Structural specification

Alexander Serebrenik
Before we start

- Time-related signal in an activity diagram is represented as:

  a)  
  b)  
  c)  

\[ R \]
Before we start

• Time-related signal in an activity diagram is represented as

  a) ![Diagram 1]
  b) ![Diagram 2]
  c) ![Diagram 3]
Before we start

• Time-related signal in an activity diagram is represented as

\[
\begin{align*}
&a) \quad \text{Diagram a} \\
&b) \quad \text{Diagram b} \\
&c) \quad \text{Diagram c}
\end{align*}
\]

• Dr. Smith is grading students’ exam papers. How would you model her grading process?
Before we start

- Time-related signal in an activity diagram is represented as:
  - a) [Diagram of a rectangle with a curve]
  - b) [Diagram of a rectangle with an arrow]
  - c) [Diagram of a rectangle with a cross]

- Dr. Smith is grading students’ exam papers. How would you model her grading process?

  - a) [Diagram with 'Grade paper']
  - b) [Diagram with 'Grade paper']
  - c) [Diagram with 'Grade paper']
  - d) [Diagram with 'Grade paper']

  **interruptible activity region**
  **CORRECT: expansion region**
  **subactivity (name missing)**
  **does not exist**
Before we start

- Seminar-registration state machine. What happens when a student is enrolled?

This week sources

Slides of Mohammad Mousavi
What is structure?

- **Structure**: the aggregate of elements of an entity in their relationships to each other. *Merriam-Webster*

- **Structural model**
What is structure?

- **Structure**: the aggregate of elements of an entity in their relationships to each other. *Merriam-Webster*

- **Structural modelling**: discover the key data contained in the problem domain and to build a structural model of the objects.
A diagram that identifies **modules, activities, or other entities** in a system or computer program and **shows how larger or more general entities break down into smaller, more specific entities**.

*IEEE Standard Glossary of Software Engineering Terminology 610.12 1990*
UML structure diagrams

- Class diagram
- Object diagram
- Component diagram
- Packages diagram
- Deployment diagram
- Composite structure diagram
If someone were to come up to you in a dark alley and say,

“Psst, wanna see a UML diagram?”

that diagram would probably be a class diagram.

[Martin Fowler, UML Distilled, Chapter 3]
Class diagrams

- most common and most useful
- describe the types of objects in the System and the various kinds of static relationships that exist among them
- show the properties and operations of a Class and the constraints that apply to the way objects are connected

[Martin Fowler, UML Distilled, 2003]

- kind of extended ER diagrams
  - NB: differences in notation
Objects and classes

• describe the **types of objects** in the System and the various kinds of static relationships that exist among them

• Do you remember?
  • **objects**: units of encapsulations of data (state) and functionality with an identity
  • **classes**: collections of objects [types of the objects]
Properties, operations and constraints

• show the **properties** and **operations** of a Class and the **constraints** that apply to the way objects are connected

• Do you remember?
  • **properties** (**attributes** and associations): units of data (responsibilities for knowing)
  • **operations**: units of functionality (responsibilities for doing)
  • **constraints**: conditions on attributes, pre- and post-conditions of operations
Example

- e-Commerce site (Amazon, bol.com, ...)
- Class: Order

Give examples of properties
Attributes and associations

- Two different ways **properties** can be presented
- **Attribute** – property as a line of text within the class box.
- **Association** – property as a solid line between two classes, directed from the source class to the target class. Name is written on the line close to the target class.

<table>
<thead>
<tr>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>number: String</td>
</tr>
</tbody>
</table>

Order

name

String
Navigability

We can query Car about its owner but not Person about her cars.

We can query Car about its owner, and Person about her cars. Moreover,
x in cars(owner(x)), y in owner(cars(y))

No navigability information.
### Order

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Optionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>String [1]</td>
<td></td>
</tr>
<tr>
<td>placedOn</td>
<td>Date [1]</td>
<td></td>
</tr>
<tr>
<td>deliveredOn</td>
<td>Date [0..1]</td>
<td></td>
</tr>
<tr>
<td>totalAmount</td>
<td>Money [1]</td>
<td></td>
</tr>
<tr>
<td>recipient</td>
<td>Name [1]</td>
<td></td>
</tr>
<tr>
<td>lineItems</td>
<td>OrderLine [1..*] {ordered}</td>
<td></td>
</tr>
</tbody>
</table>
How would we write this in UML?

