CIS 771: Software Specifications

Lecture 11: Introduction to OCL & USE

Outline

- Overview of UML
- The role of OCL within UML
- Object model components
  - defines the states to which OCL constraints are applied
- A brief look at OCL
- A brief look at USE

What is Software Modeling?

- The designing of software applications before coding
- An essential part of large software projects, and helpful to small and medium-sized projects as well
- A model plays the analogous role in software development that blueprints and other plans (site maps, elevations, physical models) play in the building of a skyscraper
- Modeling can help ensure that...
  - business functionality is complete and correct,
  - end-user needs are met, and
  - program design supports scalability, robustness, security, extendibility, and other characteristics,

  before implementation in code renders changes difficult and expensive to make.

(Source: Introduction to OMG's Unified Modeling Language)

What Do We Want To Model?

- System requirements
- Basic situations in which a system is used
- Roles of system users
- System architecture
- Behavior of components
- Static and dynamic component interaction
- ...

What is UML?

- For the past 10 years various proposals have been put forth about how to perform
  - object-oriented analysis and modeling
  - i.e., describing system designs in terms of objects and their inter-relationships
- There have been at least 10 “methods” proposed
  - UML is the “combination” of the most popular (Booch, Rumbaugh, Jacobson)

Design by Committee

- As a consequence UML is
  - a collection of methods
  - rather than a single systematic approach to object modeling
- It’s becoming a de-facto standard
  - OMG oversees an ongoing a standardization effort
  - www.omg.org/uml
- If you don’t use it already, you will …

A Quick Overview of UML

- UML defines twelve types of diagrams divided into three categories...
  - *Structural Diagrams*
    - Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram.
  - *Behavior Diagrams*
    - Use Case Diagram, Sequence Diagram, Activity Diagram, Collaboration Diagram, and Statechart Diagram.
  - *Model Management Diagrams*
    - include Packages Diagram, Subsystems Diagram, and Models Diagram.
- Advanced features include the Object Constraint Language (OCL) and Action Semantics

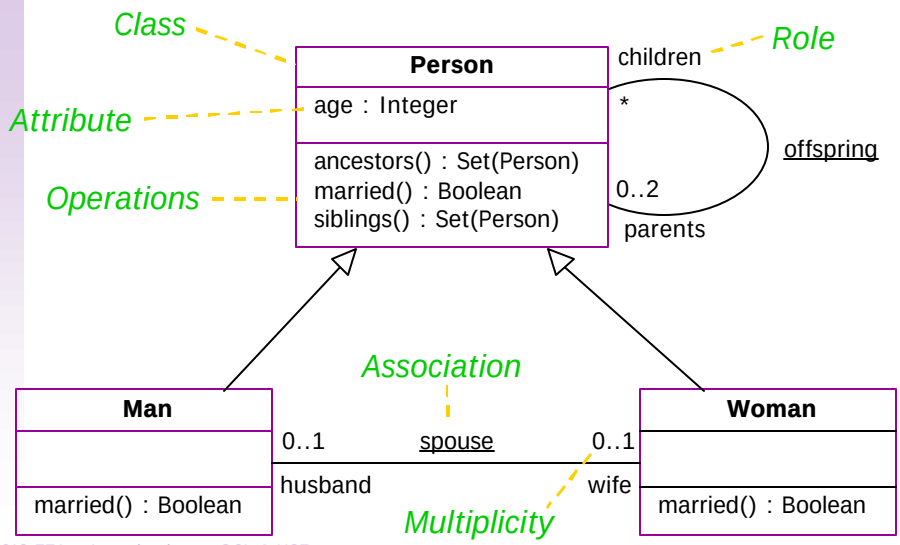
Contrast with Alloy

- Both UML and Alloy seek to describe...
  - the state space of a system
  - the transitions between states
  - in an implementation independent way
- UML has a much broader scope
  - implementation description (and synthesis) including system structure/architecture
  - problem domain modeling (requirements)
  - many different ways of capturing behavior (state diagrams, activity diagrams, sequence diagrams, etc.)
  - meant to be a bit imprecise to avoid committing to a particular platform
- Alloy has a much more rigorous semantic foundation

Structural Diagram Example

- Class Diagram
  - shows a set of classes, interfaces, and collaborations, and their relationships
  - the most common diagram found in modeling OO systems
  - address the static view of the system
  - we will emphasize class diagrams with the USE tool

