Alternatively connected through aggregation structure.

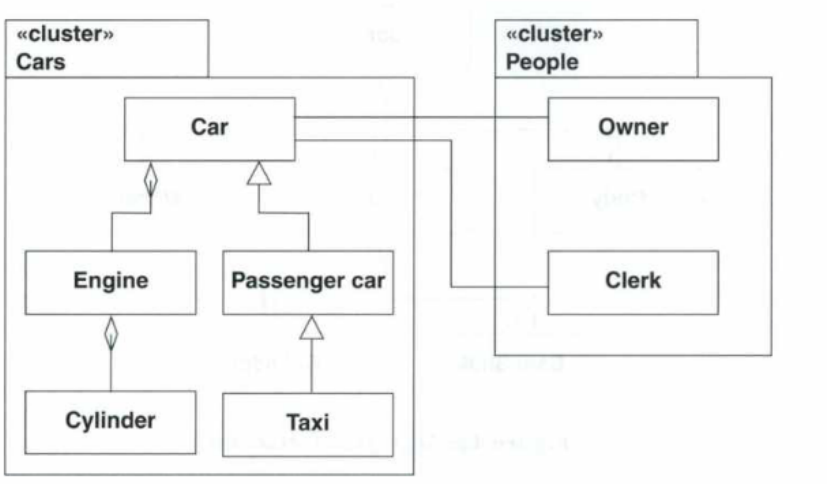


Figure 4.2: Example of clusters and cluster structure (OOAD p. 77)

4.2 Object Structure

Object structures express dynamic, concrete relations between objects. Theses relationship can change dynamically without changing the underlying description.

Aggregation

A superior object (the whole) consists of anumber of objects (the part). It's a relation between two or more objects. It expresses that one object is a fundamental and defining part of the other.

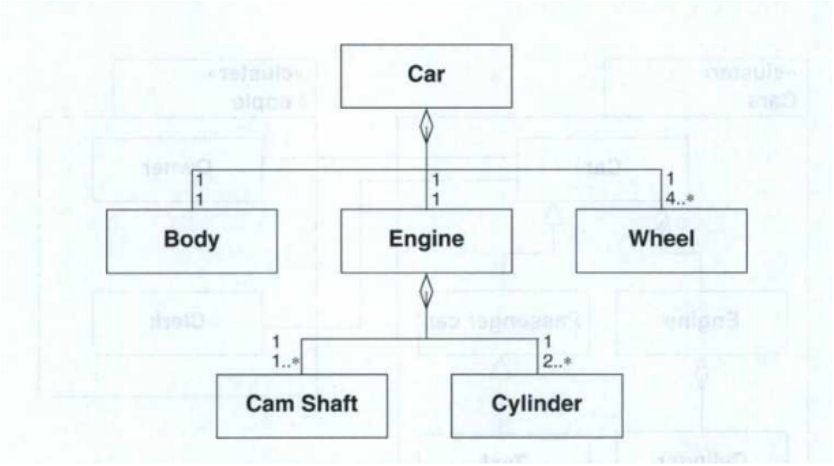


Figure 4.3: Aggregation structure example (OOAD p. 78)

Association

A meaningful relation between a number of objects. Objects are not a defining part of an object. The association structure does not define ranking

4.3 The Activity

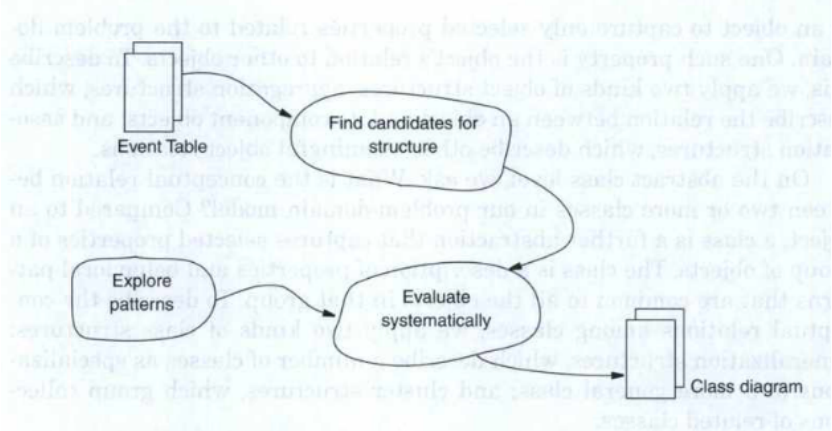


Figure 4.4: Sub-activities involved in the structure activity (OOAD P. 74)

4.4 Finding Structure (OOAD P. 79)

Finding Candidates

Similarly to selecting classes and events, it all starts with selecting candidates by brainstorming or other means.

