PDOM – Problem Domain Object Model

ARNON LAZERSON
Problem Domain Object Model

- A **domain model** is a representation of real-world concepts relevant to the domain that need to be modeled in software.

- **Problem domain** refers to all information that defines the problem and constrains the solution.

- To create the PDOM we must identify the key entities in the system and their relationships.

- Focus on the problem and not on the solution
PDOM is used for

- Defining and clarifying the main concepts and relationships
- Discovering ambiguities and contradictions in the user specifications
- A common language
- A base for class diagram
UML Class Diagram

- A Class Diagram is a diagram describing the structure of a system

- Includes
  - Classes
  - Attributes
  - operations (or methods)
  - Relationships among the classes
  - Constraint Rules and Notes
Class Relationships

- Association
- Generalization
- Realization
- Dependency
Association

- An association between two classes indicates that objects at one end of an association “recognize” objects at the other end and may send messages to them.

- Example: “An Employee works for a Company”
Association (cont.)

StaffMember

instructor

1..*

Role name

Association name

Navigable (uni-directional) association

Reflexive association

Student

instructs

*

Courses

pre - requisites

0..3

Role

Courses

0..3

PDOM - PROBLEM DOMAIN OBJECT MODEL 7
Association (cont.)

- To clarify its meaning, an association may be named.
  - The name is represented as a label placed midway along the association line.
  - Usually a verb or a verb phrase.

- A **role** is an end of an association where it connects to a class.
  - May be named to indicate the role played by the class attached to the end of the association path.
    - Usually a noun or noun phrase
    - Mandatory for reflexive associations
Association (cont.)

Multiplicity

- The number of objects that participate in the association.
- Indicates whether or not an association is mandatory.

<table>
<thead>
<tr>
<th>Multiplicity Indicators</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exactly one</td>
<td>1</td>
</tr>
<tr>
<td>Zero or more (unlimited)</td>
<td>* (0..*)</td>
</tr>
<tr>
<td>One or more</td>
<td>1..*</td>
</tr>
<tr>
<td>Zero or one (optional association)</td>
<td>0..1</td>
</tr>
<tr>
<td>Specified range</td>
<td>2..4</td>
</tr>
<tr>
<td>Multiple, disjoint ranges</td>
<td>2, 4..6, 8</td>
</tr>
</tbody>
</table>
Aggregation

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.
  - Models a “is a part-part of” relationship.

![Diagram of Aggregation]

- Car
- Door
- House

Whole
Part

PDOM - PROBLEM DOMAIN OBJECT MODEL
Aggregation (cont.)

Aggregation tests:

- Is the phrase “part of” used to describe the relationship?
  - A door is “part of” a car

- Are some operations on the whole automatically applied to its parts?
  - Move the car, move the door.

- Are some attribute values propagated from the whole to all or some of its parts?
  - The car is blue, therefore the door is blue.

- Is there an intrinsic asymmetry to the relationship where one class is subordinate to the other?
  - A door *is* part of a car. A car *is not* part of a door.
Composition

A strong form of aggregation

- The whole is the sole owner of its part.
  - The part object may belong to only one whole
- Multiplicity on the whole side must be zero or one.
- The life time of the part is dependent upon the whole.
  - The composite must manage the creation and destruction of its parts.

Circle

Point

Polygon

PDOM - PROBLEM DOMAIN OBJECT MODEL
Generalization

- Indicates that objects of the specialized class (subclass) are substitutable for objects of the generalized class (super-class).
  - “is kind of” relationship.

```abstract
{abstract} is a tagged value that indicates that the class is abstract. The name of an abstract class should be italicized
```
Generalization

- A sub-class inherits from its super-class
  - Attributes
  - Operations
  - Relationships

- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations

- A generalization relationship **may not** be used to model interface implementation.
Realization

- A realization relationship indicates that one class implements a behavior specified by another class (an interface or protocol).
- An interface can be realized by many classes.
- A class may realize many interfaces.
Dependency

- Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time.

- One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class.

- This is different from an association, where an attribute of the dependent class is an instance of the independent class.
Constraint Rules and Notes

- **Constraints** and **notes** annotate among other things associations, attributes, operations and classes.

- Constraints are semantic restrictions noted as Boolean expressions.
  - UML offers many pre-defined constraints.

```
<table>
<thead>
<tr>
<th>Customer</th>
<th>1</th>
<th>Order</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>id: long { value &gt; 0 }</td>
<td>{ total &lt; $50 }</td>
<td>may be canceled</td>
<td></td>
</tr>
</tbody>
</table>
```

Constraint: id: long { value > 0 }

Note: may be canceled
Example

Traffic violation report system

- A traffic policeman can issue traffic reports to offenders
- Each report can consist of many violations (at least one)
- Each offender may have multiple reports
Traffic violation report system

- A traffic policeman can issue traffic reports to offenders
- Each report can consist of many violations (at least one)
- Each offender may have multiple reports
ב¶ ש¶ג¶י¶ ל¶א¶ר¶ק¶, ב¶ו¶י¶ מ¶לו¶ו¶ נ¶ר¶ש¶ נ¶ר¶ש¶ ומ¶ק¶ב¶ ז¶מ¶יי¶... 
ל¶א¶ח¶ר¶ ה¶ר¶ש¶מה¶ מ¶צ¶ל¶ח¶ת¶ ي¶פ¶ח¶ת¶ ל¶י¶ל¶ד¶ כ¶ר¶ט¶יס¶ א¶ל¶ק¶ש¶ר¶ו¶ן¶... 
ה¶מ¶ל¶ו¶ו¶ ר¶כ¶ש¶ ע¶ב¶ו¶ר¶ ה¶י¶ל¶ד¶ כ¶נ¶יס¶ו¶ת¶ ל¶מ¶ת¶ק¶נ¶י¶ מ¶ש¶ו¶ני¶... 
ע¶נ¶ ה¶ר¶ש¶מ¶ה¶ נ¶פ¶ח¶ת¶ ل¶מ¶ל¶ו¶ו¶ ג¶מ¶ ח¶ש¶ב¶ו¶ ר¶כ¶י¶ש¶ח¶... 
ב¶מ¶ר¶כ¶ ה¶ש¶ל¶י¶ט¶ו¶ ה¶ב¶כ¶ר¶ה¶ ש¶ל¶ ה¶פ¶א¶ר¶ק¶ נ¶מ¶צ¶א¶ מ¶פ¶ק¶ה¶ מ¶ז¶ש¶ל¶ו¶ על¶ ה¶מ¶ת¶ק¶נ¶י¶...