NÕUETE MODELLEERIMINE
(REQUIREMENTS MODELLING)
The Viewpoint Framework

<table>
<thead>
<tr>
<th>Viewpoint aspect</th>
<th>Abstraction layer</th>
<th>Analysis</th>
<th>Design</th>
<th>Prototyping</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interaction</td>
<td>Role models and organization model</td>
<td>Interaction sequence diagrams</td>
<td>Interaction prototyping</td>
</tr>
<tr>
<td></td>
<td>Information</td>
<td>Domain model</td>
<td>Class diagrams</td>
<td>Information prototyping</td>
</tr>
<tr>
<td></td>
<td>Behaviour</td>
<td>Goal models and scenarios</td>
<td>State diagrams</td>
<td>Behaviour prototyping</td>
</tr>
</tbody>
</table>
Three Perspectives on a Solution

• **Information**
  – Information to be managed by the system
  – Static aspects – as entities, relationships between entities, attributes and attribute types

• **Behaviour**
  – States in which the components of the system can be
  – Allowed transitions between the states

• **Interaction**
  – Interactions between the components of the system
Goals, Scenarios and Solution-oriented Requirements

Level of agreement

Different views

Consolidated views

Goals

Scenarios

Solution oriented requirements

Amount of (solution) detail

Few details

All relevant details
Goal-Scenario coupling

... initiate and influence the definition of ...

Goals

... illustrate satisfaction...

Scenarios

... lead to the identification of new ...

... classify ...

... lead to revision of ...
Key Relationships

- **Goals**
- **Scenarios**

Elicitation, refinement, and validation of solution-oriented requirements

Refinement of existing and elicitation of new goals and scenarios
What is Modelling?
What is Modelling?
Modelling...

• Modelling can guide elicitation:
  – It can help you figure out what questions to ask
  – It can help to surface hidden requirements
    • i.e. does it help you to ask the right questions?

• Modelling can provide a measure of progress:
  – Completeness of the models -> completeness of the elicitation (?)
    • i.e. if we’ve filled in all the pieces of the models, are we done?

• Modelling can help to uncover problems
  – Inconsistency in the models can reveal interesting things...
    • e.g. conflicting or infeasible requirements
    • e.g. confusion over terminology, scope, etc
    • e.g. disagreements between stakeholders

• Modelling can help us to check our understanding
  – Reason over the model to understand its consequences
    • Does it have the properties we expect?
  – If possible, animate the model to help us visualise/validate the requirements
Systems involve a lot of modelling

- A model is more than just a description
  - it has its own phenomena, and its own relationships among those phenomena.
- The model is only useful if the model’s phenomena correspond in a systematic way to the phenomena of the domain being modelled.

For every B, at least one P exists such that R(P, B)

**Source:** Adapted from Jackson, 1995, p120-122
“It’s only a model”

• There will always be:
  – phenomena in the model that are not present in the application domain
  – phenomena in the application domain that are not in the model

• A model is never perfect
  – “If the map and the terrain disagree, believe the terrain”
  – Perfecting the model is not always a good use of your time...

Source: Adapted from Jackson, 1995, p124-5
Don’t forget what we’re modelling

• During **analysis**
  – we want to know about the application domain and the requirements
  – ...so we develop a course-grained model to show where responsibilities are, and how entities interact
    • Our models show interaction pathways (“acquaintances”) or perhaps a message being passed, but we don’t worry too much about the contents of each message
    • To keep things clear, use icons to represent external objects and actors, and boxes to represent system objects

• During **design**
  – we want to say how the software should work
  – ... so we develop fine-grained models to show exactly what will happen when the system runs
    • e.g. show the precise details of each method call
# The Viewpoint Framework

<table>
<thead>
<tr>
<th>Viewpoint aspect</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstraction layer</strong></td>
<td>Interaction</td>
</tr>
<tr>
<td>Analysis</td>
<td>Role models and organization model</td>
</tr>
<tr>
<td>Design</td>
<td>Interaction sequence diagrams</td>
</tr>
<tr>
<td>Prototyping</td>
<td>Interaction prototyping</td>
</tr>
</tbody>
</table>
## The Viewpoint Framework

<table>
<thead>
<tr>
<th>Viewpoint aspect</th>
<th>Abstraction layer</th>
<th>Analysis</th>
<th>Design</th>
<th>Prototyping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>Interaction</td>
<td>Role models and organization model</td>
<td>Interaction sequence diagrams</td>
<td>Interaction prototyping</td>
</tr>
<tr>
<td>Information</td>
<td>Domain model</td>
<td></td>
<td>Class diagrams</td>
<td>Information prototyping</td>
</tr>
<tr>
<td>Behaviour</td>
<td>Goal models and scenarios</td>
<td></td>
<td>State diagrams</td>
<td>Behaviour prototyping</td>
</tr>
</tbody>
</table>
# The Viewpoint Framework

