Module 2: Introduction to UML

- Background
- What is UML for?
- Building blocks of UML
- Appendix:
  - Architecture & Views
  - Process for Using UML
OO languages appear mid 70’s to late 80’s (cf. Budd: communication and complexity)

Between ’89 and ’94, OO methods increased from 10 to 50.

Unification of ideas began in mid 90’s.
- Rumbaugh joins Booch at Rational ’94
- v0.8 draft Unified Method ’95
- Jacobson joins Rational ’95
- UML v0.9 in June ’96

- UML 1.0 offered to OMG in January ’97
- UML 1.1 offered to OMG in July ’97
  - Maintenance through OMG RTF
- UML 1.2 in June ’98
- UML 1.3 in fall ’99
- UML 1.5 [http://www.omg.org/technology/documents/formal/uml.htm]
- UML 2.0 underway [http://www.uml.org/]

IBM-Rational now has Three Amigos
- Grady Booch - Fusion
- James Rumbaugh – Object Modeling Technique (OMT)
- Ivar Jacobson – Object-oriented Software Engineering: A Use Case Approach (Objectory)
  - (And David Harel - StateChart)


[According to OMG RTF]
Unified Modeling Language (UML)

- An effort by IBM (Rational) – OMG to standardize OOA&D notation

- Combine the best of the best from
  - Data Modeling (Entity Relationship Diagrams);
    Business Modeling (work flow); Object Modeling
  - Component Modeling (development and reuse - middleware, COTS/GOTS/OSS/…)

- Offers vocabulary and rules for communication
- *Not* a process but a language

*de facto* industry standard
UML is for Visual Modeling

A picture is worth a thousand words!

- standard graphical notations: Semi-formal
- for modeling enterprise info. systems, distributed Web-based applications, real time embedded systems, ...

- **Specifying & Documenting**: models that are precise, unambiguous, complete
  - UML symbols are based on well-defined syntax and semantics.
  - analysis, architecture/design, implementation, testing decisions.
- **Construction**: mapping between a UML model and OOPL.
Three (3) basic **building blocks** of UML (cf. Harry)

- **Things** - important modeling concepts
- **Relationships** - tying individual things
- **Diagrams** - grouping interrelated collections of things and relationships
3 basic building blocks of UML - Things

- **UML 1.x**
  - **Structural** — nouns/static of UML models (irrespective of time).
  - **Behavioral** — verbs/dynamic parts of UML models.
  - **Grouping** — organizational parts of UML models.
  - **Annotational** — explanatory parts of UML models.
Structural Things in UML- 7 Kinds (Classifiers)

- Nouns.
- Conceptual or physical elements.

**Class**
- Student
  - std_id
  - grade
  - changeLevel()
  - setGrade()
  - getGrade()

**Active Class**
- (processes/threads)
  - Event Mgr
    - thread
    - time
    - Start
    - suspend()
    - stop()

**Component**
- (replaceable part, realizes interfaces)

**Interface**
- (collection of externally Visible ops)
  - IGrade
  - setGrade()
  - getGrade()

**Node**
- (computational resource at run-time, processing power w. memory)

**Use Case**
- Register for Courses
  - (a system service - sequence of Interactions w. actor)

**Collaboration**
- Manage Course Registration
  - (chain of responsibility shared by a web of interacting objects, structural and behavioral)
Behavioral Things in UML

- Verbs.
- Dynamic parts of UML models: “behavior over time”
- Usually connected to structural things.

Two primary kinds of behavioral things:

- **Interaction**
  a set of objects exchanging messages, to accomplish a specific purpose.

  ![Interaction Diagram]

  harry: Student
  name = “Harry Kid”

  ask-for-an-A

  katie: Professor
  name = “Katie Holmes”

- **State Machine**
  specifies the sequence of states an object or an interaction goes through during its lifetime in response to events.

  ![State Machine Diagram]

  inStudy

  received-an-A/
  buy-beer

  inParty

  sober/turn-on-PC
Grouping Things in UML: **Packages**

- For organizing elements (structural/behavioral) into groups.
- Purely conceptual; only exists at development time.
- Can be nested.
- Variations of packages are: Frameworks, models, & subsystems.

