Conceptual Modeling

- Abstract (visual) representation of the problem domain
- Serves to enhance understanding of complex problem domains.
- Provides a basis for communication among project team members.

Conceptual Modeling

- A good conceptual model should NOT reflect a solution bias.
- Should model the problem domain, not the solution domain.
- Initial conceptual model may be rough and general.
- May be refined during incremental development.
What is Visual Modeling?

“Modeling captures essential parts of the system.”
Dr. James Rumbaugh

Visual Modeling is modeling using standard graphical notations

Visual Modeling Manages Complexity

The human mind can only handle 7 plus or minus 2 things at once
Developing a Conceptual Model

- Conceptual model is derived from **analysis** of the problem domain.
- Several different ways to analyze a problem
  - Classical (Structured) Analysis
    - Identify and model the basic *functions* and associated *data flows* of the system
  - Object-oriented Analysis (OOA)
    - Identify the primary *objects* of the system
    - Model the *relationships* among objects.

Structured Analysis

- Several well-developed methodologies exist for performing structured analysis:
  - DeMarco (1979)
  - Yourdan and Constantine (1979)
  - Page-Jones (1980)
  - Gane and Sarsan (1979, 1982)
  - Ward and Mellor (1985)
  - Hatley and Pirbhai (1987)  

{ Real-time extensions }
Structured Analysis

- Basic idea is to model information flows
- Primary modeling element is the *data flow diagram* (DFD)
- DFD Notation:
  - **External Entity**: Producer or consumer of information outside of system boundary
  - **Process**: A function that transforms information
  - **Data object**: Represents the flow of data
  - **Data store**: Repository of data

DFD Example

(Adapted from Schach, 1996)

- **Customer**
  - Order
  - Verify Order
  - Credit Status
  - Invoice
  - Assemble Order

- **Supplier**
  - Place Order with Supplier
  - Items to be Ordered From Supplier
  - Contact Info.

- **Pending Orders**
  - Items on Hand
  - Batched Order

- **Customer Data**

- **Product Data**
Difference Between Structured Analysis and OOA

Library Information System

Object-Oriented Analysis
Decompose by objects (concepts)
- Catalog
- Librarian
- Book
- Library

Structured Analysis
Decompose by processes (functions)
- System
- Record Loans
- Add Resources
- Report Fines

Unified Modeling Language (UML)
- Provides Standard Modeling Notation for OOA/D
- Developed by the “Three Amigos” of OOA/D
  - Grady Booch--Booch Method
  - James Rumbaugh--(OMT)
  - Ivar Jacobson--OOSE (Objectory)
- Many Industries and Experts Have Contributed to UML Development
- Emerging as a De facto Standard
Elements of UML
• Provides standard, well-defined syntax and semantics for modeling various aspects of software development process.
• Types of UML models:
  – Use-case diagrams
  – Static structure diagrams
  – behavior diagrams
  – implementation diagrams
• Complete UML Documentation available at:
  – http://www.rational.com/uml

Using UML
• The following material is taken from the tutorial: Analysis and Design with UML.
• The entire tutorial is available as a zipped PowerPoint file from the following URL:
• A copy of the tutorial slides in Adobe Acrobat format is provided on the Class web site.
Putting the UML to Work

• The ESU University wants to computerize their registration system
  – The Registrar sets up the curriculum for a semester
    • One course may have multiple course offerings
  – Students select 4 primary courses and 2 alternate courses
  – 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 receive their course offering rosters
  – Users of the registration system are assigned passwords which are used at logon validation

Actors

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

Registrar
Professor
Student
Billing System
Use Cases

• A use case is a pattern of behavior the system exhibits
  – Each use case is a sequence of related transactions performed by an actor and the system in a dialogue

• Actors are examined to determine their needs
  – Registrar -- maintain the curriculum
  – Professor -- request roster
  – Student -- maintain schedule
  – Billing System -- receive billing information from registration

Documenting Use Cases

• A flow of events document is created for each use case
  – Written from an actor point of view

• Details what the system must provide to the actor when the use cases is executed

• Typical contents
  – How the use case starts and ends
  – Normal flow of events
  – Alternate flow of events
  – Exceptional flow of events
Maintain Curriculum
Flow of Events

• This use case begins when the Registrar logs onto the Registration System and enters his/her password. The system verifies that the password is valid (E-1) and prompts the Registrar to select the current semester or a future semester (E-2). The Registrar enters the desired semester. The system prompts the professor to select the desired activity: ADD, DELETE, REVIEW, or QUIT.
  • If the activity selected is ADD, the S-1: Add a Course subflow is performed.
  • If the activity selected is DELETE, the S-2: Delete a Course subflow is performed.
  • If the activity selected is REVIEW, the S-3: Review Curriculum subflow is performed.
  • If the activity selected is QUIT, the use case ends.
  • ...

