The UML Modelling Language

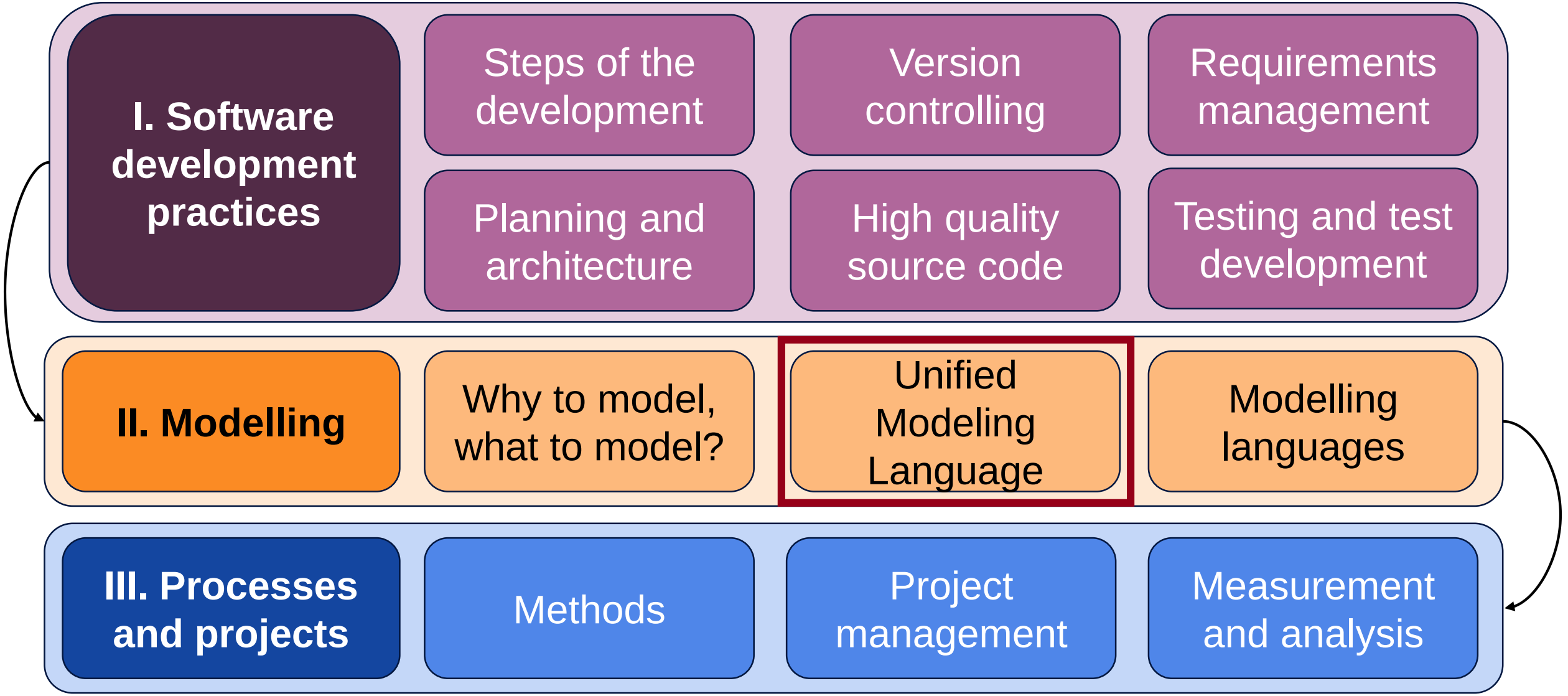
HUSZERL Gábor
huszerl@mit.bme.hu

MŰEGYETEM 1782
BME VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
Méréstechnika és Információs Rendszerek Tanszék
ftsrg Critical Systems Research Group

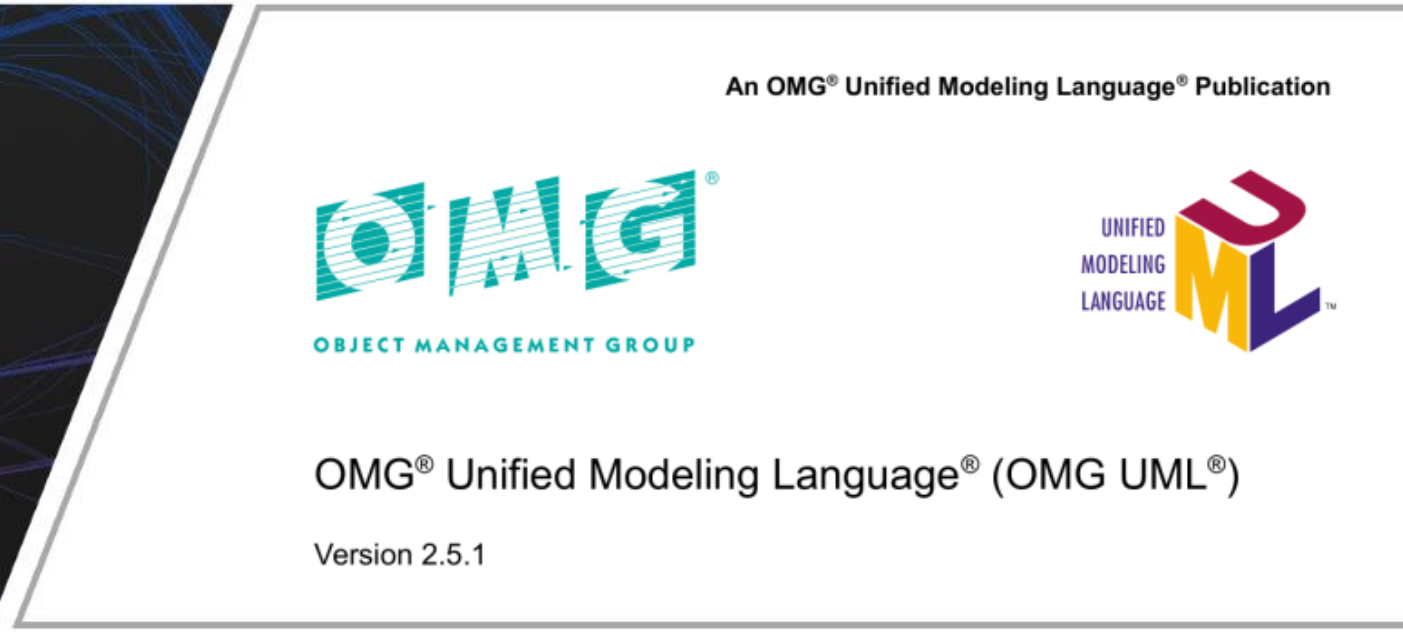
Learning Outcomes

- At the end of the lecture the students are expected to be able to
- (K1) identify the features of the UML modelling language,
- (K3) use UML use cases to define functions.