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>String</td>
<td>[1]</td>
</tr>
<tr>
<td>placedOn</td>
<td>Date</td>
<td>[1]</td>
</tr>
<tr>
<td>deliveredOn</td>
<td>Date</td>
<td>[0..1]</td>
</tr>
<tr>
<td>totalAmount</td>
<td>Money</td>
<td>[1]</td>
</tr>
<tr>
<td>recipient</td>
<td>Name</td>
<td>[1]</td>
</tr>
<tr>
<td>lineItems</td>
<td>OrderLine</td>
<td>[1..*] {ordered}</td>
</tr>
</tbody>
</table>

**Multiplicity**

- how many objects may fill the attribute: m..n
- at least m
- at most n
- * denotes ∞
How would we write this in UML?

Order

+number: String [1]
+placedOn: Date [1]
+deliveredOn: Date [0..1]
+totalAmount: Money [1]
+receipt: Name [1]
+lineItems: OrderLine [1..*] {ordered}

Multiplicity
how many objects may fill the attribute: m..n
at least m
at most n
* denotes ∞

Additional
Any kind of extra information needed about the attribute
How would we write this in UML?

<table>
<thead>
<tr>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>+number: String [1]</td>
</tr>
<tr>
<td>+placedOn: Date [1]</td>
</tr>
<tr>
<td>+deliveredOn: Date [0..1]</td>
</tr>
<tr>
<td>+totalAmount: Money [1]</td>
</tr>
<tr>
<td>+recipient: Name [1]</td>
</tr>
<tr>
<td>+lineItems: OrderLine [1..*] {ordered}</td>
</tr>
</tbody>
</table>

**Multiplicity**
- how many objects may fill the attribute: m..n
  - at least m
  - at most n
- * denotes ∞

**Visibility**
- Which other classes can access this attribute?
  - + (public) every other class
  - - (private) no other class
  - # (protected) only classes that inherit from Order
  - ~ (package)

**Additional**
- Any kind of extra information needed about the attribute
Beware: visibility in Java ≠ visibility in UML

- Different programming languages use the same names for visibility levels but subtly different meanings
Beware: visibility in Java ≠ visibility in UML

• Different programming languages use the **same names** for visibility levels but subtly **different meanings**

• **Translation** from UML to a programming language depends on the programming language

<table>
<thead>
<tr>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>+number: String [1]</td>
</tr>
</tbody>
</table>

Java

```java
public class Order {
    private String number;
    public String getNumber() {
        return number;
    }
    public void setNumber(String number) {
        this.number = number;
    }
}
```

C#

```csharp
public class Order {
    private String number;
    public String GetNumber() {
        return number;
    }
    public void SetNumber(String number) {
        this.number = number;
    }
}
```
Recall our example

- e-Commerce site (Amazon, bol.com, …)
- Class: Order

Give examples of operations

ORDER PLACED
2 Sep 2013
Order Details  |  Printable Order Summary

ORDER #026-4747744-1109156
RECIPIENT Alexander Serebrenik
TOTAL £210.97

Delivered On: Saturday 7 Sep 2013

Delivered

| Qualitative Organizational Research: Core Methods and Current Challenges |
| Symon, Gillian |
| Sold by Amazon EU S.a.r.L. |
| Experimentation in Software Engineering |
| Wohlin, Claes |
| Sold by Amazon EU S.a.r.L. |
| Guide to Advanced Empirical Software Engineering |
| Shull, Forrest |
| Sold by Amazon EU S.a.r.L. |
| Making Software: What Really Works, and Why We Believe It |
| Andy Oram |
| Sold by Amazon EU S.a.r.L. |
How would we write this in UML?

<table>
<thead>
<tr>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>+number: String [1]</td>
</tr>
<tr>
<td>+placedOn: Date [1]</td>
</tr>
<tr>
<td>+deliveredOn: Date [0..1]</td>
</tr>
<tr>
<td>+totalAmount: Money [1]</td>
</tr>
<tr>
<td>+recipient: Name [1]</td>
</tr>
<tr>
<td>+lineItems: OrderLine [1..*] {ordered}</td>
</tr>
<tr>
<td>+dispatch()</td>
</tr>
<tr>
<td>+close()</td>
</tr>
</tbody>
</table>

Visibility

Which other classes can access this operation?
The same idea as with attributes
Attribute or association?

- **Attribute**
  - **Value** is important, not **identity**: Date, Number, String
  - Little behavior and functionality

- **Association** (i.e., a separate class)
  - **Identity** is important: Customer, Order, Student, Book
  - Complex behavior or functionality
### Attribute or association?