<table>
<thead>
<tr>
<th>Viewpoint aspect</th>
<th>Analysis</th>
<th>Design</th>
<th>Prototyping</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstraction layer</strong></td>
<td>Interaction and organization model</td>
<td>Interaction sequence diagrams</td>
<td>Interaction prototyping</td>
</tr>
<tr>
<td><strong>Information</strong></td>
<td>Domain model</td>
<td>Class diagrams</td>
<td>Information prototyping</td>
</tr>
<tr>
<td><strong>Behaviour</strong></td>
<td>Goal models and scenarios</td>
<td>State diagrams</td>
<td>Behaviour prototyping</td>
</tr>
</tbody>
</table>
Classes

- A class describes a group of objects with
  - similar properties (attributes);
  - common behaviour (operations);
  - common relationships to other objects;
  - and common meaning (“semantics”)

Examples
- employee: has a name, employee# and department; an employee is hired, and fired; an employee works in one or more projects
Finding classes

• Finding classes from source data:
  – Look for nouns and noun phrases in the descriptions of the problem by the stakeholders:
    • include in the model if they explain the nature or structure of information in the application

• Finding classes from other sources:
  – Reviewing background information
  – Users and other stakeholders
  – Analysis patterns

• It’s better to include many candidate classes at first:
  – You can always eliminate them later if they turn out not to be useful
  – Explicitly deciding to discard classes is better than just not thinking about them
Selecting classes

• **Discard classes for concepts which:**
  – Are beyond the scope of the analysis;
  – Refer to the system as a whole;
  – Duplicate other classes;
  – Are too vague or too specific
    • e.g. have too many or too few instances

• External entities that produce or consume information essential to the system should be included as classes
Selecting classes

- Coad & Yourdon’s criteria:
  - **Retained information**
    - Will the system need to remember information about this class of objects?
  - **Needed Services**
    - Do objects in this class have identifiable operations that change the values of their attributes?
  - **Multiple Attributes**
    - If the class only has one attribute, it may be better represented as an attribute of another class
  - **Common Attributes**
    - Does the class have attributes that are shared with all instances of its objects?
  - **Common Operations**
    - Does the class have operations that are shared with all instances of its objects?
Objects vs. Classes

• The instances of a class are called objects:
  - Two different objects may have identical attribute values (like two people with identical name and address)

• Objects have associations with other objects:
  - E.g. Fred_Bloggs:employee is associated with the KillerApp:project object
  - But we will capture these relationships at the class level (why?)
Associations

- Objects do not exist in isolation from one another:
  - A relationship represents a connection among things

- Types of relationships:
  - Association
  - Aggregation and Composition
  - Generalization

- Class diagrams show classes and their relationships:
Multiplicity

- **Optional (0 or 1):** \(0..1\)
- **Exactly one:** \(1 = 1..1\)
- **Zero or more:** \(0..* = *\)
- **One or more:** \(1..*\)
- **A range of values:** \(1..6\)
- **A set of ranges:** \(1..3, 7..10, 15, 19..*\)
Class association

Multiplicity
A client has exactly one staff member as a contact person

Multiplicity
A staff member has zero or more clients on His/her clientList

Name of the association

Direction
The “liaises with” association should be read in this direction

Role
The staff member’s role in this association is as a contact person

Role
The clients’ role in this association is as a clientList
Association classes

- Sometimes the association is itself a class:
  - …because we need to retain information about the association
  - …and that information doesn’t naturally live in the classes at the ends of the association:
    - E.g., a “title” is an object that represents information about the relationship between an owner and her car
Aggregation and Composition

• Aggregation:
  – This is the “Has-a” or “Whole/part” relationship

• Composition:
  – Strong form of aggregation that implies ownership:
    • if the whole is removed from the model, so is the part
    • the whole is responsible for the disposition of its parts
Generalisation

- Subclasses **inherit** attributes, associations, & operations from the superclass
- A subclass may override an inherited aspect
- Superclasses may be declared {abstract}, meaning they have no instances:
  - Implies that the subclasses cover all possibilities
Generalisation

• **Look for generalisations in two ways:**
  – **Top Down:**
    • You have a class, and discover it can be subdivided
  – **Bottom Up:**
    • You notice similarities between the classes you have identified

• **But don’t generalise just for the sake of doing it:**
  – Be sure that everything about the superclass applies to the subclasses
  – Don’t add subclasses or superclasses that are not relevant to your analysis
Class diagram