Annotational Things in UML: **Note**

- Explanatory/Comment parts of UML models - usually called adornments
- Expressed in informal or formal text.

```
operation()
{
    for all g in children
    g.operation()
}
```
3 basic building blocks of UML - Relationships

1. **Associations**

   *Structural* relationship that describes a set of links, a link being a connection between objects.

   (UML2.0: The semantic relationship between two or more classifiers that involves connections among their instances.)

   **variants:** aggregation & composition

2. **Generalization**

   a specialized element (the child) is more specific the generalized element.

3. **Realization**

   one element guarantees to carry out what is expected by the other element.

   *(e.g, interfaces and classes/components; use cases and collaborations)*

4. **Dependency**

   a change to one thing (independent) may affect the semantics of the other thing (dependent).

   *(direction, label are optional)*
A connected graph: Vertices are things; Arcs are relationships/behaviors.

UML 1.x: 9 diagram types.

**Structural Diagrams**
*Represent the *static* aspects of a system.*
- Class;
- Object
- Component
- Deployment

**Behavioral Diagrams**
*Represent the *dynamic* aspects.*
- Use case
- Sequence;
- Collaboration
- Statechart
- Activity

UML 2.0: 12 diagram types

**Structural Diagrams**
- Class;
- Object
- Component
- Deployment
- Composite Structure
- *Package*

**Behavioral Diagrams**
- Use case
- Statechart
- Activity

**Interaction Diagrams**
- Sequence;
- Communication
- Interaction Overview
- Timing
Diagrams in UML

The UTD wants to computerize its registration system

- The Registrar sets up the curriculum for a semester
- Students select 3 core courses and 2 electives
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course offering rosters after students register
- Users of the registration system are assigned passwords which are used at logon validation

What’s most important?
Diagrams in UML – Actors in Use Case Diagram

- An actor is someone or some thing that must interact with the system under development.

The UTD wants to computerize its registration system.

- The Registrar sets up the curriculum for a semester.

- Students select 3 core courses and 2 electives.

- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester.

- Students may use the system to add/drop courses for a period of time after registration.

- Professors use the system to set their preferred course offerings and receive their course offering rosters after students register.

- Users of the registration system are assigned passwords which are used at logon validation.
The UTD wants to computerize its registration system.

- The **Registrar** sets up the curriculum for a semester.
- **Students** select 3 core courses and 2 electives.
- Once a student registers for a semester, the **billing system** is notified so the student may be billed for the semester.
- Students may use the system to add/drop courses for a period of time after registration.
- **Professors** use the system to set their preferred course offerings and receive their course offering rosters after students register.
- Users of the registration system are assigned passwords which are used at logon validation.

**Use Cases in Use Case Diagram**

- A use case is a sequence of interactions between an actor and the system.
Use case diagrams depict the relationships between actors and use cases.
A **uses** relationship shows behavior common to one or more use cases.

An **extends** relationship shows optional/exceptional behavior.
Diagrams in UML – Flow of Events for each use case:

Typical contents:
- How the use case starts and ends
- Normal flow of events (focus on the normal first!)
- Alternate/Exceptional flow of events

Flow of Events for Creating a Course

- This use case begins after the Registrar logs onto the Registration System with a valid password.
- The registrar fills in the course form with the appropriate semester and course related info.
- The Registrar requests the system to process the course form.
- The system creates a new course, and this use case ends
Diagrams in UML – Interaction Diagrams

A use case diagram presents an outside view of the system.

Then, how about the inside view of the system?

- **Interaction diagrams** describe how use cases are realized in terms of interacting objects.

- Two types of interaction diagrams
  - Sequence diagrams
  - Collaboration (Communication) diagrams
A sequence diagram displays object interactions arranged in a time sequence.

This use case begins after the Registrar logs onto the Registration System with a valid password.

The registrar fills in the course form with the appropriate semester and course related info.

The Registrar requests the system to process the course form.

The system creates a new course, and this use case ends.