Further Topics of the Subject



Unified Modeling Language (UML)



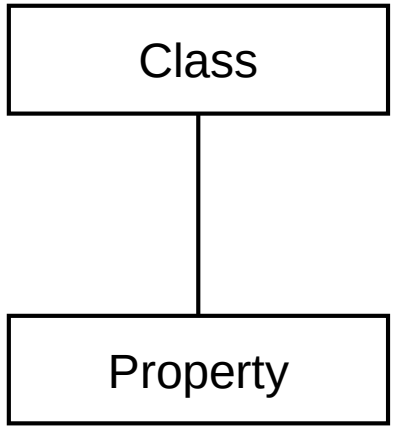
Unified Modeling Language (UML)

- Standardized language for describing software systems
  - Visual, general purpose, common language
- Maintained by the Object Management Group (OMG)
- History
  - The 90's: many different notations and methods
  - 1997: UML 1.0 proposal (G. Booch, I. Jacobsen, J. Rumbaugh and others)
  - 2005: UML 2.0, significantly renewed, common semantic basis
  - 2017: UML 2.5, simplified description, in one single document
  - 2011- defining precise semantics (fUML, PSCS, PSSM…)

Model and Diagram

A diagram shows some of the elements of the model only

UML language



Model

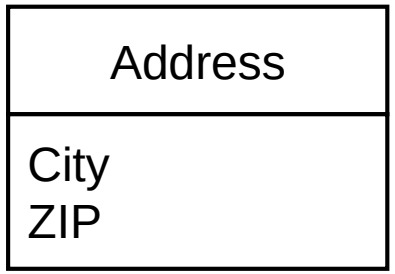
```
Person [Class]
  Name [Property]
  Age [Property]

Address [Class]
  City [Property]
  ZIP [Property]

Country [Class]
  Capital [Property]
```

