Model Driven Engineering: Basic Concepts

Lesson 2
Modelling Languages

Guglielmo De Angelis
CNR - IASI / ISTI
guglielmo.deangelis@isti.cnr.it
modeling languages at a glance

- modeling languages are conceptual tools
- they support designers in
  - formalizing their thoughts
  - represent their reality
    - either textual or graphical

- two main classification of modeling languages:
  - general-purpose languages (GPLs) : designed on purpose for a specific domain/context/company to ease the task of people that need to describe things in that domain
  - domain-specific languages (DLSs) : represent modeling notations that can be applied to any sector of domain for modeling purposes
modeling languages at a glance

- are usually oriented to describe orthogonal aspects of a system
  - e.g. offering a set of different views
- typically each aspect can refer to one or more diagrams, which may
  - use different symbols/notations
  - target different representation of either the problem or the solution
- these aspects are often classified in
  - static (or structural) : elements and their relations
  - dynamic : actions of the modeled elements and their interactions
anatomy of modeling languages
anatomy of modeling languages

- abstract syntax: it describes the structure of the language and the way the different primitives can be combined together. It is independent from any representation/encoding.
anatomy of modeling languages

- **concrete syntax**: it describes a specific representation of the language: either textual and graphical. It is used by designers in modeling activities.
anatomy of modeling languages

- **semantics**: it describes the meaning of the elements in the language. It could be either formal or semi-formal. A partial or wrong specification of the semantic leads to misunderstandings and wrong usage of the language.
from the previous lesson … - 1 -

- as natural languages, programming languages are defined by means of grammars
- often formal grammars are expressed by means of another (formal) language
from the previous lesson … - 2 -

- in MDE models are specified according to “grammars” called metamodels
- metamodels are defined by means of other languages called: meta-metamodels
modeling VS drawing

• modeling
  – represents elements from according to a grammar
  – define relations among according to a grammar

• drawing
  – there is no reference to any grammar
  – there is no meaning associated with the modeling elements
  – only deals with represents graphical part of some modeling elements
    • shapes, lines, arrow
  – no control about the relations among the modeling elements
modeling VS drawing – example: graphic notation
modeling VS drawing –
example: textual notation
general-purpose modeling

UML: the Unified Modeling Language
what is UML?

“In short, the Unified Modeling Language (UML) provides industry standard mechanisms for visualizing, specifying, constructing, and documenting software systems.”
what is UML?

“In short, the Unified Modeling Language (UML) provides industry standard mechanisms for visualizing, specifying, constructing, and documenting software systems.”

• originally (mid. 90), it was mainly used for documenting software
  – melding several existing concepts/ideas
    • Booch : object-oriented
    • Harel : state machines
    • Jacobson : objectory vision
  – furthermore
    • it provides customizations mechanism (i.e. toward DSLs approaches)
    • it does not bind to any specific programming language
what is UML?

“In short, the Unified Modeling Language (UML) provides industry standard mechanisms for visualizing, specifying, constructing, and documenting software systems.”

• originally (mid. 90), it was mainly used for documenting software

• then, UML models have been adopted also for designing, developing, and testing software

• nevertheless, the actual contribution brought by the UML vision was improving AUTOMATION during the engineering of software products
the UML models - taxonomy

Diagram

Structure Diagram
- Class Diagram
- Component Diagram
- Object Diagram

Behavior Diagram
- Activity Diagram
- Use Case Diagram

Profile Diagram
- Composite Structure Diagram
- Deployment Diagram
- Package Diagram

Interaction Diagram
- Sequence Diagram
- Communication Diagram
- Interaction Overview Diagram
- Timing Diagram

Notation: UML
the UML models - scope
smells like UML diagrams …

• structure diagrams
  – class diagram
  – component diagram

• behavior diagrams
  – use case diagram
  – interaction diagrams
    • sequence diagram
class diagram

- it describe the structure of a systems:
  - nodes + relations
class diagram

• it describe the structure of a systems:
  - **nodes** + relations

• a node models a “class”, which represents:
  - an entity of the domain
  - elements which are not part of the domain, but they are useful for engineering the system

• a class is composed by
  - a name
  - a set of properties (i.e. attributes)
  - a set of operations

• a UML class matches with the notion of “class” in O.O
  - (simplifying) a type && a set of operation defining its interaction
  - modeling an abstraction of some entity of the domain
**Sensor**

- ID: Integer [1]
- # samplingFrequency: Integer [1] = 5000
- # isMute: Boolean [1]
- numberOfInstances: UnlimitedNatural [1]

