Embedded Systems Software Engineering

Jarosław Kuchta

Unified Modeling Language (Foundation)
Modeling vs. Development

Modeling = Analysis & Design
Development = Design & Implementation
Implementation = Coding & Debugging

• UML was provided for modeling
• UML is also used for design
• UML does not have enough detail level for implementation
Models vs. Diagrams

Model – abstract representation of a system
  Abstraction – refinement stop problem
Diagram – graphical representation of a model

- UML uses diagrams to model a software system
- There are other model representation known (e.g. CRC – Class-Responsibility-Collaboration Cards)
UML Diagrams

• Static structure diagrams
  – class diagram
  – component diagram
  – deployment diagram

• Functionality diagrams
  – use case diagram
  – interaction diagram
  – sequence diagram

• Behavioral diagrams
  – state transition diagram
  – activity diagram

• Missing in UML:
  – timing diagrams
  – entity-relationship diagrams
  – navigation diagrams
  – ...
Class diagram
(misunderstandings)

• Developers treat classes as software structures (data + functions)
• Modelers treat classes as abstraction of real world entities

One model class → many software classes

• Modeling classes: no need for many classes, no need for many class attributes, no need for class functions
### Other (subtle) misunderstandings (1)

<table>
<thead>
<tr>
<th><strong>Entity</strong> – something that exists in the real world, has a large set of features</th>
<th><strong>Object</strong> – some entity that exists in the model, has limited set of features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Class</strong> – a potential object or set of objects (also empty if class is abstract)</td>
<td><strong>Object</strong> – a concrete instantiation of a class</td>
</tr>
<tr>
<td><strong>Property</strong> – an informative feature of a class</td>
<td><strong>Attribute</strong> – an information held by object</td>
</tr>
<tr>
<td><strong>Relationship</strong> – a logical binding between two or more classes</td>
<td><strong>Link</strong> – a concrete binding between two or more objects; an instantiation of a relationship</td>
</tr>
<tr>
<td><strong>Generalization-specialization</strong> – an ontology relationship between two classes (“is a” relationship)</td>
<td><strong>Inheritance</strong> – a mechanism of passing features from a generalized to a specialized class</td>
</tr>
<tr>
<td><strong>Multiple inheritance</strong> – when a class has multiple generalizations (often modeled)</td>
<td><strong>Single inheritance</strong> – when a class has only one (direct) generalization (often required by implementation language)</td>
</tr>
</tbody>
</table>
### Other (subtle) misunderstandings (2)

<table>
<thead>
<tr>
<th><strong>Aggregation (weak aggregation)</strong> – a relationship between classes that allows joining two or more objects (components) together; a new object (an assembly) is created</th>
<th><strong>Containment (strong aggregation)</strong> – a relationship between classes that allows including one or more objects in other object (a container)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Association</strong> – any other relationship between two or more classes; associations between more than two classes are hard to implement</td>
<td><strong>Pointer, reference</strong> – a specific implementation of an association; only two objects are unidirectionally linked, (bidirectional association requires two pointers / references)</td>
</tr>
<tr>
<td><strong>Association role</strong> – a name of an object when it is linked to other object</td>
<td><strong>Class name</strong> – can be used as a role substitute but only when classes at opposite sides of the association are different</td>
</tr>
<tr>
<td><strong>Association direction</strong> – a direction in which an association should be read</td>
<td><strong>Association navigation</strong> – determines the class where a pointer/reference should be implemented</td>
</tr>
</tbody>
</table>
### Other (subtle) misunderstandings (3)

<table>
<thead>
<tr>
<th><strong>Collection, list</strong> – a container of objects</th>
<th><strong>Array</strong> – a specific implementation of a collection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multiplicity</strong> – a potential count of contained or aggregated objects or associated objects playing same role</td>
<td><strong>Array size</strong> – a concrete count of objects in an array</td>
</tr>
<tr>
<td><strong>Operation</strong> – an abstract action that can be performed on an object of the class</td>
<td><strong>Function</strong> – a concrete implementation of an operation; needs parameters specification; can be overloaded and overridden</td>
</tr>
<tr>
<td><strong>Class name</strong> – a name of a class</td>
<td><strong>Object name</strong> – can be different from or the same as the class name</td>
</tr>
<tr>
<td><strong>Class name</strong> – a name of a class when it is used in a model; it can consist of two or more words and national letters</td>
<td><strong>Class identifier</strong> – a name of a class when it is implemented in a program; only one word is allowed, national letters in some languages</td>
</tr>
</tbody>
</table>
Other (subtle) misunderstandings (4)

<table>
<thead>
<tr>
<th>ID property</th>
<th>– not needed in analysis</th>
<th>– needed in database design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visibility (private, protected, public)</td>
<td>– not needed in analysis; all features are public</td>
<td>- can be used in design</td>
</tr>
</tbody>
</table>
Class diagram elements

class definition frame

- class name
- attributes
- operations

header

compartments

generalization-specialization

- weak aggregation
- strong aggregation

association (binary)

0..1 1..*

- private attribute
- protected attribute
+ public attribute
Derived attribute

• /age = current year – year(birth date)
• +/age
N-ary attributes

- name [multiplicity]
  - n – exactly n values
  - 0..1 – no values or one value
  - 0..n – no values and no more than n values
  - n..m – no less than n and no more than m values
  - 0..* – no values or possibly many values
  - n..* – no less than n values
  - * – possibly many values
Attribute types

• General types – used in analysis
  – number
  – integer
  – real

• Specific types – used in design
  – decimal
  – longint
  – double

• Defined types (enums)
Example

Controller

+Language: String
+Code: String

Device controller
+Device Name: String

Process controller
+Process ID: Number
Operation stereotypes

- **<<constructor>>** – creates a new instance
- **<<destructor>>** – destroys an instance
- **<<query>>** – gets a state of an object (state remains unchanged)
- **<<update>>** – changes a state of an object

A state of an object – one or more object attributes
Aggregation

Measurement

+Number: String
+Date: Date Time
+Person ID: Number

Measurement result

+Controller: Controller
+Value: Real
N-ary association

<table>
<thead>
<tr>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Number: String</td>
</tr>
<tr>
<td>+Date: Date Time</td>
</tr>
<tr>
<td>+Person ID: Number</td>
</tr>
</tbody>
</table>

| Employee           |

| Supervisor         |

Association roles

Employee

subordinate  superordinate

1  0..*

Employee
Association class

Employee

Department

Delegation

+Date
Class model purpose

- Static analysis of a problem domain
- Basis for functional and behavioral analysis
- Application logic design
- Helper for implementation

Note: various levels of abstraction-refinement
Bibliography