Identify Generalization (OOAD P. 80)

Do this by taking all classes and pair them to see if one is a generalization of the other. Then determine whether it's a relevant relation.
Then try to create a relevant specialized or generalized class.

Identify Aggregation (OOAD P. 80)

Find classes with objects that has a decomposition[1] object of the other classes. The whole is considered to be superior to its parts, as is reflected in the class diagram by the vertical placement.
Determine if it's relevant to aggregate the objects of a selected class into a new one.
Our third approach is to determine if each class can be decomposed few relevant classes that do not yet exist in the model. Alternatively, consider aggregating the objects of a selected class into a new class.

[1]Decomposit: The combining of distinct parts or elements to form a whole

Softer definitions of aggregation:

- Whole-part, in which the whole is the sum of the parts, if adding or removing any part, the whole fundamental changes.
- Container-Content, in which the whole is a container for the parts, if adding or removing any content, the fundamental properties of the whole doesn't change.
- Union-Member, in which the whole is an organized union of members. It doesn't change the union fundamentally by adding or removing a few members. However, there is a lower limit on the number of members, as it is artificial to model a union without members.

Identify Associations

See if the remaing class pairs can be meaningfully related. This is done when the relation needs to be administrated, monitored or controlled.

Identify Clusters

Increase class diagrams clarity by organizing conceptually related classes into clusters. A class is not allowed to be in two different clusters.

4.5 Explore Patterns (OOAD p. 82)

The Relation Pattern

Used when a relation between two objects carry it's own properties.

The Hierarchy Pattern

When the relation is between multiple objects and are in a hierarchy, where each level organizes into the hierachy. Eg. Student is related to a class which is related to a semester... etc.
A variation of this can also have objects related to multiple on a higher level. In this situation the elements are not mutually exclusive.

The Item-Descriptor Pattern

Where a descriptor class defines specific properties shared by all related objects. This is important in situation where multiple copies are treated as seperate entities, but still share specific properties between them.

4.6 Evaluate Systematically (OOAD p. 86)

Principle - *Model only the neccessary structural relations*

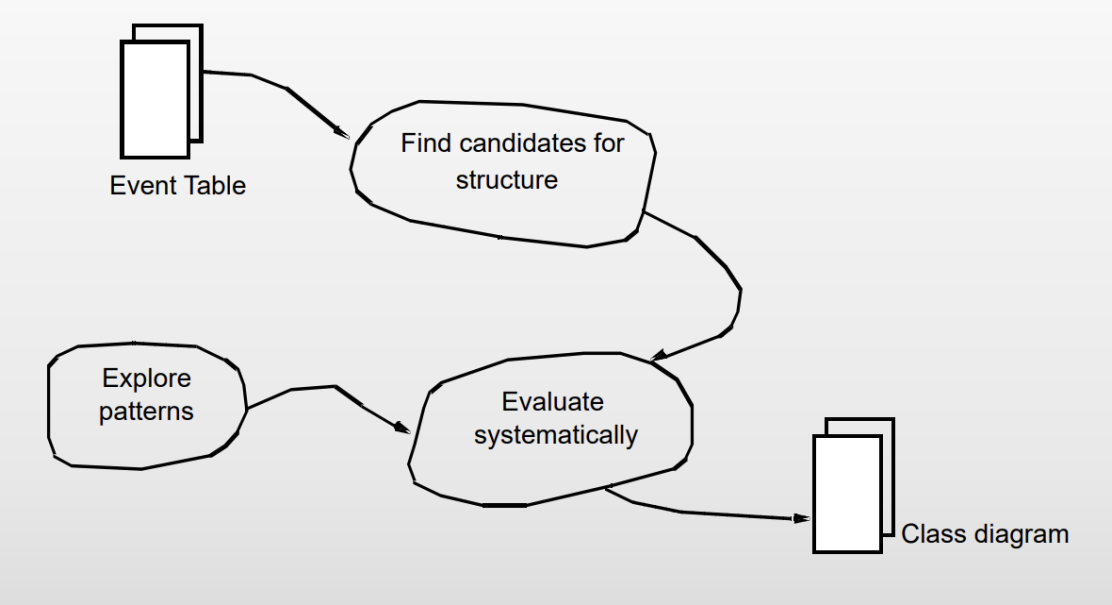
When evaluating structural relations, it's beneficial to follow certain criteria:

- Structures must be used correctly (OOAD p. 86) .
- Structures must be conceptually true (OOAD p. 87) .
- Structures must be simple (OOAD p. 88) .

Chapter 5

Behavior (OOAD P. 91)

Purpose	• To model the dynamics of a problem domain
Concepts	• Event trace: A sequence of events involving a specific object. • Behavioral pattern: A description of possible event traces for all objects in a class. • Attribute: A descriptive property of a class or an event.
Principles	• Create behavioral patterns from event traces. • Study common events. • Derive class attributes from behavioral patterns.
Result	• A behavioral pattern with attributes for every class in a class diagram.



5.1 Behavioral Pattern and Attributes

In the behavior activity the behavior is described more precisely by adding relative timings to events. Object behavior is defined by an event trace, that exhibits a certain ordering of events.

Event trace (OOAD p. 92)

Event trace - *A sequence of events involving a specific object*

An event trace is unique for a single object. It's the precise sequence in a time interval.

Birth $\rightarrow$ event_1 $\rightarrow$ event_2 $\rightarrow \cdots \rightarrow$ event_n $\rightarrow$ death.

Behavioral pattern (OOAD p. 92)

Behavioral pattern - *A description of possible event traces for all objects in a class.*

To produce the pattern, examples of event traces for individual objects in a class are used. For problem-domain dynamics, the object collaboration is emphasized; study common events. Common events are used to describe interactions between classes.

5.2 Notation for Behavioral Patterns

- (+) Sequence: Events in a set occur one by one.
- (|) Selection: Exactly one out of a set of events occurs.
- (*) Iteration: An event occurs zero or more times.