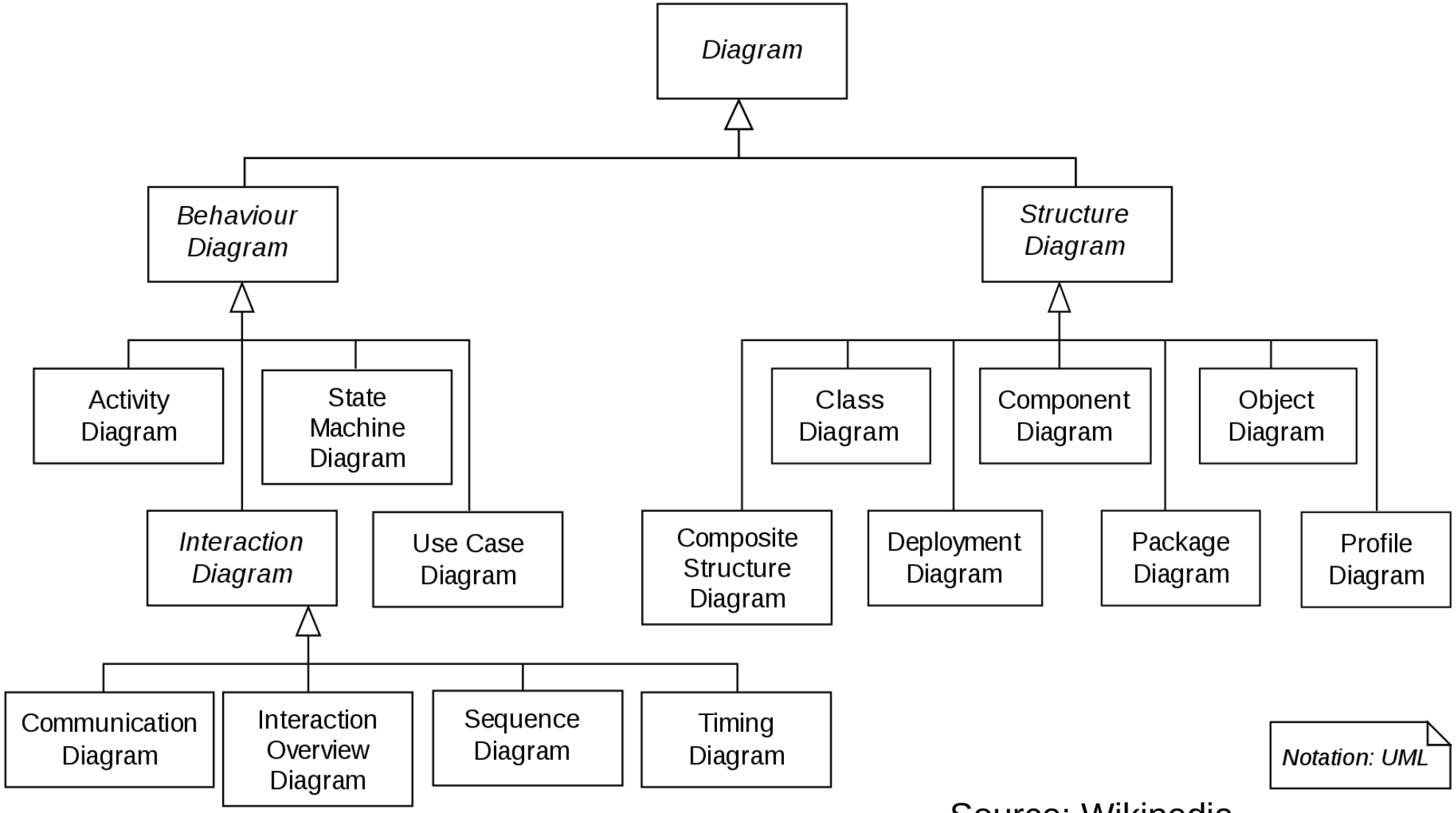
Diagram 1

Diagram 2



system ≠ modell ≠ diagram

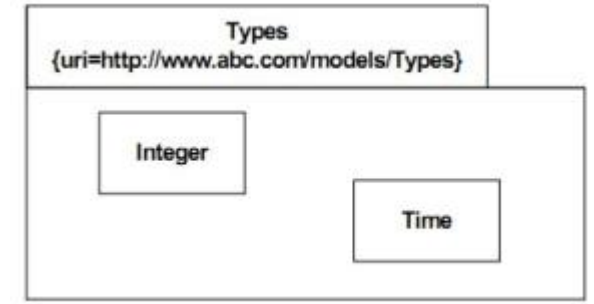
UML Diagram Types

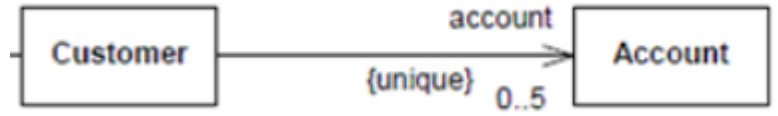


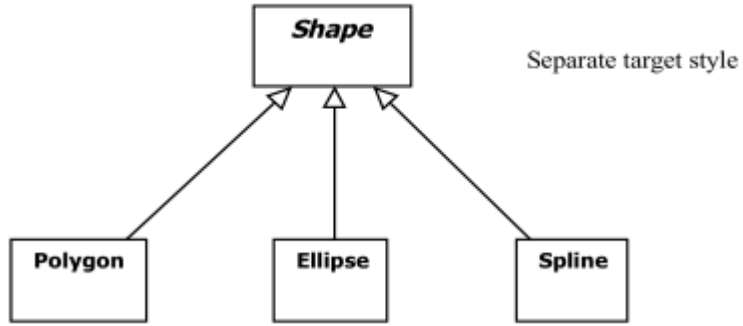
Source: Wikipedia

UML Basic Terms and Elements

- Element, Relationship: most general, abstract elements
- Package: to organize and group model elements
- MultiplicityElement: element that can have multiple instances (lower and upper bound)
- Classifier: classifies instances into classes, they may have so called Features
- Generalization: a directed relationship that organises other elements into a hierarchy

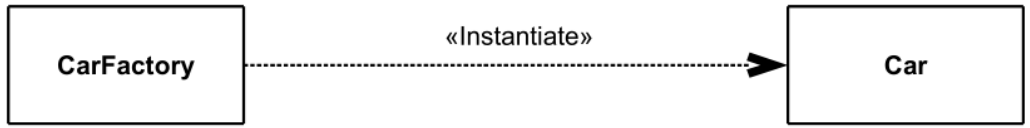


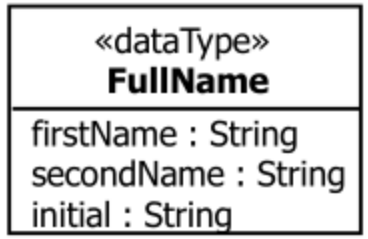


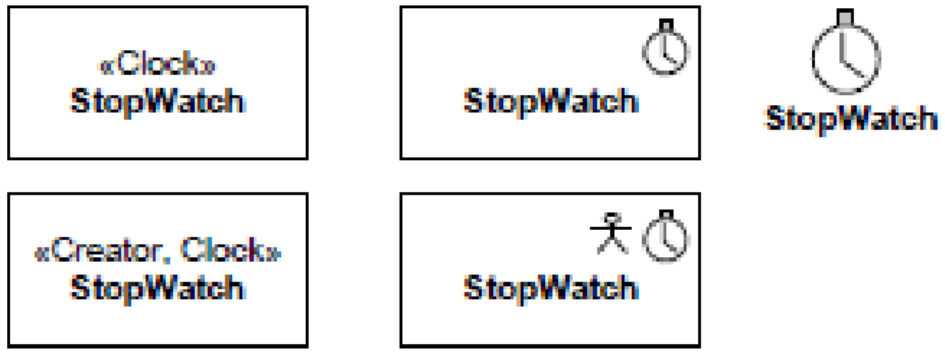


UML Stereotypes

- Extending the UML modelling elements (meta-classes)
  - Built-in UML stereotypes (e.g. component, interface)
  - Custom extensions as part of a profile
  - Extending the terminology
- Notation: a name between « » characters (guillemets)
- Examples:







The UML Specification

9 Classification

9.1 Summary

Classification is an important technique for organization. This clause specifies concepts relating to classification. The core concept is Classifier, an abstract metaclass whose concrete subclasses are used to classify different kinds of values. The other metaclasses in this clause represent the constituents of Classifiers, models of how Classifiers are instantiated using InstanceSpecifications, and various relationships between all of these concepts.

9.2 Classifiers

9.2.1 Summary

A Classifier represents a classification of instances according to their Features. Classifiers are organized in hierarchies by Generalizations. RedefinableElements may be redefined in the context of Generalization hierarchies.

9.2.2 Abstract Syntax

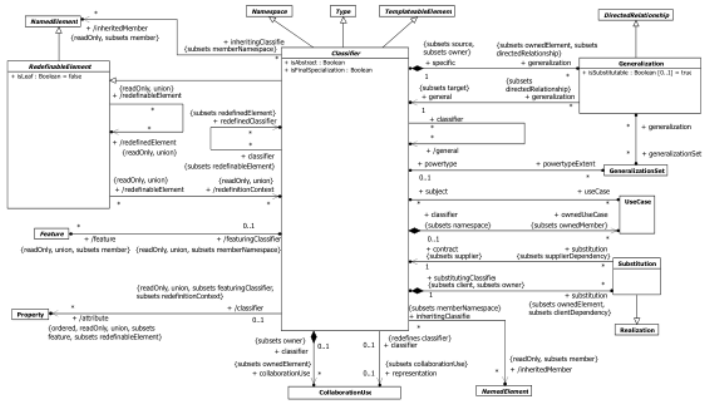


Figure 9.1 Classifiers

9.2.3 Semantics

9.2.3.1 Classifiers

A Classifier has a set of [illegible] Properties called the attributes of the Classifier. Each of the Features is a me[illegible]

The [illegible]

A Cl[illegible]

Unifi[illegible]

99

Overview of the modelling elements (abstract syntax)

A Classifier may own CollaborationUses that relate the Classifier to Collaborations. The Collaborations describes aspects of this Classifier. See 11.7 Collaborations.