**Attribute**
- **Value** is important, not **identity**: Date, Number, String
- Little **behavior** and **functionality**

**Association** (i.e., a separate class)
- **Identity** is important: Customer, Order, Student, Book
- Complex **behavior** or **functionality**

```
<table>
<thead>
<tr>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>+number: String [1]</td>
</tr>
<tr>
<td>+placedOn: Date [1]</td>
</tr>
<tr>
<td>+deliveredOn: Date [0..1]</td>
</tr>
<tr>
<td>+totalAmount: Money [1]</td>
</tr>
<tr>
<td>+recipient: Name [1]</td>
</tr>
<tr>
<td>+dispatch()</td>
</tr>
<tr>
<td>+close()</td>
</tr>
</tbody>
</table>
```

```
OrderLine 1..* +lineltems {ordered}
```

---

This slide explains the difference between attributes and associations in object-oriented modeling. Attributes are simple properties like Date, Number, or String, which hold values. Associations, on the other hand, are more complex and can involve separate classes, such as Customer, Order, Student, and Book. The behavior and functionality associated with these entities can be quite complex as well.
Attribute or association?

- **Attribute**
  - **Value** is important, not **identity**: Date, Number, String
  - Little **behavior** and **functionality**

- **Association** (i.e., a separate class)
  - **Identity** is important: Customer, Order, Student, Book
  - Complex **behavior** or **functionality**

---

**Do you agree?**

```
OrderLine 1..* OrderLine

Order
+number: String [1]
+placedOn: Date [1]
+deliveredOn: Date [0..1]
+totalAmount: Money [1]
+recipient: Name [1]
+dispatch()
+close()
```

---
Special kind of association

- Order lines do not really have identity
- Two orders cannot share an order line
- When an order is deleted, order lines do not “survive”...

Purchase Order

Electric Controls Company
12582 Camino Del Rio
San Diego, CA 92110-4264

To: US Electrical Controls
14878 Freemont Avenue
Suite 1800
St. Louis, MO 63127-5588

P.O. Number 700001
Please include this number on all invoices and shipping documents.

P.O. Date January 14, 2005
Vendor Number 1007
Expected Ship Date January 29, 2005

<table>
<thead>
<tr>
<th>Your Item Number</th>
<th>Our Item Number</th>
<th>Description</th>
<th>Quantity</th>
<th>Price</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>240-100-SW264</td>
<td>102</td>
<td>Switch, DPST 24V 180a</td>
<td>50</td>
<td>$14.98</td>
<td>$744.00</td>
</tr>
<tr>
<td>240-100-SW164</td>
<td>105</td>
<td>Switch, SPST 24V 100a</td>
<td>100</td>
<td>$0.47</td>
<td>$947.00</td>
</tr>
<tr>
<td>240-50-SW236</td>
<td>112</td>
<td>Switch, DPST 24V 50a</td>
<td>80</td>
<td>$5.66</td>
<td>$692.80</td>
</tr>
<tr>
<td>120-40-CB79</td>
<td>115</td>
<td>Circuit breaker, 120V 40a</td>
<td>100</td>
<td>$6.95</td>
<td>$695.00</td>
</tr>
</tbody>
</table>

Purchase Order Total $3,082.80

http://blog.procurify.com/wp-content/uploads/2013/06/Purchase-Order.gif
Special kind of association

- Order lines do not really have identity
- Two orders cannot share an order line
- When an order is deleted, order lines do not “survive”…

**Composition**

**Order**

+number: String [1]
+placedOn: Date [1]
+deliveredOn: Date [0..1]
+totalAmount: Money [1]
+recipient: Name [1]
+dispatch()
+close()

**OrderLine**

1..*
+lineItems {ordered}
Does this make sense?

Wheel \(\leftrightarrow 4 \leftrightarrow 1 \rightarrow\) Car
No, we can have spare parts!
No, persons might be members of multiple committees!
Weaker form: aggregation

- **Aggregation**: “part of” relation
  - Parts can exist without the whole
  - Parts can be shared by multiple “wholes”
  - “Whole” can exist without its parts
Recipient – we need more than a name

- **Customer**
  - name
  - billing address
  - private or corporate

- **Generalization**
  - The same notation as in the use cases

```plaintext
Customer
#number: String [1]
#name: Name [1]
#address: String [0..1]

Corporate Customer
+contractNumber: String [1]
+contactName: Name [1]
+creditRating: String [1]

Private Customer
+creditCardNumber: String [1]
```
Abstract class

- We said **private or corporate** but the intention was **private, corporate and nothing else!**

- **Abstract class** cannot be directly instantiated. Instead, one instantiates an instance of a subclass.

```
Customer
#number: String [1]
#name: Name [1]
#address: String [0..1]
```

```
Corporate Customer
+contractNumber: String [1]
+contactName: Name [1]
+creditRating: String [1]
```

```
Private Customer
+creditCardNumber: String [1]
```
Abstract class: Another example

- **GeometricObject** is an abstract class
- **getArea()** and **getPerimeter()** are abstract methods
  - Not implemented in **GeometricObject** but in its subclasses
  - Why?
Exercise

For classes A, B, C, D which class diagram(s) is/are illegal?

