L9 - Requirements analysis models

CS3028 - Principles of Software Engineering

Ernesto Compatangelo

Department of Computing Science

UNIVERSITY OF ABERDEEN

9.1 Reminding past issues and mapping them to current topics

Where are we now?

Software development paradigms

⇒ The Unified Process (UP) paradigm

⇒ UP phases and UP disciplines (activities) within each phase

⇒ Inception (first UP phase)

⇒ Elaboration (second UP phase)

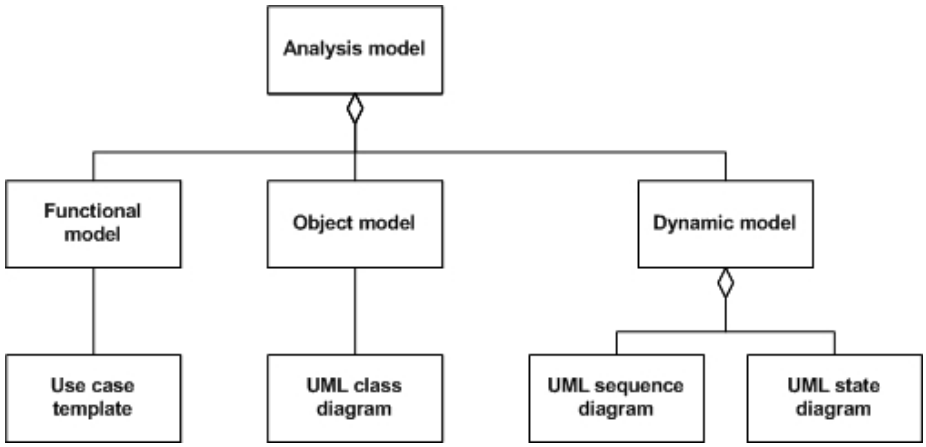
⇒ Elaboration requirements

⇒ Requirements analysis models (main concepts)

⇒ . . . . . .

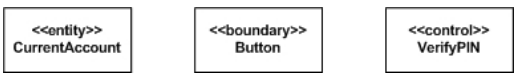
9.2 Requirements analysis concepts

A schema of requirements analysis models



Behond the functional model: the object model

- Focuses on **concepts** manipulated by the system (NOT on objects in OO-programming terms!), their properties, and their relationships
- Is depicted using **UML class diagrams** at the proper abstraction level, including classes, attributes, and operation names
- Represents a **visual dictionary** of user-visible main concepts
- Explicitly differentiates between entity, boundary, and control objects



9.3 Entity, boundary, and control objects

Entity, boundary, and control objects

- **Entity** objects represent persistent information tracked by the system
- **Boundary** objects represent interactions between actors and system. They model the user interface at a coarse level, without describing its visual aspect
- **Control** objects realise use cases providing the backbone for the flow of events as a sequence of (i) computations performed by and (ii) messages exchanged between control objects
- The above distinction is not always adequate/appropriate, but in most cases it helps to force the separation between objects belonging to **distinct architectural layers**
- Stereotypes are used in UML class modelling to denote these three distinct categories of requirements analysis objects

Heuristics for entity objects For entity objects, consider the following:

- Terms that developers or users must clarify to understand the use case
- Recurring nouns in the use cases
- Real-world entities that the system must track
- Real-world activities that the system must track
- Data sources or sinks (*e.g.*, `Printer`)

Heuristics for boundary objects For boundary objects, consider the following:

- User interface controls that the user needs to initiate a use case
- Forms that the users need to enter data into the system
- Messages that the system uses to respond to the user
- Identify 'actor terminals' to refer to the user interface under consideration when multiple actors are involved in a use case
- Do not model the visual aspects of the interface with boundary objects (user mock-ups are better suited for that)
- Always use end user's terms to describe interfaces; never use terms from the solution or implementation domains

Heuristics for control objects For control objects, consider the following:

- Identify one control object per use case
- Identify one control object per actor in a use case
- The life span of a control object should cover the extent of the use case or the extent of a user session.

  If it is difficult to identify the beginning and the end of a control object activation, the corresponding use case probably does not have well-defined entry and exit conditions.

9.4 UML analysis object relationships

The need for (UML) analysis object relationships

Identifying analysis 'objects' and modelling them as UML classes is a first step towards the creation of a comprehensive object model from requirements analysis. However, a diagram composed of non-connected, sparse classes is neither useful nor adequate.