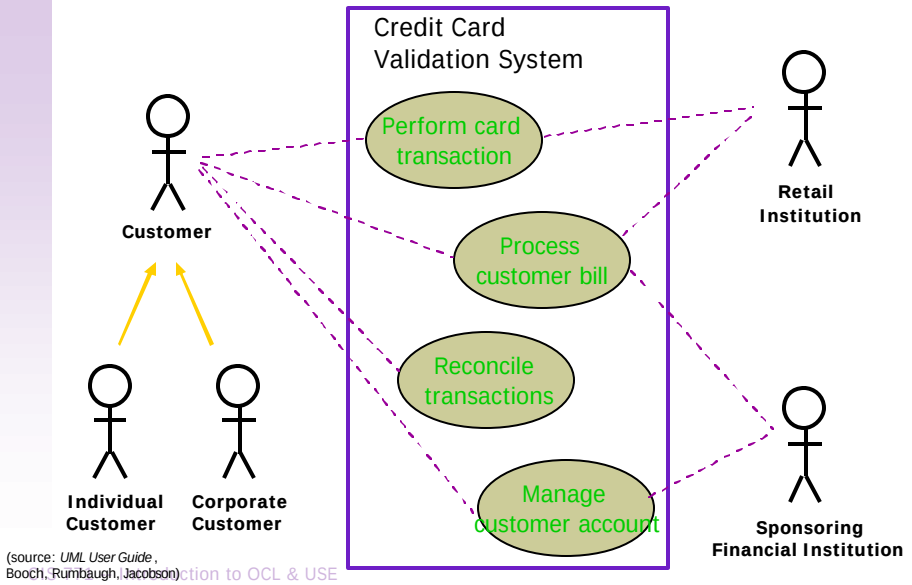
Class Diagram



Behavior Diagram Example

- Use Case Diagram
  - shows a set of cases of system use
  - shows actors that participate in each use case
  - used to model the context of a system
    - describing which actors “outside the system” interact with it
  - used to model the requirements of a system
    - “what” (not “how”) the system should do as viewed from outside the system

Use Case Diagram

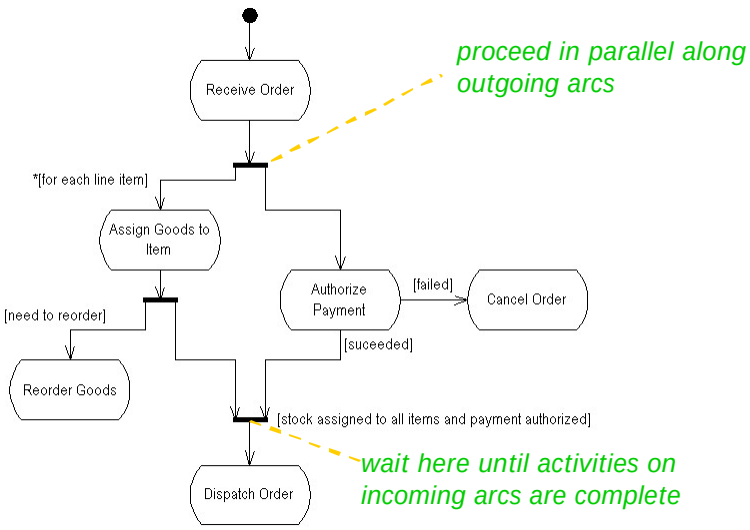


(source: *UML User Guide*, Booch, Rumbaugh, Jacobson)

Behavior Diagram Example

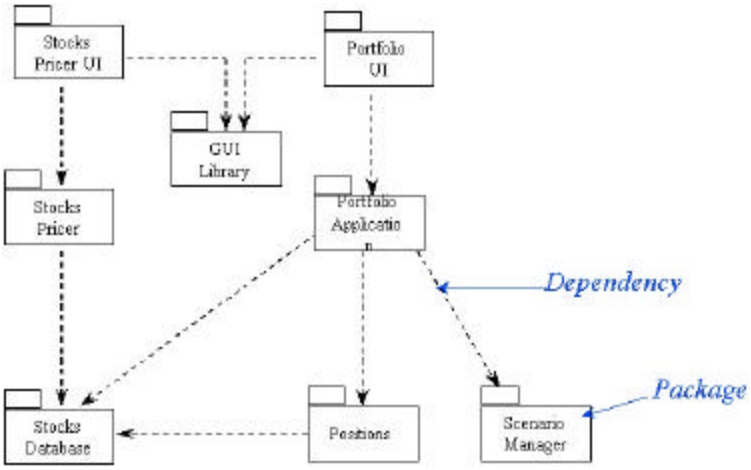
- Activity Diagram
  - shows flow from activity to activity within a system
  - addresses the dynamic view of a system

Activity Diagram



Module Management Diagram Example

- Packages Diagram



UML is Visual

- A picture is worth a thousand words
  - pictures can be misleading
- Most people can look at a UML description and get a “reasonable” idea of the parts of the system that are described
  - is reasonable good enough?
  - Usually, yes!