Traceability!
Diagrams in UML – Collaboration (Communication)

- Displays object interactions organized around objects and their direct links to one another.
- Emphasizes the structural organization of objects that send and receive messages.

```
: Registrar

course form : CourseForm

theManager : CurriculumManager

1: set course info
2: request processing
3: add course
4: <<create>>
```

```
: Registrar

course form: CourseForm

1: set course info
2: request processing
3: add course

: Registrar

Course

course form: CourseForm

1: set course info
2: request processing
3: add course
4: <<create>>

theManager : CurriculumManager
```
Diagrams in UML – Collaboration (Communication)

What would be the corresponding collaboration diagram?

Which use case could this be for? How about <---- (see M3.2)
Skip the following for now: In M3.2
public class Selection {
    private Purchase myPurchase = new Purchase();
    private Payment myPayment;
    public void purchase() {
        myPurchase.buyMajor();
        myPurchase.buyMinor();
        myPayment = new Payment(cashTender);
        //...
    }
    //...
}
Interactions - Modeling Actions

Simple → Call → Return → Send

asynchronous in 2.0 (stick arrowhead) – no return value expected at end of callee activation
activation of caller may end before callee’s

half arrow in 1.x

\[ \text{c : Client} \]

\[ \text{p : PlanningAgent} \]

\[ \text{<<create>>} \]

\[ \text{TicketAgent} \]

\[ \text{<<destroy>>} \]

\[ \text{notify()} \]

\[ \text{destroy: e.g., in C++, manual garbage collection; in Java/C#, unnecessary} \]

\[ \text{natural death/self destruction} \]

loop

\[ \text{setItenerary( i )} \]

\[ \text{calculateRoute()} \]

\[ \text{for each conference} \]

\[ \text{return value} \]

\[ \text{call on self} \]

\[ \text{end of object life} \]

\[ \text{send} \]

\[ \text{return} \]

\[ \text{route} \]

\[ \text{actual parameter} \]
2 forms of sd:

- **Instance** sd: describes a specific scenario in detail; no conditions, branches or loops.
- **Generic** sd: a use case description with alternative courses.

Here, conditional or concurrency?
Interaction Diagram: sequence vs communication

Observer design pattern

Objects:
- p: StockQuotePublisher
- s1: StockQuoteSubscriber
- s2: StockQuoteSubscriber

Activations:
- Procedure call, RMI, JDBC, ...
- Shows duration of execution
- Shows call stack
- Return message

Implicit at end of activation
Explicit with a dashed arrow

Time:
- 1: attach(s1)
- 2: attach(s2)
- 3: notify()
- 4: update()
- 5: update()
- 6: getState()
- 7: getState()
Skip end: resume
A class diagram shows the existence of classes and their relationships

Recall: A class is a collection of objects with common structure, common behavior, common relationships and common semantics

Some classes are shown through the objects in sequence/collaboration diagram
Diagrams in UML - Class Diagrams: static structure in the system

- Naming & (often) 3 Sections;
- Inheritance (as before);
- Relationships - Multiplicity and Navigation
Diagrams in UML – Object Diagrams

- Shows a set of objects and their relationships.
- As a static snapshot.

Anything wrong?
Prepare a UML class diagram with all the needed classes and relationships, as precisely as possible, from the instance diagram below:

Any issues?
Diagrams in UML – State Transition Diagram (Statechart Diagram)

- The life history (often of a given class: from class to object behavior)
- States, transitions, events that cause a transition from one state to another
- Actions that result from a state change

What life history/class is this for?
What would be the value of “count”, after two(2) students have been added to the course?
Until what stage?
Diagrams in UML – Activity Diagrams

- A special kind of statechart diagram that shows the flow *from activity to activity*.

```
Initialize course
```
```
Add student
```
```
Notify Registrar
```
```
Notify Billing
```

What is this for? Traceability???
Diagrams in UML – Component Diagram

shows the organizations and dependencies among a set of components \textit{(mostly \texttt{uses})}.