• A) 

• B) 

• C) 

• D)
Exercise

For classes A, B, C, D which class diagram(s) is/are illegal?

- **A)**

- **B)**

- **C)**

- **D)**

illegal (cycle)

illegal (cycle)
Exercise (from Northumbria University)

• Consider Checkers game.
• Draw the class diagram
  • Ignore attributes and operations
Exercise (from Northumbria University)

- Consider Checkers game.
- Draw the class diagram
  - Ignore attributes and operations
Back to Orders: another kind of relation

- **Dependency**: the source uses the target in order to realize its functionality (but does not include an instance of it)
  - Lots of dependencies clutter the diagram.
    - Less frequently shown than association, generalization, composition, and aggregation
    - UML distinguishes more than 10 kinds of dependencies
    - Stereotypes are frequently omitted
  - **Usually**, target appears a parameter in source’s operations.

```
Order
+updatePriceFor(Product)
```

```
OrderLine
+lineItems {ordered}
1..*  1
```

```
Product
```
Examples of dependencies

- **<<create>>**
  - Order creates entries in the log book
- **<<use>>**
  - Order uses information about products
- **<<call>>**
  - Order calls a System operation to obtain the current time
- …
Cousin of an Abstract Class

- **Interface**: a class-like construct that contains only constants and abstract methods.
- Directly correspond to interfaces in Java and C#
- Subject to all relations we have seen
- Classes **require** (dependency) interfaces or **provide** (implementation) interfaces

[Diagram of UML 1 and UML 2 showing relationships between Playlist, Track Data, and iTunes Reader]

http://martinfowler.com/bliki/BallAndSocket.html
Cousin of an Abstract Class

• What are **advantages** and **disadvantages** of the UML 2 “ball and socket” notation for interfaces?

---

**UML 1**

- Playlist
  - «Interface» Track Data
  - iTunes Reader

**UML 2**

- Playlist
  - TrackData
  - iTunes Reader

http://martinfowler.com/bliki/BallAndSocket.html
Cousin of an Abstract Class

• What are **advantages** and **disadvantages** of the UML 2 “ball and socket” notation for interfaces?
  • How to represent multiple classes providing the same interface?
    − Solution due to Jim Rumbaugh and Martin Fowler

---

**UML 1**

![UML 1 Diagram](http://martinfowler.com/bliki/BallAndSocket.html)

**UML 2**

![UML 2 Diagram](http://martinfowler.com/bliki/BallAndSocket.html)
When I first saw the mated ball and socket notation I rather liked it. Since then, however, I've found no great inclination to use it. For simple cases the UML 1 style with sockets works well, but when things get more complicated I prefer to have explicit class boxes for the interfaces.

[Martin Fowler, http://martinfowler.com/bliki/BallAndSocket.html]
Putting it all together

**Order**
- +number: String [1]
- +placedOn: Date [1]
- +deliveredOn: Date [0..1]
- +approved: Boolean [1]
- +totalAmount: Money [1]
- +dispatch()
- +close()
- +updatePriceFor(Product)

**Customer**
- #number: String [1]
- #name: Name [1]
- #address: String [0..1]
- #getCreditRating()

**Corporate Customer**
- +contractNumber: String [1]
- +contactName: Name [1]
- +creditRating: String [1]

**Private Customer**
- +creditCardNumber: String [1]

---

if customer.getCreditRating is "poor" then approved must be true
Advantages / disadvantages

• Looks nice but cluttered?
  • 6 classes only, 2 without attributes/operations

• How can we reduce the amount of information?
  • Reduce the number of classes
  • Reduce the number of associations / generalizations / compositions / aggregations
    – Association classes [see next slide]
  • Reduce information per association
    – Drop names of associations/roles unless meaningful
  • Reduce the number of attributes
  • Reduce information per attribute
    – Multiplicity of [1] can be assumed
Association class

- OrderLine represents an association between Order and Product
  - No multiplicity: one instance per association
  - No additional associations
Correct or incorrect?
No, companies can play multiple roles in the same contract:
  e.g., supplier and maintenance provider
Correct or incorrect?

- Person
  - *
  - Competency level
    - *
    - Skill
Correct or incorrect?