**Operations**

+ getID(): Integer
+ setSamplingFrequency(in Integer)
+ getSamplingFrequency(): Integer
+ restoreSamplingFrequency()
+ setMuteOn()
+ setMuteOff()
+ getMeasure(): Integer
+ getInstances(): UnlimitedNatural

**Name**

Attributes
a UML class – 2

**CLASS**: Sensor

**PRIVATE**
- ID: Integer [1]
- #samplingFrequency: Integer [1] = 5000
- #isMute: Boolean [1]
- _numberOfInstances: UnlimitedNatural [1]

**PROTECTED**
- +getID(): Integer
- + setSamplingFrequency(in Integer)
- + getSamplingFrequency(): Integer
- + restoreSamplingFrequency()
- + setMuteOn()
- + setMuteOff()
- + getMeasure(): Integer
- + getInstances(): UnlimitedNatural
class diagram

- it describe the structure of a systems:
  - nodes + relations

- simplifying: they model the possible interactions among the classes within a model
  - a relation between a class A and a class B means that (in some way) A can interact with B
  - the type of the relation defines the way the classes interact
class diagram

- it describe the structure of a systems:
  - nodes + relations

- simplifying: they model the possible interactions among the classes within a model
  - a relation between a class $A$ and a class $B$ means that (in some way) $A$ can interact with $B$
  - the type of the relation defines the way the classes interact

association    dependency
aggregation    generalization
composition    interface Realization
it describe the structure of a system:

- nodes
- relations

simplifying: they model the possible interactions among the classes within a model.

A relation between a class A and a class B means that (in some way) A can interact with B.

The type of the relation defines the way the classes interact.

<table>
<thead>
<tr>
<th>Path Type</th>
<th>Notation</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregation</td>
<td></td>
<td>See “AggregationKind (from Kernel)” on page 38.</td>
</tr>
<tr>
<td>Association</td>
<td></td>
<td>See “Association (from Kernel)” on page 39.</td>
</tr>
<tr>
<td>Composition</td>
<td></td>
<td>See “AggregationKind (from Kernel)” on page 38.</td>
</tr>
<tr>
<td>Dependency</td>
<td></td>
<td>See “Dependency (from Dependencies)” on page 62.</td>
</tr>
<tr>
<td>Generalization</td>
<td></td>
<td>See “Generalization (from Kernel, PowerTypes)” on page 71.</td>
</tr>
<tr>
<td>InterfaceRealization</td>
<td></td>
<td>See “InterfaceRealization (from Interfaces)” on page 89.</td>
</tr>
</tbody>
</table>
association

- it models that instances of the source class can communicate with instances of the target class
  - if a class $A$ is not associated with a class $B$, then the instances of $A$ cannot call the operations exported by the instances of $B$
- it can model both symmetric, and circular communications
- it can be binary o N-ary
association – examples
association : classes & objects – 1
association: classes & objects – 2

Diagramma delle classi

rete di associazione

Esempio di diagramma degli oggetti
generalization

- is a relation between a more generic class and one of its specializations
  - generic class → super-class
  - specialized class → sub-class
- a sub-class inherits **ALL** the features defined by the super-class
  - is-a-kind-of
aggregation

- represents hierarchical links
  - A.K.A. “whole-part” relation
- instances of the “part”-side can be shared among several “whole”s
- circular aggregations are not semantically correct
  - A aggregates B
  - B aggregates C
  - C aggregates A
composition

- it models a **stronger** form of aggregation
  - instances of the "part"-side exclusively belong to one "whole"
  - if the "whole"-side is destroyed, all the binded "part"s **MUST** be also destroyed
smells like UML diagrams …

• structure diagrams
  – class diagram
  – component diagram

• behavior diagrams
  – use case diagram
  – interaction diagrams
    • sequence diagram
what is a software component?

• [C. Szyperski, Component Software – Beyond Object Oriented Programming, Addison Wesley, 1998]
  - “A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition.”

• [S. Crane, Component Interaction in Distributed Systems, ICCDS’98]
  - “A component is a unit of distributed program structure that encapsulates its implementation behind a strict interface comprised of services provided by the component to other components in the system and services required by the component and implemented elsewhere. The explicit declaration of a component's requirements increases reuse by decoupling components from their operating environment.”
**what is a software component?**

- [C. Szyperski, Component Software – Beyond Object Oriented Programming, Addison Wesley, 1998]
  
  “A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition.”