Use Case Diagram

• Use case diagrams are created to visualize the relationships between actors and use cases

[Diagram showing relationships between Student, Registrar, Professor, Billing System, and Processes like Maintain Schedule, Request Course Roster, Maintain Curriculum, and Registrar]
Uses and Extends Use Case Relationships

• As the use cases are documented, other use case relationships may be discovered
  – A *uses* relationship shows behavior that is common to one or more use cases
  – An *extends* relationship shows optional behavior

![Diagram showing uses and extends relationships]

Use Case Realizations

• The use case diagram presents an outside view of the system
• Interaction diagrams describe how use cases are realized as interactions among societies of objects
• Two types of interaction diagrams
  – Sequence diagrams
  – Collaboration diagrams
Sequence Diagram
• A sequence diagram displays object interactions arranged in a time sequence

```
- Student
  registration form
  registration manager
  math 101
  math 101 section 1

1: fill in info
2: submit
3: add course(joe, math 01)
4: are you open?
5: are you open?
6: add (joe)
7: add (joe)
```

Collaboration Diagram
• A collaboration diagram displays object interactions organized around objects and their links to one another

```
- Registrar
  course form : CourseForm

1: set course info
2: process

1Course : Course
theManager : CurriculumManager

3: add course
4: new course
```

Copyright © 1997 by Rational Software Corporation
Class Diagrams

• A class diagram shows the existence of classes and their relationships in the logical view of a system
• UML modeling elements in class diagrams
  – Classes and their structure and behavior
  – Association, aggregation, dependency, and inheritance relationships
  – Multiplicity and navigation indicators
  – Role names

Classes

• A class is a collection of objects with common structure, common behavior, common relationships and common semantics
• Classes are found by examining the objects in sequence and collaboration diagram
• A class is drawn as a rectangle with three compartments
• Classes should be named using the vocabulary of the domain
  – Naming standards should be created
  – e.g., all classes are singular nouns starting with a capital letter
Classes

- RegistrationForm
- RegistrationManager
- Student
- Professor
- Course
- CourseOffering
- ScheduleAlgorithm

Copyright © 1997 by Rational Software Corporation

Operations

- The behavior of a class is represented by its operations
- Operations may be found by examining interaction diagrams

Copyright © 1997 by Rational Software Corporation
Attributes

- The structure of a class is represented by its attributes
- Attributes may be found by examining class definitions, the problem requirements, and by applying domain knowledge

Each course offering has a number, location and time

Classes

- RegistrationForm
- RegistrationManager
  - addStudent(Course, StudentInfo)
- ScheduleAlgorithm
- Course
  - name
  - numberCredits
  - open()
- Student
  - name
  - major
- Professor
  - name
  - tenureStatus
- CourseOffering
  - location
  - open()
  - addStudent(StudentInfo)
Relationships

• Relationships provide a pathway for communication between objects
• Sequence and/or collaboration diagrams are examined to determine what links between objects need to exist to accomplish the behavior -- if two objects need to “talk” there must be a link between them
• Three types of relationships are:
  – Association
  – Aggregation
  – Dependency

Relationships

• An association is a bi-directional connection between classes
  – Shown as a line connecting the related classes
• An aggregation is a stronger form of relationship where the relationship is between a whole and its parts
  – Shown as a line connecting the related classes with a diamond next to the class representing the whole
• A dependency relationship is a weaker form of relationship showing a relationship between a client and a supplier where the client does not have semantic knowledge of the supplier
  – Shown as a dashed line pointing from the client to the supplier
Finding Relationships

- Relationships are discovered by examining interaction diagrams
  - If two objects must “talk” there must be a pathway for communication

```
3: add student(joe)
```

Relationships

```
RegistrationForm
ScheduleAlgorithm
Course
Student
Professor
CourseOffering
RegistrationManager
```
Multiplicity and Navigation

- Multiplicity defines how many objects participate in a relationship
  - Multiplicity is the number of instances of one class related to ONE instance of the other class
  - For each association and aggregation, there are two multiplicity decisions to make: one for each end of the relationship
- Although associations and aggregations are bi-directional by default, it is often desirable to restrict navigation to one direction
- If navigation is restricted, an arrowhead is added to indicate the direction of the navigation

```
Copyright © 1997 by Rational Software Corporation
```
OOA--Conceptual Modeling

• Most important task is identification of basic objects (concepts) that comprise the problem domain.
  – Identified objects should belong to the problem domain--i.e. no artifacts of the software design (windows, GUIs, etc.)
  – Don’t worry about assignment of responsibilities (methods) to objects at this time.
  – Don’t think in terms of OOD concepts such as inheritance

Conceptual Modeling--Identifying Objects

• Candidate Objects:
  – Shlaer & Mellor (1988):
    • Tangible things--car, sensor, book, bank, …
    • Roles--teacher, teller, student, …
    • Events--interrupt, sale, request, …
    • Interactions--meeting, reservation, etc.
  – Pressman (1997)
    • External entities--other systems, people, …
    • things--reports, orders, …
    • occurrence or events
    • roles
    • organizational units--division, group, team
    • places
    • structures
  – Larman
    • List on pp. 92-93 in text.
Identifying (Conceptual) Objects

• Noun Phrase Analysis
  – Identify all noun phrases in Requirements
  – Retain those that “pass muster” as objects.
  – This approach may not be practical for complex requirements.