UML Shortcomings

- There are many different diagrams, and sometimes the purpose of one diagram isn't that different than the purpose of another
  - "You can model 80 percent of most problems by using about 20 percent of the UML."-- Grady Booch
- Many aspects of UML are imprecise
  - makes it difficult to build tools that carry out semantic reasoning
  - the Precise UML effort attempts to give a formal semantics for various aspects of UML (www.cs.york.ac.uk/puml)

UML-Based Processes

- A surprising amount of energy in the UML community focuses "process" guidelines
  - provide guidance as to the order of a team's activities
  - specify which artifacts should be developed and when they should be developed
  - direct the tasks of individual developers and the team is a whole
  - offer criteria for monitoring and measuring the project's products and activities
- The Rational Unified Process is one of the most popular UML-based software processes.

(Source: Grady Booch: *Object Solutions – Managing the Object-oriented Project*, Addison-Wesley, 1995.)

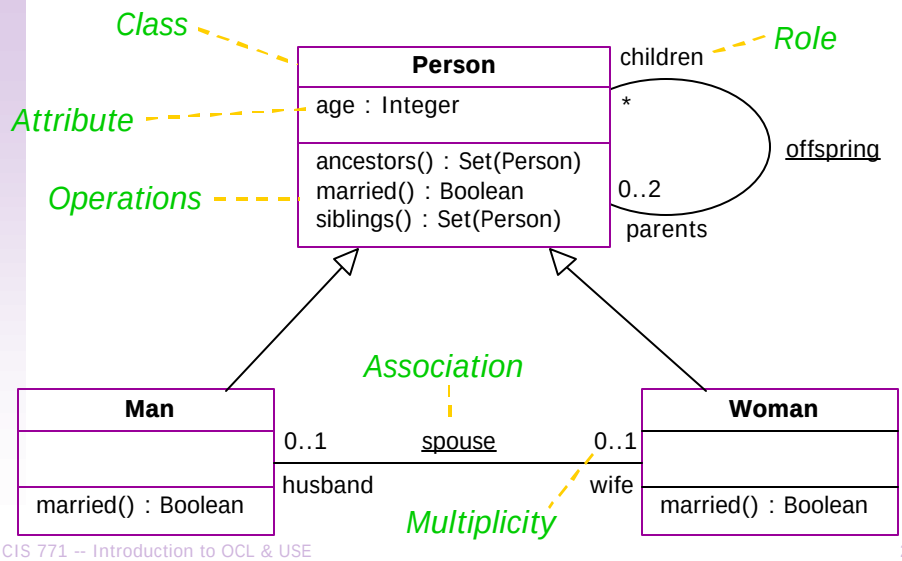
UML-Based Tools

- Rational Rose is one of the most popular
- ArgoUML is a free open-source tool set that supports many aspects of UML
- Many tools focus on *generating code templates* from class diagrams
- Other tools provide method for *automated reasoning* about system behavior expressed as statecharts
- USE (UML-based Specification Environment) supports the checking of system "snapshots" against OCL constraints

Object Constraint Language

- Allows specification of invariants, preconditions, postconditions, and guards of state transitions in UML
- Not a part of any particular diagram – OCL constraints may appear in several diagrams
- Often presented as annotations to different types of diagrams
- We will focus on OCL annotations for class diagrams

Class Diagram (revisited)



USE Object Models

Formalization of things to which an OCL expression can refer

- a set of *class names*
- a set of typed *attributes* associated with each class
- a set of *operations* – methods of a class that do not have side effects
  - these can be used in OCL expressions
- a set of *association names*, and for each association
  - a list of *participating classes*
  - *role name* for each end of the association
  - *multiplicities* for each end of the association
- a partial order < on the set of classes that captures the *generalization hierarchy*