A Classifier may own UseCases. See 18.1 Use Cases.

9.2.3.2 Generalization

Generalizations define generalization/specialization relationships between Classifiers. Each Generalization relates a specific Classifier to a more general Classifier. Given a Classifier, the transitive closure of its general Classifiers is often called its *generalizations*, and the transitive closure of its specific Classifiers is called its *specializations*. The immediate generalizations are also called the Classifier's parents, and where the Classifier is a Class, its superClasses (see 11.4).

NOTE. The concept of parent (a generalization relationship between Classifiers) is unrelated to the concept of owner (a composition relationship between instances).

An instance of a Classifier is also an (indirect) instance of each of its generalizations. Any Constraints applying to instances of the generalizations also apply to instances of the Classifier.

When a Classifier is generalized, certain members of its generalizations are *inherited*, that is they behave as though they were defined in the inheriting Classifier itself. For example, an inherited member that is an attribute may have a value or collection of values in any instance of the inheriting Classifier, and an inherited member that is an Operation may be invoked on an instance of the inheriting Classifier.

The set of members that are inherited is called the inheritedMembers. Unless specified differently for a particular kind of Classifier, the inheritedMembers are members that do not have private visibility.

Type conformance means that if one Type conforms to another, then any instance of the first Type may be used as the value of a TypedElement whose type is declared to be the second Type. A Classifier is a Type, and conforms to itself and to all of its generalizations.

The isAbstract property of Classifier, when true, specifies that the Classifier is abstract, i.e., has no direct instances: every instance of the abstract Classifier shall be an instance of one of its specializations.

If one Classifier (the parent) generalizes another (the child) it is not necessarily the case that instances of the child are substitutable for instances of the parent under every possible circumstance. For example, Circle may be defined as a specialization of Ellipse, and its instances would be substitutable in every circumstance involving accessing the properties of an Ellipse. However, if Ellipse were to define a stretch behavior that modifies the length of its major axis only, then a Circle object would be unable to implement such a behavior. The isSubstitutable property may be used to indicate whether the specific Classifier can be used in every circumstance that the general Classifier can be used.

9.2.3.3 Redefinition

Any member (that is a kind of RedefinableElement) of a generalization of a specializing Classifier may be redefined instead of being inherited. Redefinition is done in order to augment, constrain, or override the redefined member(s) in the context of instances of the specializing Classifier. When this occurs, the redefining member contributes to the structure or behavior of the specializing Classifier in place of the redefined member(s); specifically, any reference to a redefined member in the context of an instance of the specializing Classifier shall resolve to the redefining member (note that to avoid circularity "any reference" here excludes the redefinedElement reference itself).

The Classifier from which the memb[illegible]e redefined is called the redefinitionContext. Although in the metamodel redefinitionContext has the multiplic[illegible] are no cases in the UML specification where there is more than one redefinitionContext. The redefinitio[illegible]ed for each kind of RedefinableElement; it is often, but not always, the owner of the member.

A redefining element sh[illegible]edefinableElement it redefines, but it may add specific constraints or other [illegible] the gener[illegible]

One [illegible] redef[illegible]

100 [illegible].5.1

Description of the semantics (in natural language)

conflicts may occur. Instances that invoke a BehavioralFeature need to coordinate so that only one invocation to a target on any BehavioralFeature occurs at once.

- guarded
  Multiple invocations of a BehavioralFeature that overlap in time may occur to one instance, but only one is allowed to commence. The others are blocked until the performance of the currently executing BehavioralFeature is complete. It is the responsibility of the system designer to ensure that deadlocks do not occur due to simultaneous blocking.

- concurrent
  Multiple invocations of a BehavioralFeature that overlap in time may occur to one instance and all of them may proceed concurrently.

9.9.4 Classifier [Abstract Class]

9.9.4.1 Description

A Classifier represents a classification of instances according to their Features.

9.9.4.2 Diagrams

Classifiers, Classifier Templates, Features, Instances, Generalization Sets, Executable Nodes, Use Cases, Structured Classifiers, Classes, Associations, Components, Collaborations, State Machine Redefinition, DataTypes, Signals, Interfaces, Information Flows, Artifacts, Actions, Accept Event Actions, Object Actions

9.9.4.3 Generalizations

Namespace, Type, TemplateableElement, RedefinableElement

9.9.4.4 Specializations

Association, StructuredClassifier, BehavioredClassifier, DataType, Interface, Signal, InformationItem, Artifact

9.9.4.5 Attributes

- isAbstract : Boolean [1..1] = false
  If true, the Classifier can only be instantiated by instantiating one of its specializations. An abstract Classifier is intended to be used by other Classifiers e.g., as the target of Associations or Generalizations.

- isFinalSpecialization : Boolean [1..1] = false
  If true, the Classifier cannot be specialized.

9.9.4.6 Association Ends

- /attribute : Property [0..*]{ordered, union, subsets Classifier::feature, subsets A_redefinitionContext_redef[illegible]bleElement::redefinableElement} {opposite A_attribute_classifier::classifier}
  All of the Properties that [illegible]t (i.e., not inherited or imported) attributes of the Classifier.

- ♦ collaborationUse [illegible]e [0..*]{subsets Element::ownedElement} {opposite A_collaborationU[illegible]er}
  The Collabora[illegible]assifier.