- Organ
  - natural
  - implant
  - donor

- Eye
  - colour
  - diameter
  - correction

- Kidney
  - operational

- Heart
  - normal bpm
  - blood type

- Patient
  - name
  - date of birth
  - height
  - weight

- In-patient
  - room
  - bed
  - physician

- Out-patient
  - last visit
  - next visit
  - physician

{incomplete, disjoint}
# The Viewpoint Framework

<table>
<thead>
<tr>
<th>Viewpoint aspect</th>
<th>Interaction</th>
<th>Information</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstraction layer</strong></td>
<td>Interaction</td>
<td>Information</td>
<td>Behaviour</td>
</tr>
<tr>
<td>Analysis</td>
<td>Role models and organization model</td>
<td>Domain model</td>
<td>Goal models and scenarios</td>
</tr>
<tr>
<td>Design</td>
<td>Interaction sequence diagrams</td>
<td>Class diagrams</td>
<td>State diagrams</td>
</tr>
<tr>
<td>Prototyping</td>
<td>Interaction prototyping</td>
<td>Information prototyping</td>
<td>Behaviour prototyping</td>
</tr>
</tbody>
</table>
# The Viewpoint Framework

<table>
<thead>
<tr>
<th>Viewpoint aspect</th>
<th>Abstraction layer</th>
<th>Analysis</th>
<th>Design</th>
<th>Prototyping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>Interaction</td>
<td>Role models and organization model</td>
<td>Interaction sequence diagrams</td>
<td>Interaction prototyping</td>
</tr>
<tr>
<td>Information</td>
<td>Information</td>
<td>Domain model</td>
<td>Class diagrams</td>
<td>Information prototyping</td>
</tr>
<tr>
<td>Behaviour</td>
<td>Behaviour</td>
<td>Goal models and scenarios</td>
<td>State diagrams</td>
<td>Behaviour prototyping</td>
</tr>
</tbody>
</table>
Getting objects to behave

- All objects have “state”:
  - The object either exists or it doesn’t
  - If it exists, then it has a value for each of its attributes
  - Each possible assignment of values to attributes is a “state”
    - (and non-existence is a state, although we normally ignore it)

- A stack object:
State diagrams

• **States:**
  – “Interesting” configurations of the values of an object’s attributes
  – States may be “on” or “off” at any given moment

• **Transitions:**
  – Are enabled when the state is “on”; disabled otherwise
  – Every transition has an **event** that acts as a trigger
  – A transition may also have a condition (or **guard**)
  – A transition may also cause some action to be taken

• **Events:**
  – Occurrences of stimuli that can trigger an object to change its state
  – Determine when transitions can fire
Object states

<table>
<thead>
<tr>
<th>:person</th>
<th>❉  man  ♀</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td></td>
</tr>
<tr>
<td>havebirthday()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>child</th>
</tr>
</thead>
<tbody>
<tr>
<td>havebirthday()</td>
</tr>
<tr>
<td>[age &lt; 18]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>adult</th>
</tr>
</thead>
<tbody>
<tr>
<td>havebirthday()</td>
</tr>
<tr>
<td>[age = 18]</td>
</tr>
<tr>
<td>havebirthday()</td>
</tr>
<tr>
<td>[age &lt; 65]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>senior</th>
</tr>
</thead>
<tbody>
<tr>
<td>havebirthday()</td>
</tr>
<tr>
<td>[age = 65]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>:person</th>
</tr>
</thead>
<tbody>
<tr>
<td>dateOfBirth</td>
</tr>
<tr>
<td>dateOfDeath</td>
</tr>
<tr>
<td>recordBirth()</td>
</tr>
<tr>
<td>setDOB()</td>
</tr>
<tr>
<td>recordDeath()</td>
</tr>
<tr>
<td>setDateofDeath()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordBirth()</td>
</tr>
<tr>
<td>/setDOB()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>child</th>
</tr>
</thead>
<tbody>
<tr>
<td>when</td>
</tr>
<tr>
<td>[thisyear-birthyear&gt;18]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>adult</th>
</tr>
</thead>
<tbody>
<tr>
<td>when</td>
</tr>
<tr>
<td>[thisyear-birthyear&gt;65]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>senior</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordDeath()</td>
</tr>
<tr>
<td>/setDateofDeath()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>deceased</th>
</tr>
</thead>
</table>

34
States

• **A state represents a time period during which:**
  – A predicate is true:
    • e.g. (budget - expenses) > 0
  – An action is being performed, or an event is awaited:
    • e.g., checking inventory for items to be ordered
    • e.g., waiting for arrival of a missing item that has been ordered