In UML 1.1, a component represented \textit{implementation} items, such as files and executables;

\ldots

( In \textit{UML 2.0}, a component is a replaceable/reusable, \textit{architecture}/design-time construct w. interfaces)
Diagrams in UML – Deployment Diagram

- shows the configuration of run-time processing elements and the software processes living on them.
- visualizes the distribution of components across the enterprise.
3 basic building blocks of UML - Diagrams

Using UML Concepts in a Nutshell

- Display the boundary of a system & its major functions using use cases and actors
- Illustrate use case realizations with interaction diagrams
- Represent a static structure of a system using class diagrams
- Model the behavior of objects with state transition diagrams
- Reveal the physical implementation architecture with component & deployment diagrams
- Extend your functionality with stereotypes

Here, UML 1.x first (UML 2.0 later)

- Use case
- Sequence; Collaboration (Communication)
- Class; Object
- Statechart Activity
- Component Deployment
Summary

- Background

- What is UML for (both 1.x and 2.0)?
  for visualizing, specifying, constructing, and documenting models

- Building blocks of UML
  Things, Relationships (4 kinds) and Diagrams (9 different kinds)
Points to Ponder, for now

- What kind of use case diagram is your program for?
  1. List main actors
  2. List main use cases
  3. Associate the actors with the use cases

- What kind of class diagram would you need?
  1. List main classes
  2. List main attributes
  3. List main operations, …
  4. Associate classes
  5. Multiplicity
  6. Visibility
  7. Refine relationships

- What kind of sequence diagrams would you need?

- …
Module 2: Introduction to UML - Appendix
Extensibility of UML

- Stereotypes (<< >>) can be used to extend the UML notational elements
- Stereotypes may be used to classify and extend associations, inheritance relationships, classes, and components
- Examples:
  - Class stereotypes: boundary, control, entity, utility, exception
  - Inheritance stereotypes: uses and extends
  - Component stereotypes: subsystem

*Stereotypes* — extends vocabulary (metaclass in UML metamodel)
*Tagged values* — extends properties of UML building blocks (i.e., metamodel)
*Constraints* — extend the semantics of UML building blocks.

*More on this later*
UML is for visualizing, specifying, constructing, and documenting with emphasis on system architectures (things in the system and relationships among the things) from five different views.

**Software architecture** = \{Elements, Forms, Rationale/Constraints\}

**Five views:**
- the *logical* view, which is the object model of the design (when an object-oriented design method is used),
- the *process* view, which captures the concurrency and synchronization aspects of the design,
- the *physical* view, which describes the mapping(s) of the software onto the hardware and reflects its distributed aspect,
- the *development* view, which describes the static organization of the software in its development environment.
- The *scenario* view, which consists of a few selected *use cases* or *scenarios*, illustrates the description of an architecture, and also helps it evolve as well.

The diagram illustrates the relationships between the views:
- **Logical View**: End user functionality
- **Process View**: Integrators, performance, scalability
- **Physical View**: Development View
- **Development View**: Programmers, software management
- **Scenarios**: System engineers, topology, communications
The "4+1" View Model of Software Architecture: Notation and Example

Components

- Class
- Class Utility
- Parameterized Class
- Class category

Connectors

- Association
- Containment, Aggregation
- Usage
- Inheritance
- Instanciation

Figure 2 — Notation for the logical blueprint

Figure 3 — a. Logical blueprint for the Téléc PABX, b. Blueprint for an Air Traffic Control System
The “4+1” View Model of Software Architecture: Notation and Example

Figure 4 — Notation for the Process blueprint

Figure 5 — Process blueprint for the Télic PABX (partial)
The "4+1" View Model of Software Architecture: Notation and Example

Components
- Module
- Subsystem

Connectors
- Reference
- Compilation dependency (include, "with")

Layer

Figure 5 — Notation for the Development blueprint

Figure 6 — The 5 layers of Hughes Air Traffic Systems (HATS)
The “4+1” View Model of Software Architecture: Notation and Example