Defining Classes in USE

USE Specification

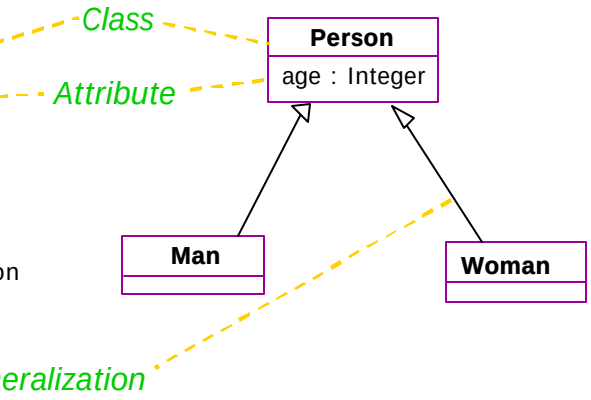
```
model Family

class Person
  age : Integer
end

class Man < Person
end

class Woman < Person
end
```

Class Diagram



USE Checks Snapshots

- USE allows one to create system "snapshots" and then it checks to see if the snapshots satisfy the specified OCL constraints
- Such snapshots
  - illustrate feasible object values and relationships
  - do not capture the set of all feasible objects

Command-line Instance Creation

Creating a simple snapshot at the command line

```
use> !create Ian:Man
use> !create Cormac:Man
use> !create Fiona:Woman
use> !create Erin:Woman
```

```
use> !set Ian.age = 56
use> !set Cormac.age = 45
use> !set Fiona.age = 38
use> !set Erin.age = 25
```

Object instances can also be created by...

- using the State – Create Object menu selection in the GUI
- dragging a class name to the Object Diagram view

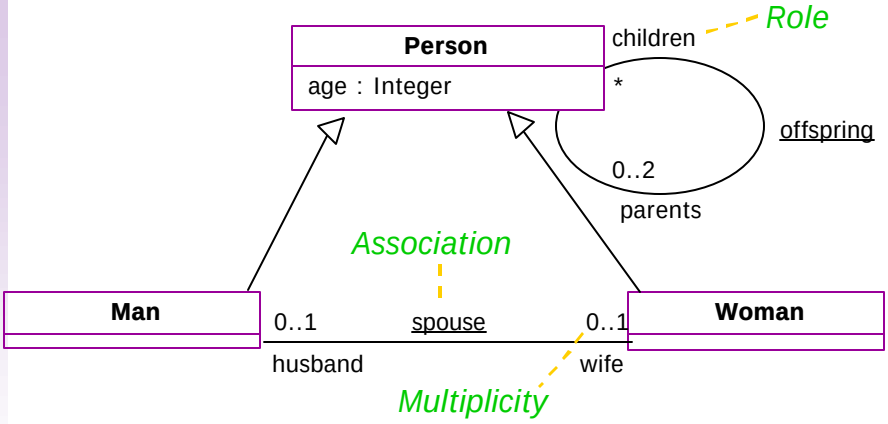
For You To Do…

- Pause the lecture
- Load the family-1.use model into USE (use the File – Open Specification menu option)
- Create an Object Diagram view
- Use the command line to create the Ian, Cormac, Fiona, and Erin instances with age attribute values as in the previous slide
- Use the State – Create Object menu option to create two additional Irish lasses (e.g., Molly and Keegan)
- Set Molly's age to be 22 and Keegan's age to be 19
- Use the drag & drop feature to create an additional Man instance

Associations

- Associations are a special class of attributes that represent object relationships
- Multiplicities
  - impose constraints on the cardinalities of the collections that embody the relationships
  - analogous to '!' and '?' in Alloy

Defining Associations in USE




```
association spouse between
  Man[0..1] role husband;
  Woman[0..1] role wife;
end
```

```
association offspring between
  Person[0..2] role parents;
  Person[*] role children;
end
```

Creating Association Snapshots

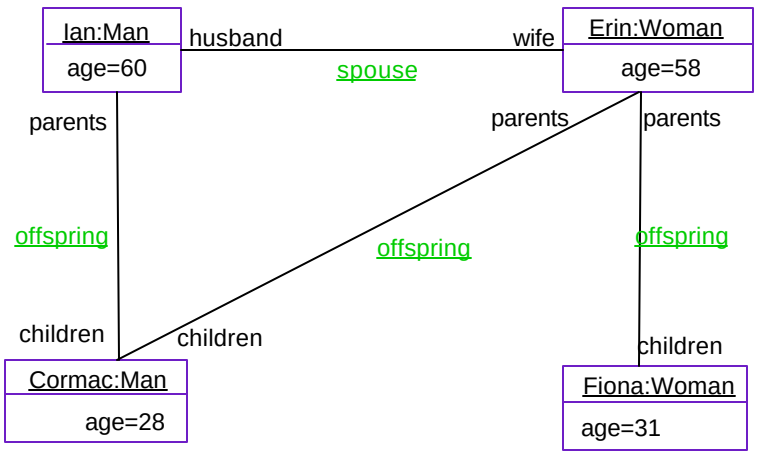
Inserting tuples into association relations

```
use> !insert (Cormac,Fiona) into spouse
use> !insert (Ian,Erin) into spouse
use> !insert (Ian,Molly) into spouse
```