The following structural features must be introduced to develop a class model that can be effectively transformed into a detailed design later on

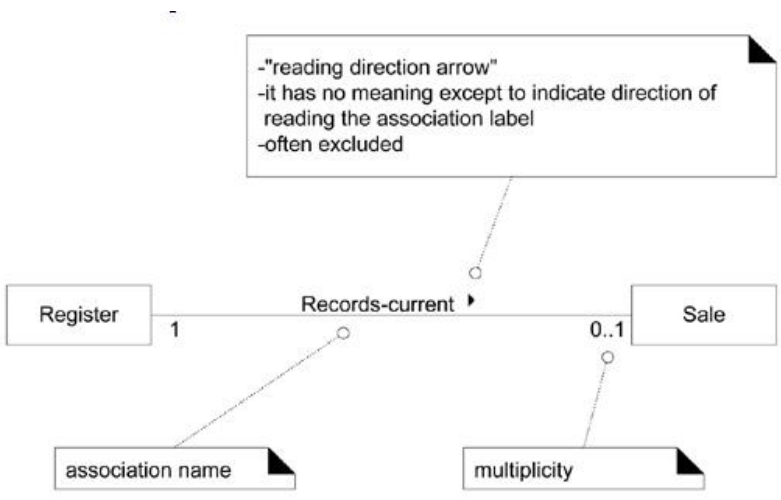
- **Associations**
- **Aggregation**
- **Attributes**
- **Generalisation/specialisation (inheritance)**

9.4.1 Associations

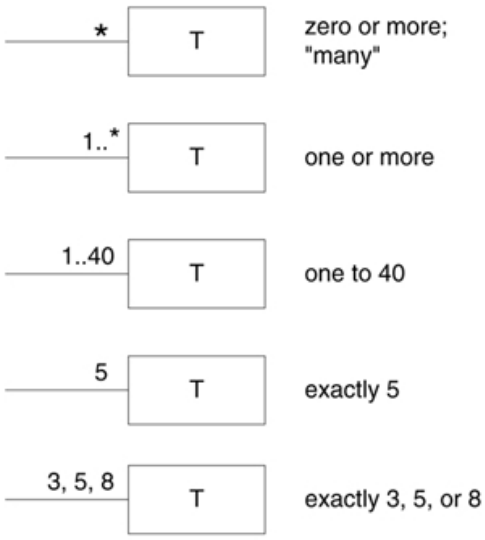
Associations

- An association is a relationship between classes (more precisely, instances of those classes) that indicates some meaningful and interesting connection.
- Associations worth noting usually imply knowledge of a relationship that needs to be preserved for some duration — from milliseconds to years, depending on context.
- An association is NOT a statement about data flows, database foreign key relationships, instance variables, or object connections in a software solution.
- Each association end is a **role**. Associations may optionally have:

Association example featuring all properties



Associations: multiplicity by example



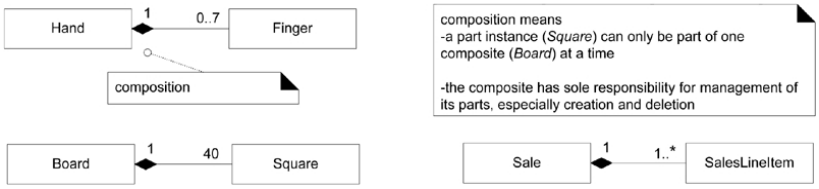
9.4.2 Aggregation

Aggregation

- An aggregation denotes a part-whole relationship between classes (more precisely, instances of those classes). connection.
- **Shared aggregation** loosely suggests a whole-part relationship between elements that can exist independently (as in ordinary associations). In UML, it has no meaningful distinct semantics w.r.t. a plain association, so it is just a 'modelling placebo'.
- **Composite aggregation** (a.k.a. composition) is a strong kind of whole-part aggregation. It implies that
  - an instance of the part (such as a Square) belongs to only one composite instance (such as one Board) at a time
  - the part must always belong to a composite (no free-floating Fingers)
  - the composite is responsible for the creation and deletion of its parts - either by itself creating/deleting the parts, or by collaborating with other objects.