Figure 7 — Notation for the Physical Blueprint

A small PABX physical architecture with

Figure 8 — Physical blueprint for the PABX

Figure 10 — Physical blueprint for a larger PABX showing process allocation
The “4+1” View Model of Software Architecture: Notation and Example

Figure 11 — Embryo of a scenario for a local call—selection phase
The "4+1" View Model of Software Architecture: Notation and Example

Figure 12: Mapping from Logical to Process view
The “4+1” View Model of Software Architecture: Notation and Example

<table>
<thead>
<tr>
<th>Title Page</th>
<th>Change History</th>
<th>Table of Contents</th>
<th>List of Figures</th>
</tr>
</thead>
</table>

1. Scope  
2. References  
3. Software Architecture  
4. Architectural Goals & Constraints  
5. Logical Architecture  
6. Process Architecture  
7. Development Architecture  
8. Physical Architecture  
9. Scenarios  
10. Size and Performance  
11. Quality  

Appendices  
A. Acronyms and Abbreviations  
B. Definitions  
C. Design Principles  

Figure 13 — Outline of a Software Architecture Document


### The “4+1” View Model of Software Architecture: Notation and Example

<table>
<thead>
<tr>
<th>View</th>
<th>Logical</th>
<th>Process</th>
<th>Development</th>
<th>Physical</th>
<th>Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Components</strong></td>
<td>Class</td>
<td>Task</td>
<td>Module, Subsystem</td>
<td>Node</td>
<td>Step, Scripts</td>
</tr>
<tr>
<td><strong>Connectors</strong></td>
<td>association, inheritance, containment</td>
<td>Rendez-vous, Message, broadcast, RPC, etc.</td>
<td>compilation dependency, “with” clause, “include”</td>
<td>Communication medium, LAN, WAN, bus, etc.</td>
<td></td>
</tr>
<tr>
<td><strong>Containers</strong></td>
<td>Class category</td>
<td>Process</td>
<td>Subsystem (library)</td>
<td>Physical subsystem</td>
<td>Web</td>
</tr>
<tr>
<td><strong>Stakeholders</strong></td>
<td>End-user</td>
<td>System designer, integrator</td>
<td>Developer, manager</td>
<td>System designer</td>
<td>End-user, developer</td>
</tr>
<tr>
<td><strong>Concerns</strong></td>
<td>Functionality</td>
<td>Performance, availability, S/W fault-tolerance, integrity</td>
<td>Organization, reuse, portability, line-of-product</td>
<td>Scalability, performance availability</td>
<td>Understand-ability</td>
</tr>
<tr>
<td><strong>Tool support</strong></td>
<td>Rose</td>
<td>UNAS/SALE DADS</td>
<td>Apex, SoDA</td>
<td>UNAS, Openview DADS</td>
<td>Rose</td>
</tr>
</tbody>
</table>

Table 1 — Summary of the “4+1” view model
UML is for visualizing, specifying, constructing, and documenting with emphasis on system architectures (things in the system and relationships among the things) from five different views.

Architecture - set of significant decisions regarding:

- Organization of a software system.
- Selection of structural elements & interfaces from which a system is composed.
- Behavior or collaboration of elements.
- Composition of structural and behavioral elements.
- Architectural style guiding the system.
Views

Use Case View
- Use Case Analysis is a technique to capture business process from user’s perspective.
- Encompasses the behavior as seen by users, analysts and testers.
- Specifies forces that shape the architecture.
- Static aspects in use case diagrams; Dynamic aspects in interaction (statechart and activity) diagrams.

Design View
- Encompasses classes, interfaces, and collaborations that define the vocabulary of a system.
- Supports functional requirements of the system.
- Static aspects in class and object diagrams; Dynamic aspects in interaction diagrams.

Process View
- Encompasses the threads and processes defining concurrency and synchronization.
- Addresses performance, scalability, and throughput.
- Static and dynamic aspects captured as in design view; emphasis on active classes.

Implementation View
- Encompasses components and files used to assemble and release a physical system.
- Addresses configuration management.
- Static aspects in component diagrams; Dynamic aspects in interaction diagrams.