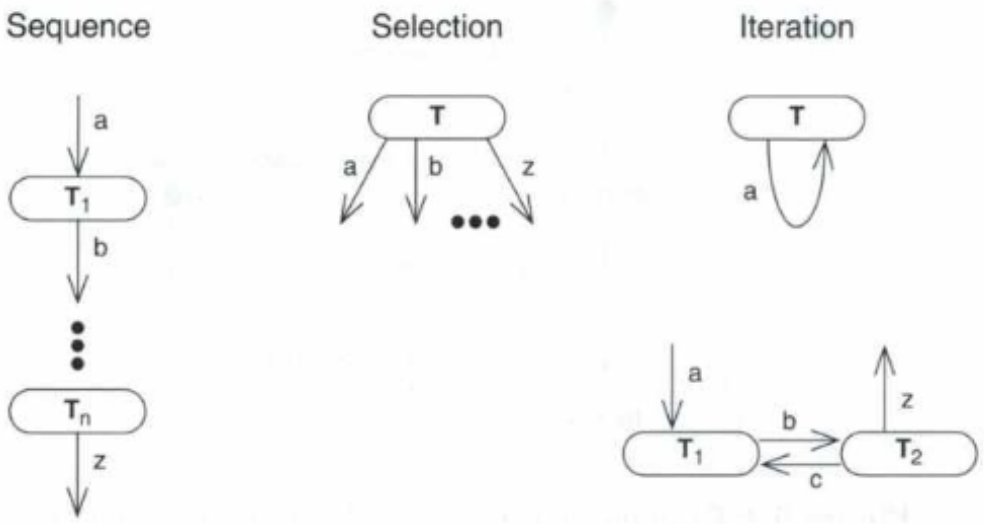


Figure 5.3: Control structures in a statechart diagram

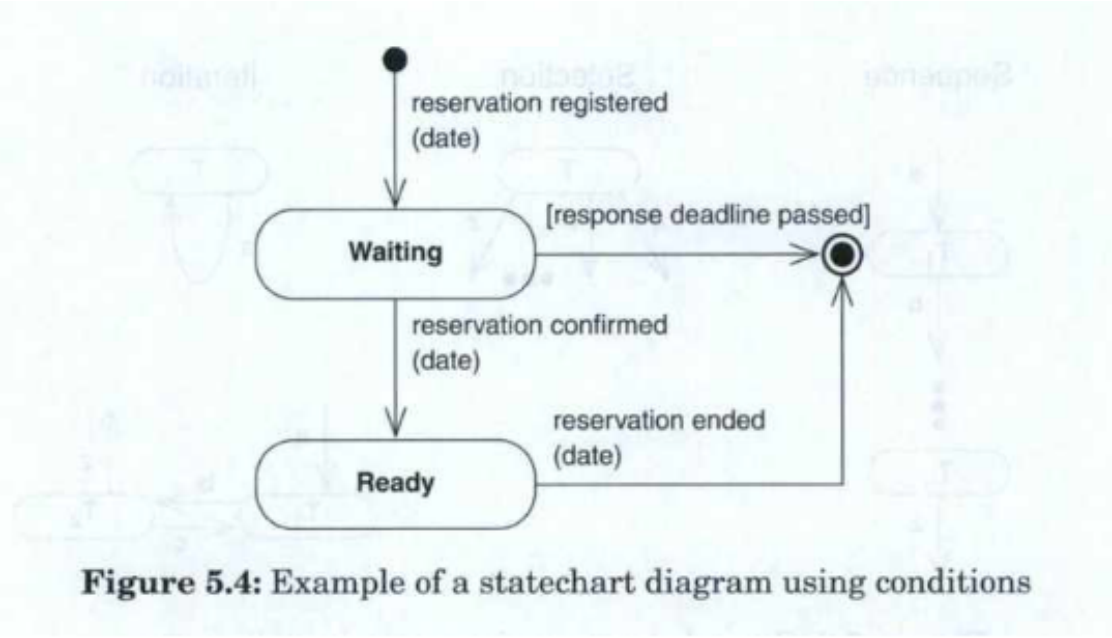


Figure 5.4: Example of a statechart diagram using conditions

5.3 Describe Behavioral Patterns (OOAD P. 98)

In the class activity the connection between classes and events was created in an event table. These unordered events can then be ordered by identifying the first and last events in an objects life.

- Which events cause the creation of a problem-domain object? These events are grouped as selections that can cause the birth of an object.
- Which events cause the disappearance of a problem-domain object? These events are grouped as selections that can cause the death of an object.

An object is birthed when the first event is triggered, and is dead when the last occurs. This does however not mean it ceases to exists, only events cannot occur on it any more.

Generally ensure that the problem-domain model includes both structured and unstructured forms of behavioral patterns.

Unstructured

The unstructured form contains a collection of intermediate events in a combination of selction and iteration. Meaning events can occur an arbitrary amount of times in an arbitrary order.

Structured

Characterized by an overall sequence, that includes all major events between birth and death. Physical objects and documents are often described this way.

When describing behavioral patterns using event traces, ask these questions:

- Is the overall form structured or unstructured?
- Which events occur together in a sequence?
- Are there any alternative events?
- Can a given event occur more than once?

Sufficient, but Simple (OOAD P. 100)

Some times conflicting goals:

- The behavioral pattern should be sufficiently precise to describe all legal, and thus all illegal, event traces.
- The behavioral pattern should provide an overview and thus be as simple as possible.

The first goal is to capture all allowable event traces sufficiently, by describing typical behavior.
The second goal is to make simple descriptions.