• **States can have associated activities:**
  – **do/activity:**
    • carries out some activity for as long as the state is “on”
  – **entry/action and exit/action:**
    • carry out the action whenever the state is entered or exited
  – **include/stateDiagramName:**
    • “calls” another state diagram, allowing state diagrams to be nested
Events

• **Events:**
  – Must be relevant to the system (or object) being modelled
  – Must be **amenable to modelling as an instantaneous occurrence** (from the system’s point of view):
    • E.g. completing an assignment, failing an exam, a system crash
  – Are implemented by message passing in object-oriented design

• **Four types of events:**
  – **Change events** occur when a condition becomes true
    • e.g. when \([\text{balance} < 0]\)
  – **Call events** occur when an object receives a call for one of its operations to be performed
  – **Signal events** occur when an object receives an explicit (real-time) signal
  – **Elapsed-time events** mark the passage of a designated period of time
    • e.g. after \([10 \text{ seconds}]\)
What does the model mean?

• Finite State Machines:
  – There are a finite number of states (all attributes have finite ranges):
    • E.g. imagine a stack with max length = 3

  – The model specifies a set of traces:
    • E.g. new();Push();Push();Top();Pop();Push()…
    • E.g. new();Push();Pop();Push();Pop()…
    • There may be an infinite number of traces (and traces may be of infinite length)

  – The model excludes some behaviours:
    • E.g. no trace may have more Pops than Pushes
    • E.g. no trace may have more than 3 Pushes without a Pop in between
The Viewpoint Framework

<table>
<thead>
<tr>
<th>Viewpoint aspect</th>
<th>Interaction</th>
<th>Information</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstraction layer</strong></td>
<td><strong>Interaction</strong></td>
<td>Information</td>
<td>Behaviour</td>
</tr>
<tr>
<td>Analysis</td>
<td>Role models and organization model</td>
<td>Domain model</td>
<td>Goal models and scenarios</td>
</tr>
<tr>
<td>Design</td>
<td>Interaction sequence diagrams</td>
<td>Class diagrams</td>
<td>State diagrams</td>
</tr>
<tr>
<td>Prototyping</td>
<td>Interaction prototyping</td>
<td>Information prototyping</td>
<td>Behaviour prototyping</td>
</tr>
</tbody>
</table>
# The Viewpoint Framework

<table>
<thead>
<tr>
<th>Viewpoint aspect</th>
<th>Interaction</th>
<th>Information</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstraction layer</strong></td>
<td>Role models and organization models</td>
<td>Domain model</td>
<td>Goal models and scenarios</td>
</tr>
<tr>
<td><strong>Analysis</strong></td>
<td>Interaction sequence diagrams</td>
<td>Class diagrams</td>
<td>State diagrams</td>
</tr>
<tr>
<td><strong>Design</strong></td>
<td>Interaction prototyping</td>
<td>Information prototyping</td>
<td>Behaviour prototyping</td>
</tr>
<tr>
<td><strong>Prototyping</strong></td>
<td>Interaction prototyping</td>
<td>Information prototyping</td>
<td>Behaviour prototyping</td>
</tr>
</tbody>
</table>
Sequences of events

• **Objects “own” information and behaviour:**
  – They have attributes and operations relevant for their *responsibilities*
  – They don’t “know” about other objects’ information, but can ask for it
  – Objects have to collaborate...
    • ...by sending messages to one another to invoke each others’ operations
  – Objects can only send messages one to another if they “know” each other...
    • ...if there is an association between them

• **(Interaction) sequence diagrams:**
  – Sequence diagrams show how objects interacts with one another
  – Sequence diagrams...
    • ...should remain easy to read and understand
    • ...do not include complex control logic
Example

Initiator: Person

Staff: Person

Scheduler: Person

Participant: Person

Call()

Respond()

What’s up?()

Give mtg details()

[for all participants] *Inform()

Acknowledge()

[for all participants] *Remind()

Prompt()

Qshow schedule()

[decision=OK] ScheduleOk’ed()

[for all participants] *Inform()
Another example
The Viewpoint Framework

<table>
<thead>
<tr>
<th>Viewpoint aspect</th>
<th>Abstraction layer</th>
<th>Analysis</th>
<th>Design</th>
<th>Prototyping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>Role models and organization model</td>
<td>Domain model</td>
<td>Interaction sequence diagrams</td>
<td>Information prototyping</td>
</tr>
<tr>
<td>Information</td>
<td></td>
<td>Goal models and scenarios</td>
<td>Class diagrams</td>
<td>Behaviour prototyping</td>
</tr>
<tr>
<td>Behaviour</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>