Deployment View
- Encompasses the nodes that form the system hardware topology.
- Addresses distribution, delivery, and installation.
- Static aspects in deployment diagrams; Dynamic aspects in interaction diagrams.
Rules of UML

- Well formed models — *semantically self-consistent and in harmony with all its related models.*

- Semantic rules for:
  - Names — what you can call things.
  - Scope — context that gives meaning to a name.
  - Visibility — how names can be seen and used.
  - Integrity — how things properly and consistently relate to one another.
  - Execution — what it means to run or simulate a dynamic model.

- Avoid models that are

  Elided — certain elements are hidden for simplicity.

  Incomplete — certain elements may be missing.

  Inconsistent — no guarantee of integrity.
Process for Using UML

How do we use UML as a notation to construct a good model?

- **Use case driven** — use cases are primary artifact for defining behavior of the system.

- **Architecture-centric** — the system’s architecture is primary artifact for conceptualizing, constructing, managing, and evolving the system.

- **Iterative and incremental** — managing streams of executable releases with increasing parts of the architecture included.

The Rational Unified Process (RUP)
Process for Using UML - Iterative Life Cycle

• It is planned, managed and predictable …almost
• It accommodates changes to requirements with less disruption
• It is based on evolving executable prototypes, not documentation
• It involves the user/customer throughout the process
• It is risk driven

**Primary phases**

- **Inception** — seed idea is brought up to point of being a viable project.
- **Elaboration** — product vision and architecture are defined. ([http://www.utdallas.edu/~chung/OOAD_SUMMER04/HACS_vision_12.doc](http://www.utdallas.edu/~chung/OOAD_SUMMER04/HACS_vision_12.doc))
- **Construction** — brought from architectural baseline to point of deployment into user community.
- **Transition** — turned over to the user community.
Process for Using UML - Iterative Approach

Three Important Features

- Continuous integration - Not done in one lump near the delivery date
- Frequent, executable releases - Some internal; some delivered
- Attack risks through demonstrable progress - Progress measured in products, not documentation or engineering estimates

Resulting Benefits

- Releases are a forcing function that drives the development team to closure at regular intervals - Cannot have the “90% done with 90% remaining” phenomenon
- Can incorporate problems/issues/changes into future iterations rather than disrupting ongoing production
- The project’s supporting elements (testers, writers, toolsmiths, QA, etc.) can better schedule their work
Initial Project Risks  
Initial Project Scope

Plan Iteration N
• Cost
• Schedule

Develop Iteration N
• Collect cost and quality metrics

Assess Iteration N

Iteration N

Define scenarios to address highest risks

Revise Overall Project Plan
• Cost
• Schedule
• Scope/Content

Revise Project Risks
• Reprioritize

Risks Eliminated
Process for Using UML - Use Cases Drive the Iteration Process

Iteration 1 → Iteration 2 → Iteration 3

“Mini-Waterfall” Process
- Results of previous iterations
- Up-to-date risk assessment
- Controlled libraries of models, code, and tests

Selected scenarios

- Iteration Planning
- Reqs Capture
- Analysis & Design
- Implementation
- Test
- Prepare Release

Each iteration is defined in terms of the scenarios it implements.
Points to Ponder

Are Sequence and Collaboration Diagrams Isomorphic?
Points to Ponder

- How much unification does UML do?
  Consider the Object Model Notation on the inside cover on the front and back of the textbook "Object Oriented Modeling and Design" by Rumbaugh, et.al.
  1. List the OMT items that do not exist in UML
  2. List the UML items that do not exist in OMT
  3. For those items of OMT for which UML equivalents exist, map the notation to UML.

- Where would you want to use stereotypes?
- Model the “Business Process” on page 6 in UML.
- Map the four (4) phases of the RUP to the traditional software lifecycle.
- If an object refers to a concept, can an object refer to a concept of a concept? Consider some examples.
- What would be the essential differences between a property and an attribute? Consider some examples.
- What is the syntax and semantics of a class diagram?
- In Component-Based Software Engineering (CBSE), components are the units, or building blocks, of a (distributed) software system. What kind of building blocks of UML can be components for CBSE?