Inheritance of Behavioral Patterns (OOAD P. 102)

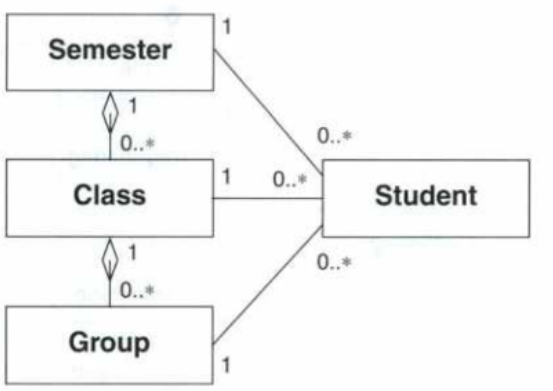
Specialized classes will inherited its super classes behavior, and usually build more subpatterns on top. Inheritance means it inherites all behavior and their timings.

5.4 Explore Patterns (OOAD P. 104)

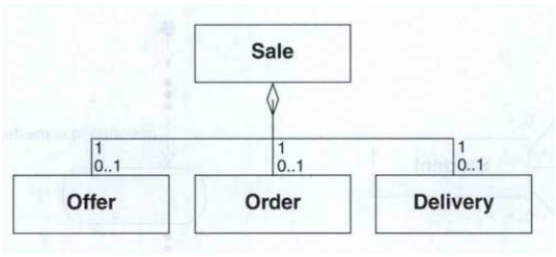
Three different models.

The Stepwise Relation Pattern (OOAD P. 104)

Stepwise relation pattern is used when certain objects are related to the elements of a hierarchy in a stepwise of sequential manner.

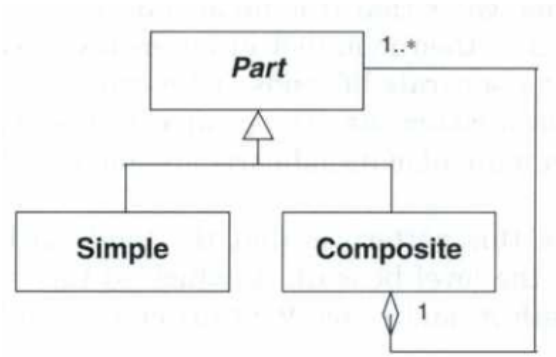


The Stepwise Role Pattern (OOAD p. 104)



The Composite Pattern (OOAD p. 107)

The composite pattern offers a way to describe the creation and destruction of a hierarchy using a detailed structure that is unknown at model-development time.



5.5 Consider Structure

Aggregation and Association (OOAD p. 108)

- If two or more objects have common events, consider adding an aggregation or association structure between them.
- If two classes are related by an aggregation or association structure, at least one common event should be considered.

Generalization (OOAD p. 109)

- If the same event is tied to two classes, consider whether one class is a generalization of the other.
- If two classes have many events with the same name, consider whether they are different specializations of a third class.

Part III

Application Domain Analysis

Purpose	• To determine a system's usage requirements.
Concepts	• Application domain: An organizarion that administrates, monitors, or controls a problem domain. • Requirements: A system's externally observable behavior.
Principles	• Determine the application domain use cases. • Collaborate with users.
Result	• A completelist of the system's overall usage requirements.

Concepts:

- Actors (Users and other systems)
- Use cases
- Functions
- Interfaces

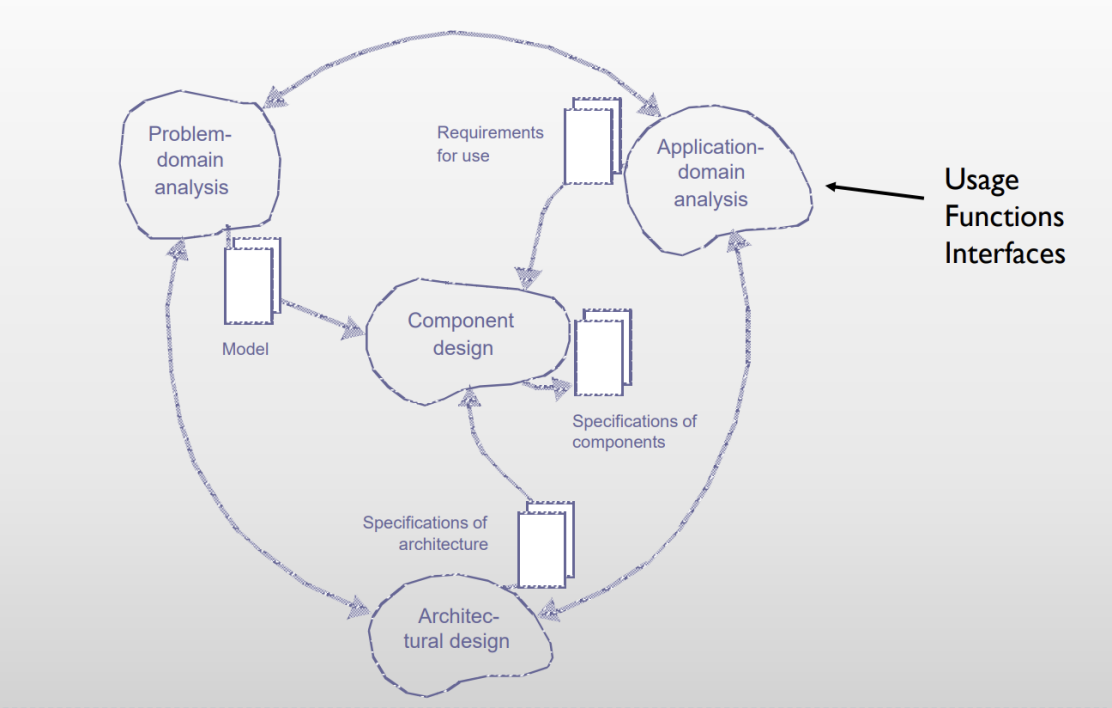


Figure 5.1: Application Domain Analysis: Activities

Chapter 6

Usage (OOAD P. 121)

Purpose	• To determine how actors interact with a system.
Concepts	• Actor: An abstraction of users or other systems that interact with the target system. • Use case: A pattern for interaction between the system and actors in the application domain.
Principles	• Determine the application domain with use cases. • Evaluate use cases in collaboration with users. • Assess social changes in the application domain.
Result	• Descriptions of all use cases and actors, also known as actor tables.

6.1 Use cases

Focus on interaction between users and the system.

Actor **-** *An abstraction of users or other systems that interact with the target system.*

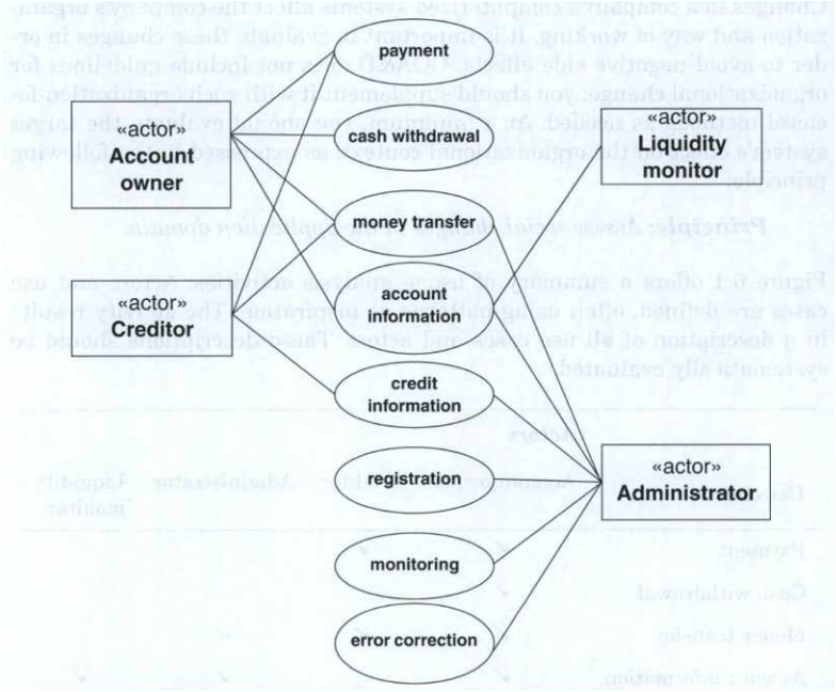
Actors are an abstraction of people and other systems that activates a target system function.
The complete set of use cases determines all uses of the target system within the application domain.