• Use Case Analysis
  – Objects may be identified during preparation of sequence diagrams and collaboration diagrams

Conceptual Modeling--Selecting the Proper Objects

Discard candidate classes according to the following criteria (Rumbaugh et al, 1991):

– redundant classes
– irrelevant classes
  • little or nothing to do with problem domain
– vague classes
  • ill-defined or too broad
– attributes
– operations

– Roles
  • An entity may play several roles
  • roles are more appropriately expressed as associations among classes
– Implementation constructs
Choosing Conceptual Classes--An Example (from Rumbaugh et al)

- An Automated Teller Machine (ATM) Network

Modeling Example--Continued

- Requirements:
  Develop the software for a computerized banking network, including both human cashiers and automated teller machines (ATMs) to be shared by a consortium of banks. Each bank provides its own computer to maintain its own accounts and process transactions against them. Cashier stations are owned by individual banks and communicate directly with their own bank’s computers. Human cashiers enter account and transaction data. ATMs communicate with a central computer which clears transactions with the appropriate bank(s). An ATM accepts a cash card, interacts with the user, communicates with the central system to carry out the transaction, dispenses cash, and prints receipts. The system requires appropriate record keeping and security provisions. The system must handle concurrent accesses to the same account correctly. The banks will provide their own software for their own computers. Thus the design of this system will cover only the software for the ATMs and the network. The cost of the shared system will be apportioned to the banks according to the number of customers with cash cards.
ATM Example--Candidate Classes

- Software
- Banking Network
- Cashier
- ATM
- Consortium
- Bank
- Bank Computer
- Account
- Transaction
- Cashier station
- Account data

- Transaction data
- Central computer
- Cash card
- User
- Cash
- Receipt
- System
- Record keeping provision
- Security provision
- Access
- Cost
- Customer

ATM Example--Additional Candidate Classes

- Identified from knowledge of problem domain
  - communication line
  - transaction log
ATM Example--Elimination of Unnecessary Classes

- **Vague**
  - System
  - Security provision
  - Record keeping provision
  - Banking Network

- **Redundant**
  - User

- **Irrelevant**
  - Cost

- **Attribute**
  - Account data
  - Receipt
  - Cash
  - Transaction data

- **Implementation**
  - Transaction log
  - Access
  - Software
  - Communication Line

---

ATM Example--Good Classes

- Account
- ATM
- Bank
- Bank Computer
- Cash Card
- Cashier
- Cashier Station
- Central Computer
- Consortium
- Customer
- Transaction
Conceptual Modeling--Identifying Associations

- Associations typically correspond to stative verbs or verb phrases in the requirements.
  - Part of
  - drives
  - communicates with
  - etc.
- See the list of common associations in Larman (Page 108)

Initial Class Diagram for ATM Example
ATM Class Diagram--A Slight Refinement

ATM Class Diagram--Further Refinement
Conceptual Modeling--Adding Attributes

• Attributes are properties or data values associated with an object
  – name
  – account number
  – account balance
  – etc.

• Attributes normally correspond to nouns followed by possessive phrases in requirements.

Identifying Attributes--General Rules

• Don’t get carried away--Keep it simple
  – Focus on important (essential) attributes
  – Attributes should be *simple*.
    • Don’t model complex domain concepts as attributes.
  – Don’t confuse attributes with associations.
    • Relationships should be represented as associations, not attributes.
ATM Class Diagram with Attributes

Conceptual Modeling--Additional Elements

- Data Dictionary (Glossary)
  - Defines terms used in analysis models
    - class descriptions
    - association descriptions
    - attribute descriptions
    - anything else that needs description or further explanation
  - No standard format
Modeling System Behavior--Sequence Diagrams

- Show time-dependent behavior of system with respect to externally generated events.
- Derived from use cases
- Each diagram typically shows one path through a use case.
  - Normal case(s)
  - Special cases

Behavioral Analysis Example

- Use case for ATM withdrawal transaction
  - “Normal” case:
    ATM prompts user to insert card; User inserts card.
    ATM accepts card and reads information from it.
    ATM requests password; User enters password.
    ATM verifies card info. And password with consortium;
    ATM prompts user for transaction type: withdrawal, deposit, transfer, query.
    User selects withdrawal
    ATM prompts for withdrawal amount. User enters amount.
    ATM verifies that amount is within allowed limit.
    ATM forwards transaction to consortium.
    Consortium passes transaction request to user’s bank.
    Bank processes transaction and passes new account balance to consortium.
    Consortium authorizes cash dispensing by ATM and passes new account balance.
    ATM dispenses cash; User takes cash.
    ATM prompts user for additional transactions; User declines.
    ATM prints receipt and ejects card; User takes card.
Behavioral Modeling--Contracts

- Describe effect of operations on system state
- Define behavior in terms of pre-conditions and post-conditions
- Can be useful at many different levels of analysis and design
- During analysis we are only interested in system behavior--what it does, not how.
Pre-conditions and Post-conditions

• Pre-conditions:
  – Assumptions about system state at beginning of the operation
  – Things that must be true or defined for the operation to be valid.

• Post-conditions:
  – Must be true of system state AFTER operation is completed.
    • objects created or destroyed
    • associations formed or broken
    • attributes modified