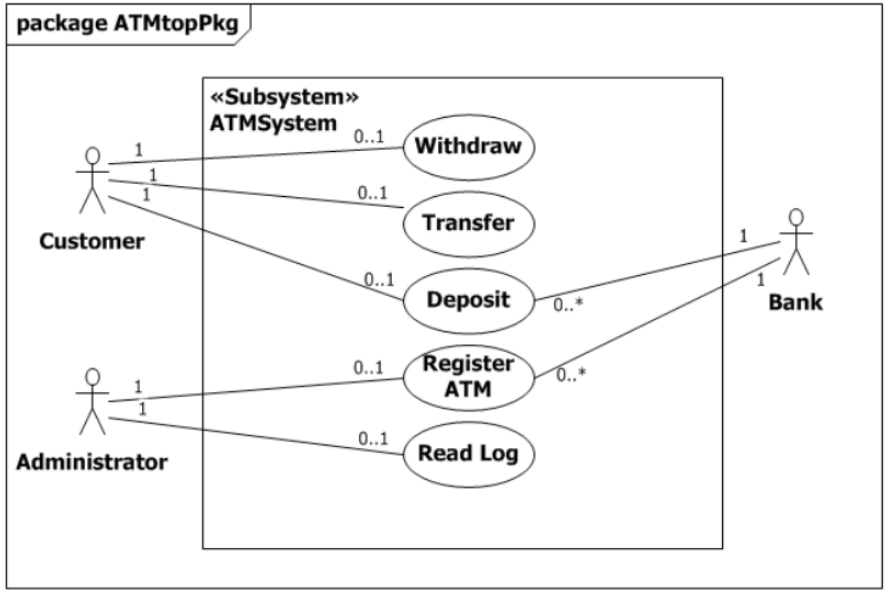
Details of the modelling elements („metamodell")

UML and the Development Methodologies

- The UML does not specify a development methodology

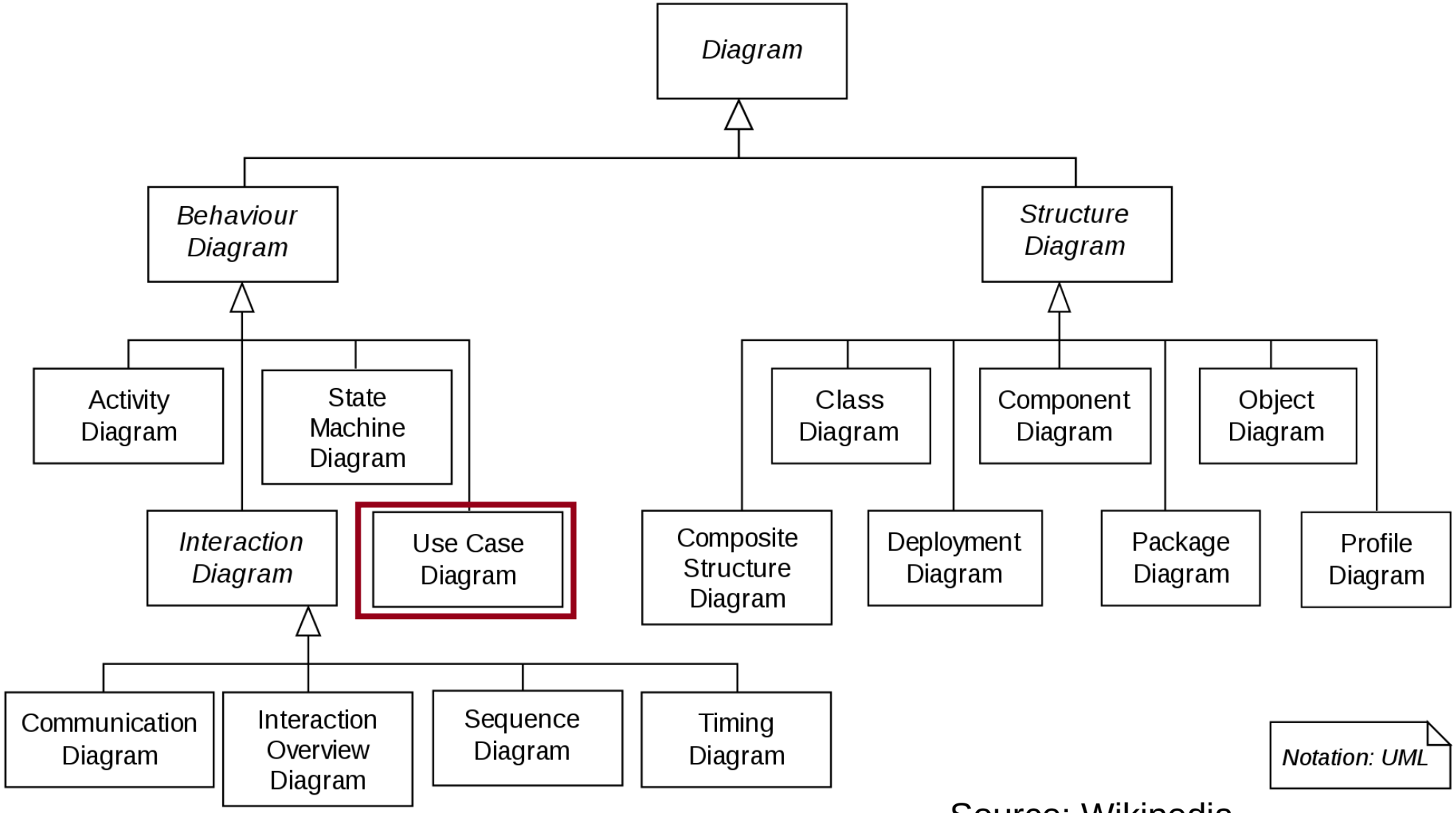
- It can be used in various environments and with different methods
  - agile,
  - Unified Process,
  - V-Model,
  - …

Use Cases



Use Case Diagrams

UML Diagram Types



Source: Wikipedia

Definition of Use Cases

- **Use case** captures a main functionality of the system corresponding to a functional requirement
- UCs describe
  - the typical interactions
  - between the *users* of a *system* and
  - **the system itself,**
  - by providing a narrative of how a system is used
- A set of scenarios tied together by a **common user goal**
- Language template: **Verb + Noun (Unique)!**
  - Example: Drive train, Switch turnout

Users ⊆ Stakeholders

M. Fowler: UML Distilled. 3rd Edition. Addison-Wesley

Use Case Descriptions

- Additional textual description to detail use cases
  - **Preconditions**: must hold for the use case to begin
  - **Postconditions**: must hold once the use case has been completed
  - **Primary flow**: the most frequent scenario(s) of the use case (aka. main success scenario, “happy path”)
  - **Alternate flow**: less frequent (or not successful)
  - **Exception flow**: not in support of the goals of the primary flow
- Elaborated behaviour (discussed later)
  - Activity diagrams: scenarios with complex control logic
  - Interaction diagrams: for message-based scenarios