Use case - *A pattern for interaction between the system and actors in the application domain.*

Principle - *Determine the application domain with use cases.*

	Actors			
Use cases	Account owner	Creditor	Administrator	Liquidity monitor
Payment	✓	✓		
Cash withdrawal	✓			
Money transfer	✓	✓	✓	
Account information	✓		✓	✓
Credit information		✓	✓	
Registration			✓	
Monitoring			✓	
Error correction			✓	

Figure 6.2: Actor table for the automatic payment system



6.2 Find Actors and Use Cases

Criterion for determening different actors is a dissimilarity of roles. If multiple roles appear the same to the system, they should be consolidated.

Describe actors

Account owner

Goal: person who owns an account. The account owner's basic need is to make payments with their plastic card.

Characteristics: The system's users include many account owners, with different levels of experience and sophistication.

Examples: Account Owner A feels insecure about using a plastic card as a form of payment. Owner A originally got a card be cause it was the only way he could get an ID card for his checks. Owner A only withdraws money from the ATM in emergency situations.

Account Owner B is technologically curious and she uses the system often, optimally, and to the limit of its abilities. B has never had major problems in understanding the system's possibilities, and also examines possibilities that are not obviously accessible.

Describe Use Cases

Can be described with statechart diagram for a good overview or a use-case specification for more details. It should make use of abstraction, the goal is to collect many possible ways of using the traget system in a few well-chosen use cases.

How to Select Work Tasks

A work task uses a use a case, and in turn is used by the actor.

Help to select
What tasks exist in the application domain?
What is the division of labor?
How are the different tasks delimited?
Describe the work tasks:

- Name and content

- Purpose
- How is it assigned?
- Who perfomed it?
- Relationship and other tasks
- Result

Chapter 7

Functions (OOAD P. 139)

Purpose	• To determine the system's information processing capabilitites.
Concepts	• Function: A facility for making a model useful for actors.
Principles	• Udentify all functions. • Specify only complex funtions. • Check consistency with use cases and the model.
Result	• A complete list of functions with specification of complex functions.

Functions focus on what the system can do to assist actors in their work. When determining requirement for functions:

- What is the system going to do?

7.1 System Functions (OOAD p. 139)

Function - *A facility for making a model useful for actors.*

From an analytical perspective a function represents the intent of a system. A function is activated, executed and provides a result. The execution can change the state of a model's components or create a reaction in application or problem domain.

Function Types (OOAD p. 140)

Update functions are activated by a problem-domain event and result in a change in the model's state.

Signal functions are activated by a change in the m odel's state and result in a reaction in the context; this reaction might be a display to the actors in the application domain, or a direct intervention in the problem domain.

Read functions are activated by a need for information in an actor's work task and result in the system displaying relevant parts of the model.

Compute functions are activated by a need for information in an actor's work task and consist o f a computation involving information provided by the actor or the model; the result is a display of the computation's result.

Functions are not always pure and can be mixture of the above.

Analysing Functions (OOAD p. 143)

Principle - *Identify all functions.*

Principle - *Specify only complex functions.*

Principle - *Check consistency with use cases and the model.*

7.2 Find Functions

There are two essential aspects when finding functions:

- Consider the sources for identifying functions. Where do the systems' function requirements come from.

- Consider the level of detail. How detailed should the description of a function be. How specific or general should defining individual functions be?

The sources are problem-domain description expressed by it's classes and events. Also the application domain expressed by its use cases.

- Classes typically gives read and update functions.
- Events typically gives update functions.
- Use cases gives all functions.

Functions must be described enough for both and overview of total functionality, but also a basis for an agreement between developer and users.

The result should be a list functional requirements for the system.

7.3 Principles

Identify all functions. One of the main purposes of the analysis is to determine the level of ambition for the target system. The complete list of functions is an important element in achieving this.

Specify only complex functions. We recommend that you describe the functions briefly and informally in a list. However, it may sometimes be necessary to specify certain functions in detail in order to understand them and assess their complexity.

Check consistency with use cases and the model. The list of functions must be consistent with the list of use cases and the m odel's classes and events. Checking this can reveal insufficient analysis.