- [S. Crane, Component Interaction in Distributed Systems, ICCDS’98]
  
  “A component is a unit of distributed program structure that encapsulates its implementation behind a strict interface comprised of services provided by the component to other components in the system and services required by the component and implemented elsewhere. The explicit declaration of a component's requirements increases reuse by decoupling components from their operating environment.”
a software component
- incapsulates a self-contained unit of function
- it is often developed independently from its running context
- must declare the requested functionalities for its execution
- it manifests by means of models, source code, executables, libraries
- usually it is considered as **BLACK-BOX**
  - it is known **WHAT** the component does, but no **HOW**
## Table 8.1 - Graphic nodes included in structure diagrams

<table>
<thead>
<tr>
<th>NODE TYPE</th>
<th>NOTATION</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td><img src="image" alt="Component notation" /></td>
<td>See “Component”</td>
</tr>
<tr>
<td>Component implements Interface</td>
<td><img src="image" alt="Component implements Interface notation" /></td>
<td>See “Interface”</td>
</tr>
<tr>
<td>Component has provided Port (typed by Interface)</td>
<td><img src="image" alt="Component has provided Port notation" /></td>
<td>See “Port”</td>
</tr>
<tr>
<td>Component uses Interface</td>
<td><img src="image" alt="Component uses Interface notation" /></td>
<td>See “Interface”</td>
</tr>
</tbody>
</table>
### Table 8.1 - Graphic nodes included in structure diagrams

<table>
<thead>
<tr>
<th>NODE TYPE</th>
<th>NOTATION</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component has required Port (typed by Interface)</td>
<td><img src="image" alt="Diagram" /></td>
<td>See “Port”</td>
</tr>
<tr>
<td>Component has complex Port (typed by provided and required Interfaces)</td>
<td><img src="image" alt="Diagram" /></td>
<td>See “Port”</td>
</tr>
</tbody>
</table>

### Table 8.2 - Graphic nodes included in structure diagrams

<table>
<thead>
<tr>
<th>PATH TYPE</th>
<th>NOTATION</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly connector</td>
<td><img src="image" alt="Diagram" /></td>
<td>See “assembly connector.” Also used as notation option for wiring between interfaces using Dependencies.</td>
</tr>
</tbody>
</table>
component diagram – example
smells like UML diagrams …

- structure diagrams
  - class diagram
  - component diagram
- behavior diagrams
  - use case diagram
  - interaction diagrams
    - sequence diagram
sequence diagram

- models the messages exchanged among class instances
  - each diagram models one admissible interaction among more objects
  - sometime they are improperly used in order to model the interactions among classes

- messages reflect a (partial) temporal order

- useful for:
  - modeling use-case
  - modeling behaviors observable from the interactions of objects

- they includes primitives modeling how the execution flows over time (i.e. fragments)
smells like UML diagrams …

- structure diagrams
  - class diagram
  - component diagram
- behavior diagrams
  - use case diagram
  - interaction diagrams
    - sequence diagram
use case diagram

• ingredients:
  • actors: roles or user of some functionality (often it is a human being)
  • system: the subject to be modeled
  • use case: a functional unit, a way the actors can interact with the system
  • relations between actors and the use cases: define the actors participating to a use case
  • relations among the use cases: define relations such as inclusions, extensions, and generalizations
use case: logical structure
Table 16.1 - Graphic nodes included in use case diagrams

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Notation</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseCase</td>
<td>Withdraw</td>
<td>See “UseCase (from UseCases)” on page 594.</td>
</tr>
<tr>
<td></td>
<td>On-Line Help</td>
<td></td>
</tr>
<tr>
<td></td>
<td>extension points</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Perform ATM Transaction</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OrderStationery</td>
<td></td>
</tr>
</tbody>
</table>
extension : logical structure

flows described by extensions from external use cases
inclusion : logical structure

portion of the main flow included from an external use case
### Table 16.1 - Graphic nodes included in use case diagrams

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Notation</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Extend</strong></td>
<td><img src="image1" alt="Diagram" /></td>
<td>See “Extend (from UseCases)” on page 589.</td>
</tr>
<tr>
<td><strong>Extend (with Condition)</strong></td>
<td><img src="image2" alt="Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>

### Table 16.1 - Graphic nodes included in use case diagrams

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Notation</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Include</strong></td>
<td><img src="image3" alt="Diagram" /></td>
<td>See “Include (from UseCases)” on page 592.</td>
</tr>
</tbody>
</table>
use case diagram – example 1

- **Customer**:
  - Withdraw (1..1)
  - Transfer Funds (0..1)
- **Administrator**:
  - Deposit Money (0..1)
  - Register ATM at Bank (0..1)
  - Read Log (0..1)
- **ATM System**
- **Bank**:
  - (1..1)