Definition of Actors

- **Actor** is a **role** that a user plays with respect to the system.
  - *Primary actor*: invokes the system to deliver a service
  - *Secondary actor*: the system relies on them while carrying out the service
  - May be different in different use cases
- An actor is **outside the boundary** of the system
  - Actors ⊆ Context entities
- One person/entity may act as more than one actor
  - Example: A flight attendant can also be a passenger on another flight
- Can be an external subsystem (and not a person)

Use Case Form (Example)

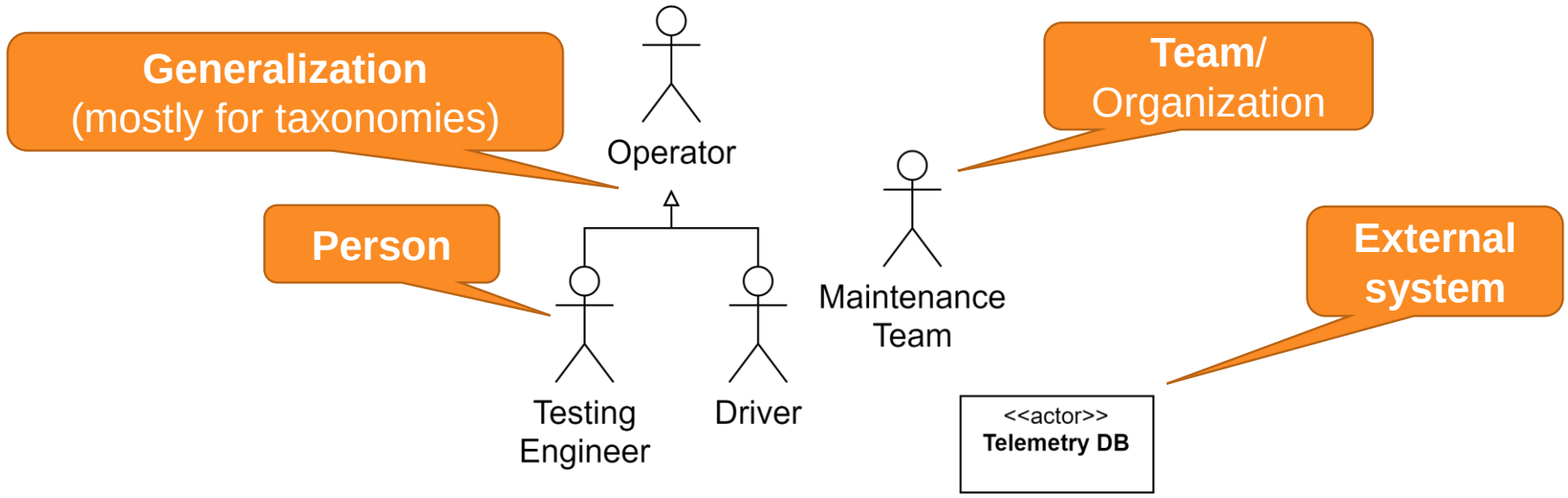
Use Case Sample Form

NAME:	*Unique and descriptive name of use case (typically verb + noun), optionally with ID*		
INVOLVED ACTORS:	*Who is invoking the functionality? Who is invoked to deliver the functionality?*	**INFORMATION ITEMS:**	*What physical/logical items are transferred during the interactions?*
INCLUDES:	*Which use cases are included?*	**EXTENDS:**	Which use cases are extended?
OBJECTIVE:	*What is the objective of the use case? What is the system supposed to achieve?*	**EXTENSION POINTS:**	Where can other use cases extend this one?
		REFINED REQUIREMENTS:	Which requirement(s) are applicable to the illustrated functionality?
PRECONDITIONS:	*What is necessary/expected to start the use case?*		
POSTCONDITIONS:	*What is promised after successfully executing the use case?*		
PRIMARY FLOW:	*The most frequent/typical scenario of the use case (aka. main success scenario or "happy path"). Example:* 1. *Actor 1 sends Item 1 to the system.* 2. *Included Use Case 1 is executed.* 3. *...*		
ALTERNATE FLOW:	*Less frequent or not successful (but expected) scenarios. Example: access as a guest instead of logging in; driving during the night.*		
EXCEPTIONAL FLOW:	*Scenarios not in support of the primary flow; typically error handling. May refer to extension points to allow other use cases to specify details (may be listed here as well).*		

Actors

Actor: specifies a role played by a user or any other system that interacts with the subject system.