Yes, for each skill a person would typically have only one competency level.
Further simplification

- Association is **stronger** than dependency
- We do not need to show both!
Order
+ number: String
+ placedOn: Date
+ deliveredOn: Date [0..1]
+ approved: Boolean
+ totalAmount: Money
+ dispatch()
+ close()
+ updatePriceFor(Product)

Customer
# number: String
# name: Name
# address: String [0..1]
# getCreditRating()

if customer.getCreditRating is "poor" then approved must be true

Corporate Customer
+ contractNumber: String
+ contactName: Name
+ creditRating: String

Private Customer
+ creditCardNumber: String

OrderLine

Product

1..*
Example: EasyShop system

Ellen and Portia live in Los Angeles. Ellen is a TV-host and stand-up comedian; Portia is an actress and both have a very busy schedule.

To schedule their shopping and household activities: EasyShop.
Ellen and Portia should be able

- to enter their presence for meals for each day
- to register invitations for guests for each meal
- to fix a menu for cold meals with given ingredients
- to choose whether they will cook in their agenda, and if so, fix a recipe for each warm meal
- to prepare a weekly shopping list, and fax the shopping list to the supermarket for delivery
- to keep track of ingredients in the kitchen; give a report when needed
- to show recipes (e.g., when cooking a dish)
What do we model?

• Identify the **actors**

Ellen and Portia  

?  

Supermarket
Candidate classes

Ellen and Portia should be able
- to enter their presence for meals for each day
- to register invitations for guests for each meal
- to fix a menu for cold meals with given ingredients
- to choose whether they will cook in their agenda, and if so, fix a recipe for each warm meal
- to prepare a weekly shopping list, and fax the shopping list to the supermarket for delivery
- to keep track of ingredients in the kitchen; give a report when needed
- to show recipes (e.g., when cooking a dish)

1. Identify noun phrases
2. Ignore actors
Ellen and Portia should be able

- to enter their presence for meals for each day
- to register invitations for guests for each meal
- to fix a menu for cold meals with given ingredients
- to choose whether they will cook in their agenda, and if so, fix a recipe for each warm meal
- to prepare a weekly shopping list, and fax the shopping list to the supermarket for delivery
- to keep track of ingredients in the kitchen; give a report when needed
- to show recipes (e.g., when cooking a dish)
Ellen and Portia should be able

- to enter their presence for meals for each day
- to register invitations for guests for each meal with given ingredients
- to choose whether they will cook in their agenda, and if so, fix a recipe for each warm meal
- to prepare a weekly shopping list, and fax the shopping list to the supermarket for delivery
- to keep track of ingredients in the kitchen; give a report when needed
- to show recipes (e.g., when cooking a dish)
Sketching the class diagram step by step

- **meals for each day**

  ![Class diagram for meals](image)

- **menu for cold meals with given ingredients**
  - menu should be recorded in ColdMeal

  ![Class diagram for cold meals](image)

- **fix a recipe for each warm meal**

  ![Class diagram for warm meals](image)

- **to show recipes**

  ![Class diagram for recipe](image)
So far

Day

Meal

WarmMeal

ColdMeal

Recipe

Ingredient

+showRecipe()
So far

Continue at the instruction group meeting
What is structure?

- **Structure**: the aggregate of elements of an entity in their relationships to each other. *Merriam-Webster*

- **Structural modelling**: discover the key data contained in the problem domain and to build a structural model of the objects.

---

Putting it all together

```
Order
+number: String [1]
+placedOn: Date [1]
+deliveredOn: Date [0..1]
+approved: Boolean [1]
+totalAmount: Money [1]
+dispatch()
+close()
+updatePriceFor(Product)

Customer
#number: String [1]
#name: Name [1]
#address: String [0..1]
#GetCreditRating()

if customer.getCreditRating is "poor" then approved must be true

OrderLine
+contractNumber: String [1]
+contactName: Name [1]
+creditRating: String [1]

Corporate Customer
+contractNumber: String [1]
+contactName: Name [1]
+creditRating: String [1]

Private Customer
+creditCardNumber: String [1]
```

---

How would we write this in UML?

```
Order
+number: String [1]
+placedOn: Date [1]
+deliveredOn: Date [0..1]
+totalAmount: Money [1]
+recipient: Name [1]
+items: OrderLine [1..*] {ordered}
```

**Multiplicity**
- how many objects may fill the attribute: m..n
- at least m
- at most n
- * denotes ∞

**Visibility**
- Which other classes can access this attribute?
- + (public) every other class
- - (private) no other class
- # (protected) only classes that inherit from Order
- ~ (package)

**Additional**
- Any kind of extra information needed about the attribute
Next week

Carnaval 2015
13 en 17 Februari
Eindhoven