Aggregation examples

- Model shared aggregations as normal associations
- Model composite aggregations as follows

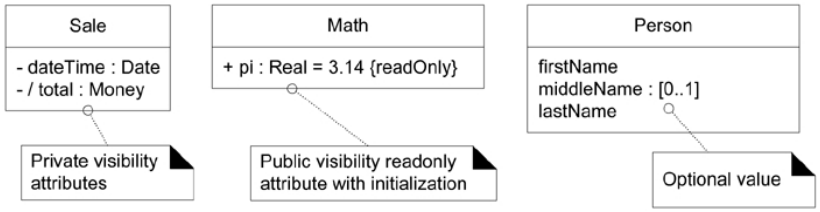


9.4.3 Atributes

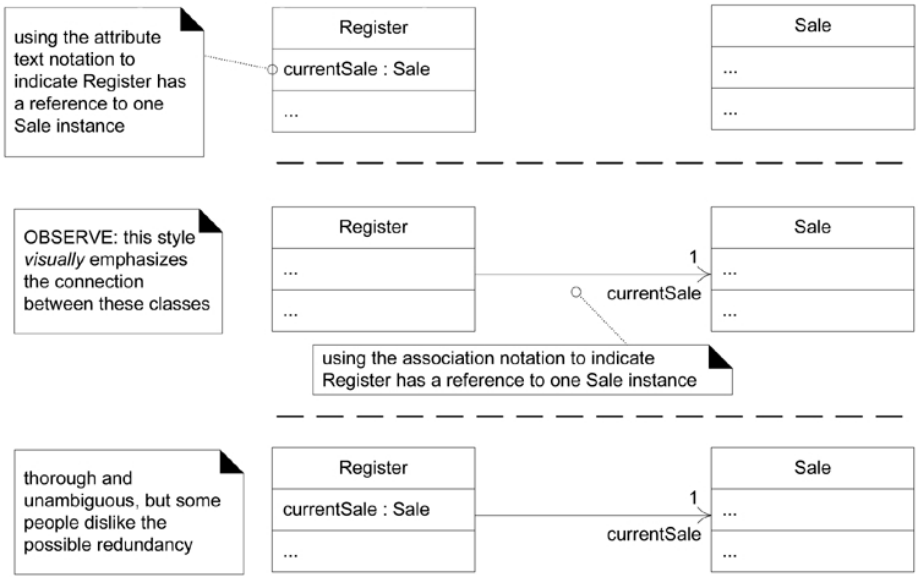
Attributes

- An attribute is a *named* property of an object referring to a logical data value. It is useful to identify those class properties needed to specify information about the scenarios under analysis.
- As a convention, most modelers will assume attributes have private visibility (-) unless shown otherwise.
- Attributes in an analysis model have *types* preferably drawn from the *concrete typeset* (Boolean, Date, Time, Number, Character, String)

Attribute examples



Attributes or associations?



Heuristics for attribute identification Consider the following:

- Examine possessive phrases
- Represent stored state as an attribute of the entity object
- Describe each attribute
- Do not represent an attribute as an object: use an association instead
- Do not waste time describing fine details before the object structure is stable

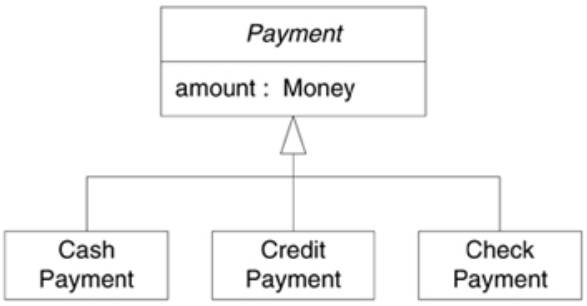
9.4.4 Generalisation, specialisation, inheritance

Generalisation/specialisation vs. inheritance

- **Generalisation** (UML): A taxonomic relationship deriving a more general class from more specific ones.
- **Specialisation** (UML): A taxonomic relationship deriving more specific classes from a more general one.
- Each instance of the more specific class is also an instance of the more general class, but not vice versa. Hence, each elements in the subclass inherits (and at most further specialises) ALL the features of the superclass.

  This is the ER view of the world, and also the view in the Knowledge representation (AI) community)
- **Inheritance** (OOP world): some of the features are inherited, some others may be not and the inherited ones can be overridden (which is – sort of – not allowed in UML)

Generalisation/specialisation example



9.5 Preparing for the topic ahead

Next lecture...

Other requirements analysis models

We will focus on dynamic models, :

- Requirements event modelling
- Event-driven scenarios
- System and object sequence diagrams
- State machine diagrams