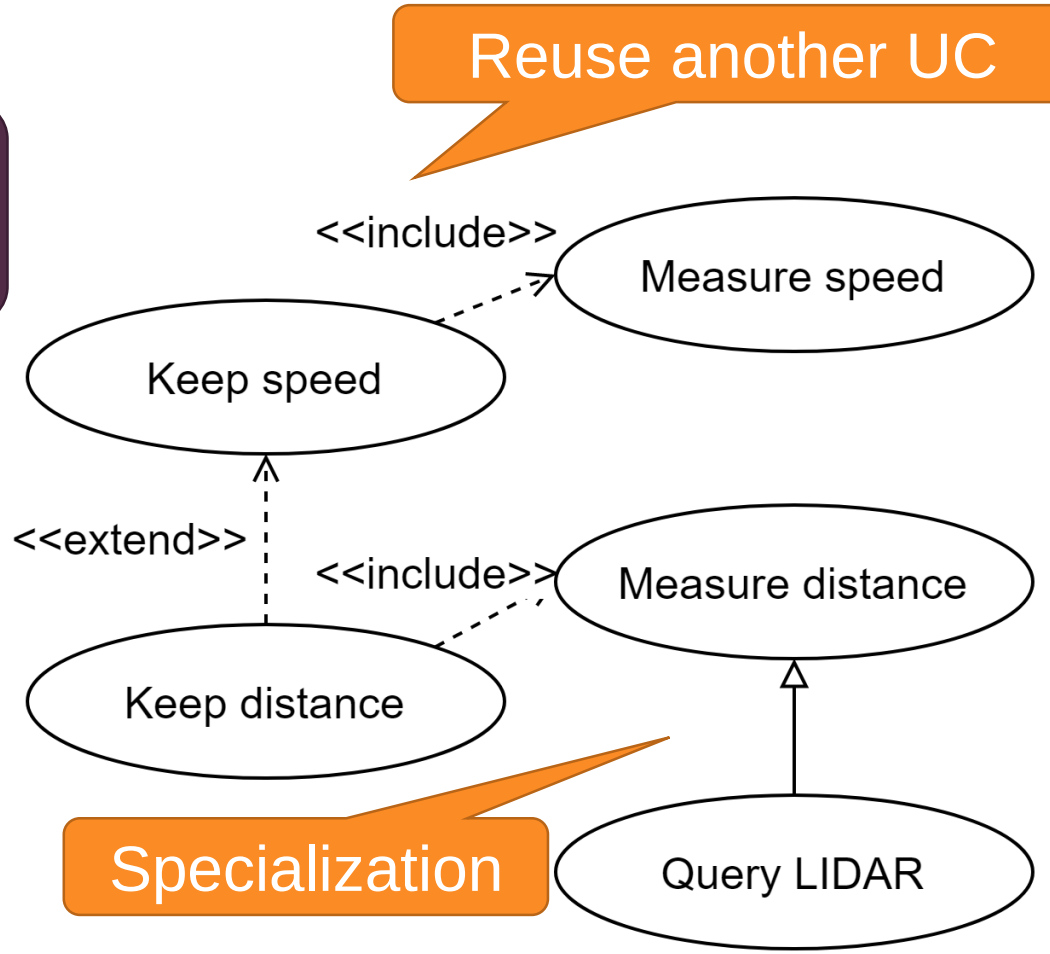
actor stereotype, or stick figure



Use Cases – Notation

Use cases: named oval shapes

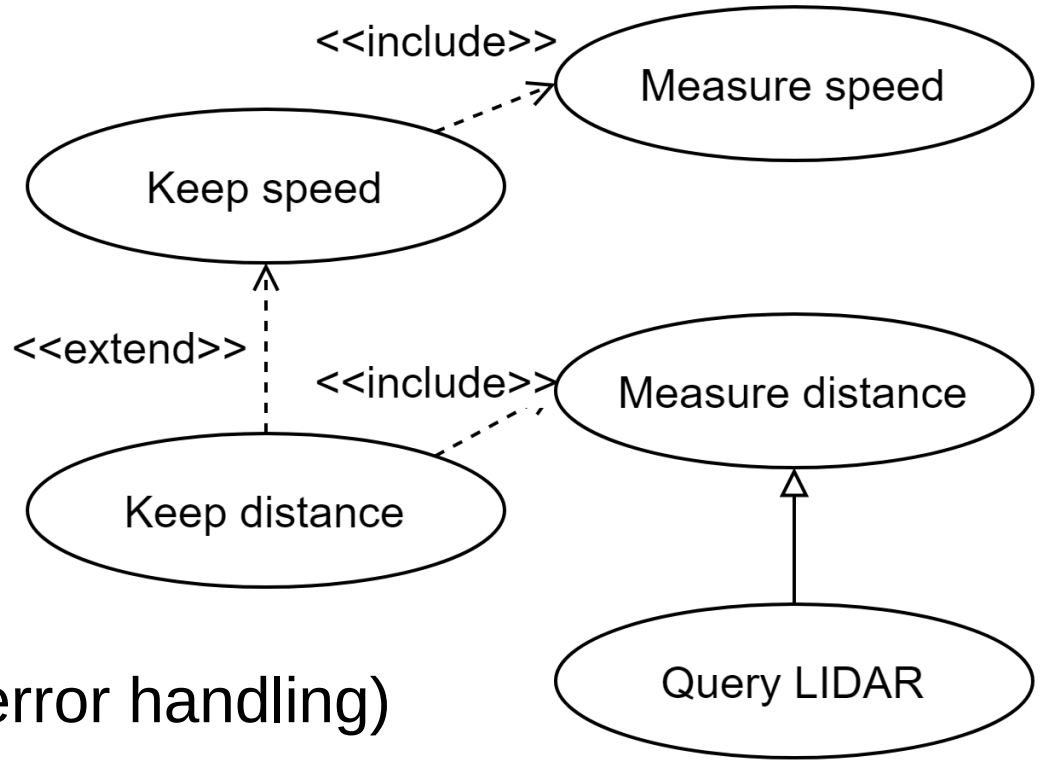
Relationships:
extension, inclusion, generalization



Use Cases – Relationships

- Generalization relationship
  - Abstract-concrete relationship
  - Expresses alternatives
- «*include*» relationship
  - Reusability
  - The included UC contributes to the primary functionality
- «*extend*» relationship
  - UC is not considered part of extended UC
  - Often models exceptional cases (e.g., error handling)