The final snapshot above violates a multiplicity constraint

```
Multiplicity constraint violation in association `spouse':
  Object `Ian' of class `Man' is connected to 2 objects of
class `Woman‘ but the multiplicity is specified as `0..1'.
checking structure, found errors.
```


An Example Snapshot



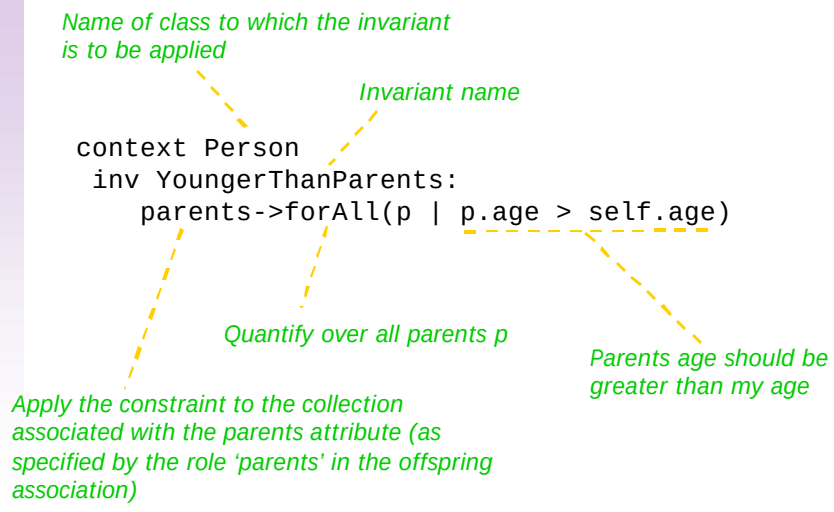
For You To Do…

- Pause the lecture
- Load the family-2.use model into USE
- Create an Object Diagram view
- Use either the command line or GUI to create the snapshot shown on the previous slide
- Right-click on the Object Diagram view pane and switch on the "show attribute value" box
- Create some additional tuples that cause the spouse multiplicity constraint to be violated (e.g., have Ian marry Molly)
- Undo the above tuple insert and insert a tuples that lead to the offspring multiplicity constraints being violated
- Play around with the other "views" present on the toolbar and on the View menu options

OCL Invariants

- Invariants are typically used to constrain the possible values of attributes
  - and thereby object relationships
- OCL provides a rich expression language for expressing invariants
  - we'll begin to study this in the next lecture

OCL Invariant Example



Checking Invariants in USE

- Select the 'class invariant view'
- This view holds a table that reports on the status of each invariant in the current snapshot (whether the invariant holds or fails)

For You To Do…

- Pause the lecture
- Load the family-3.use model into USE
- Create an Object Diagram view and a Class Invariant view
- Create a snapshot that violates the YoungerThanParents invariant

Contrasting with Alloy

- Alloy's constraint language is less expressive, and this allows its checking to be more powerful
  - Alloy checks against *all possible model instances* within a given scope
  - USE checks against only those instances that you create in scripts
- Alloy's notion of state is more abstract (sets, binary relations); USE's notion of state is more tied to OO architecture (classes, generalization, attributes, collections, etc.)
- You might imagine carrying out an initial design in Alloy to capture semantic entities and the relationships between them, then moving to USE to capture an initial architecture where many of the Alloy constraints can be carried over and expressed in OCL.

Summary

- UML is the most widely-used modeling language in software development
- There are many aspects to UML
  - many diagrams, processes, tools, etc.
- OCL is UML's constraint language and it allows constraints to be attached to various diagrams
- We will focus on USE's view of OCL: associating constraints with class diagrams and checking snapshots of systems
- USE checks multiplicity constraints, invariants, and pre/post conditions

Next Lecture

- Overview of expression language used for defining invariants
- OCL Types
- The Academia enterprise as a running example
- Contrasting with Alloy