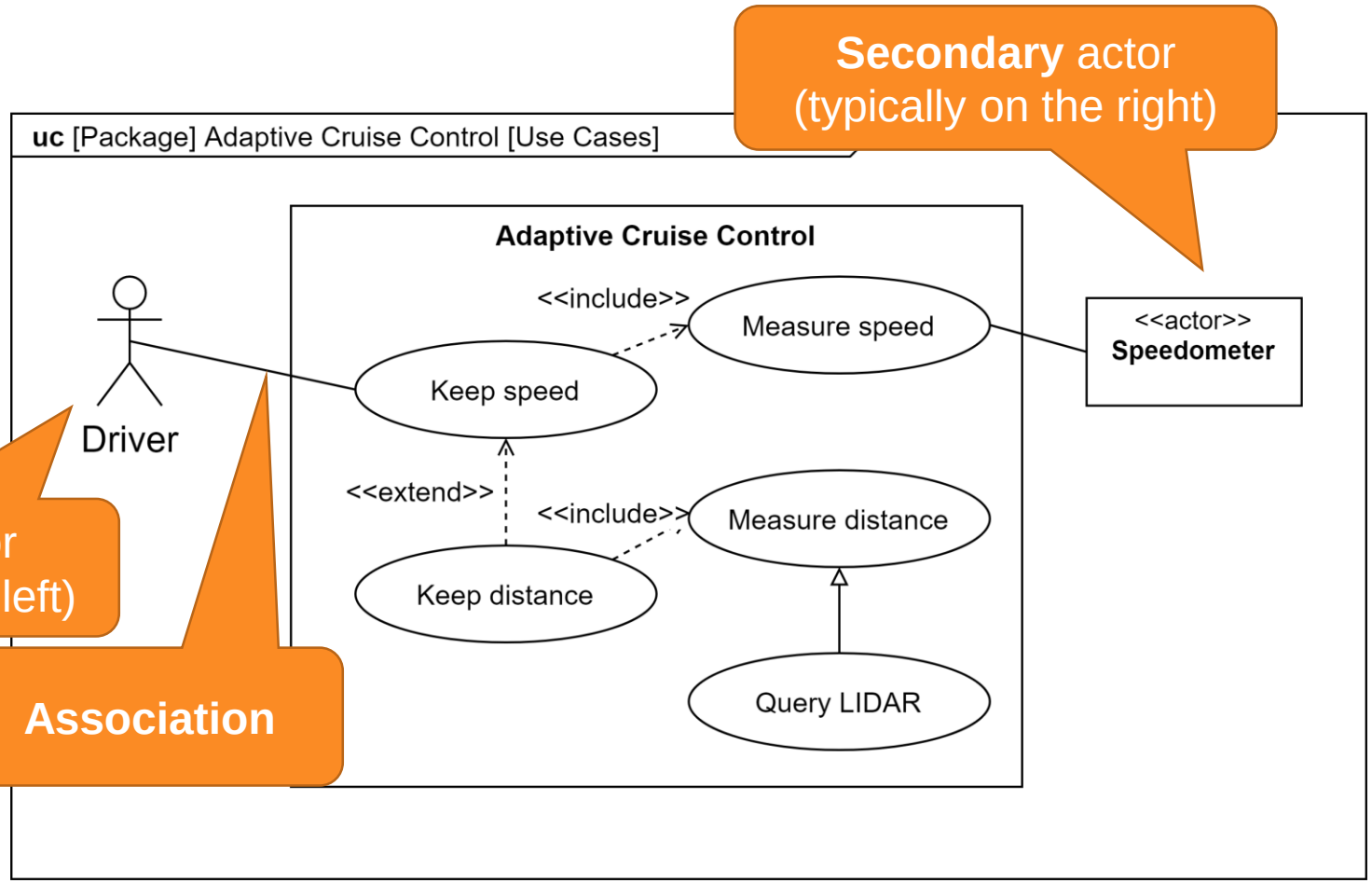
Note the arrow direction!
Which ends are client/supplier?



Use Case Diagram

Actor - use case relation:
communication path

Special **association**:
no properties, by default 0..1 multiplicity



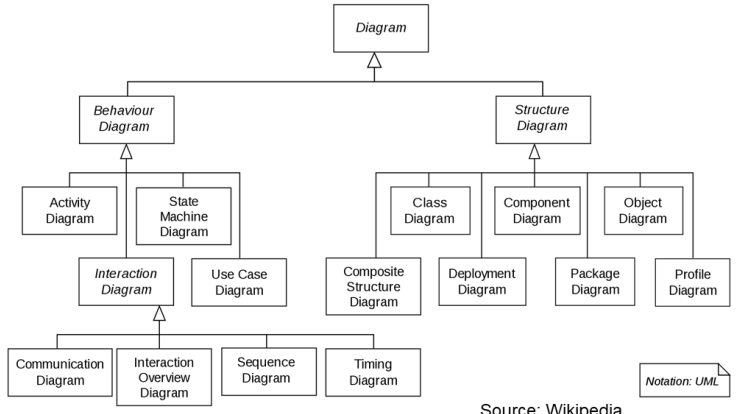
Summary

Summary

Unified Modeling Language (UML)

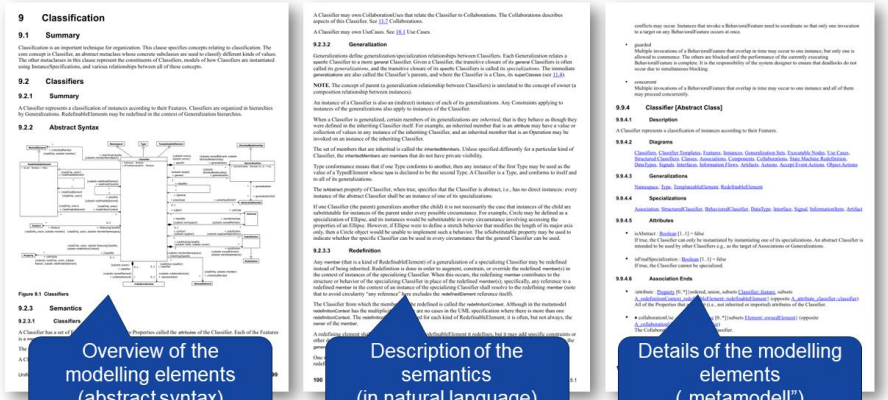
- Standardized language for describing software systems
  - Visual, general purpose, common language
- Maintained by the Object Management Group (OMG)
- History
  - The 90's: many different notations and methods
  - 1997: UML 1.0 proposal (G. Booch, I. Jacobsen, J. Rumbaugh and others)
  - 2005: UML 2.0, significantly renewed, common semantic basis
  - 2017: UML 2.5, simplified description, in one single document
  - 2011- defining precise semantics (fUML, PSCS, PSSM…)

UML Diagram Types



Source: Wikipedia

The UML Specification



Overview of the modelling elements (abstract syntax)

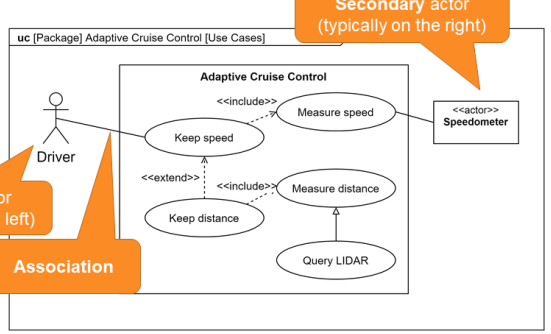
Description of the semantics (in natural language)

Details of the modelling elements („metamodell")

Use Case Diagram

Actor - use case relation: *communication path*

Special **association**: no properties, by default 0..1 multiplicity



Secondary actor (typically on the right)

Primary actor (typically on the left)

Association