Basic Structural Modeling
Part I

Classes
Classes

- Description of a set of objects sharing the same attributes, operations and semantics
- Abstraction of the things that are part of the application domain vocabulary
- Can represent software, hardware or conceptual things
Terms and Concepts

• Names
  – nouns drawn from the application domain vocabulary, beginning with a capital letter
  – short but long enough to carry a meaning
  – can include the path of the package(s) it belongs to

- Shape
- java.awt.Rectangle

simple names

- Temperature Sensor

path name
Terms and Concepts

• Attributes
  – named properties of a class representing the state of the objects
  – extension of classical data structures
  – can include type and default values

<table>
<thead>
<tr>
<th>Customer</th>
<th>Wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>address</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Terms and Concepts

• Operations
  – implementation of a behavior or service
  – messages that can be understood by the class
  – verb or verb phrase describing the intended behavior
  – can define the interface of the functions
Terms and Concepts

- Organizing Attributes and Operations
  - some attributes and operations can be omitted
  - they can be organized using stereotypes

```plaintext
FraudAgent
<<constructor>>
new()
new(p : Policy)
<<process>>
process(o : Order)
...
<<query>>
isSuspect(o : Order)
isFraudulent(o : Order)
<<helper>>
validateOrder(o : Order)
```
Terms and Concepts

- Responsibilities
  - defines the requirements of classes
  - expressed as form-free text
  - can be used in early stages for abstraction

<table>
<thead>
<tr>
<th>FraudAgent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- determine the risk of a customer order</td>
</tr>
<tr>
<td>-- handle customer-specific criteria for fraud</td>
</tr>
</tbody>
</table>
Common Modeling Techniques

• Modeling the vocabulary of the system
  – identify things that users use to describe the problem
  – identify things that implementers use to describe the solution
  – for each class, identify a set of responsibilities
  – provide the attributes and operations needed to carry out these responsibilities
  – aggregate related clusters into packages
Common Modeling Techniques

• Modeling the distribution of responsibilities in a system
  – identify a set of classes that work together to carry out some behavior
  – identify a set of responsibilities for these classes
  – aggregate classes that have too few responsibilities
  – split classes that have too many responsibilities
  – consider the workload involved by each responsibility and distribute the load evenly
Hints and Tips

• When you model, define classes that
  – are an abstraction of something drawn either from the vocabulary of the problem or the solution
  – embody a small, well-defined set of responsibilities
  – are understandable, simple, extensible and adaptable
Hints and Tips

• When you draw class diagrams
  – show only those properties of the class that important in the current context
  – organize long lists of attributes and operations using stereotypes
  – show related classes in the same diagrams
Part II

Relationships
Relationships

• Very few classes stand alone
• Classes collaborate to respond to service requests
• We must model the interactions between objects and classes
  – Dependencies: uses relationships
  – Generalizations: subclass/superclass relationships
  – Associations: structural relationships

Copyright Joey Paquet, 2000
Relationships

Window
- open()
- close()
- move()
- display()
- handleEvent()

Event

ConsoleWindow

DialogBox

Control

dependency

generalization

association
Terms and Concepts

• Dependencies
  – a change in the state of an object may affect the behavior of the depending object
  – a class uses another in its behavior
  – drawn as a dashed directed line
  – most often used to show that one of the operations uses another class as an argument
  – dependencies can be used at various levels of abstraction
Terms and Concepts

• Generalizations
  – relationship between a general thing (parent) and a more specific thing (child) of the same kind
  – the child inherits the attributes and operations of the parent class
  – operations can be redefined in the child and override the operations of the parent (polymorphism)
  – drawn as a directed line with open arrowhead
Terms and Concepts

Diagram:
- **Shape**
  - origin
  - move()
  - resize()
  - display()

  - **super class**

- **Rectangle**
  - ...
  - ...

- **Circle**
  - radius
  - ...

- **Polygon**
  - points : List
  - display()
Terms and Concepts

- **Associations**
  - used to show structural relationships where none of the classes is part of the other
  - drawn as a solid line connecting classes

- **Name of an association**
  - describes the nature of the relationship
  - can include the direction of the relationship
  - drawn as a tag on the middle of the association
Terms and Concepts

• Roles of an association
  – identifies the role that each class plays in the relationship
  – drawn as a tag on the near end of the association
  – same class can play different roles in different contexts
Terms and Concepts

- Defining multiplicity in an association
  - shows how many objects of the same type are involved in the association
  - similar to multiplicity defined in entity-relationship diagrams
Terms and Concepts

• Aggregation
  – an association where one class is part of the other (*has-a* relationship)
  – drawn as a solid line with a diamond on the whole side of the relation

![Diagram of Aggregation](image.png)
Common Modeling Techniques

• Modeling simple dependencies
  – e.g. dependency between a class and a class being a parameter in one of its operations
  – can be omitted if the signature of the operations is provided in the class description
Common Modeling Techniques

• Modeling single inheritance
  – some classes that have common behavior or structure can be have a common superclass
  – given a set of classes, look for responsibilities, attributes and operations common to two or more classes
  – elevate these elements to a more general class without introducing too many levels
  – specify that the children inherit from the parent using a generalization relationship from children to parent
  – multiple inheritance is allowed, cyclic inheritance is not
Common Modeling Techniques

• Modeling structural relationships
  – for each pair of classes, if you need to navigate between the two, define an association between them (data-driven associations)
  – for each pair of classes, if an interaction is needed between the two (other than parameters to an operation) specify an association between the two (behavior-driven associations)
  – for each of these associations, define their multiplicity
  – if one of the classes is a part of the other, make it an aggregation
Common Modeling Techniques
Hints and Tips

• When you model
  – use dependencies only when the relationship is not structural
  – use generalizations only when you have a is-kind-of relationship
  – cyclical generalization is not allowed
  – inheritance trees should not be too deep nor too wide
Hints and Tips

• When you draw
  – use rectilinear or oblique lines consistently
  – avoid lines that cross
  – show only relationships that are necessary to the understanding of the diagram in the current context
Part III

Common Mechanisms
Common Mechanisms

- Some important facts about a design might not be expressible in the basic UML notation
  - notes
  - version number of components
  - constraints on dependencies
Terms and Concepts

• Notes
  – a graphical symbol used to attach comments or constraints to an element
  – drawn as a rectangle with a dog-eared corner containing a textual or graphical comment
  – it does not carry any semantic impact

• Tagged values
  – extension of the properties of a UML element
  – drawn as a string enclosed in \{brackets\}
Terms and Concepts

• Stereotypes
  – extension of the vocabulary of the UML
  – used to create new kinds of building blocks
  – drawn as a name enclosed in <<guillemets>>
  – can be represented as a new kind of icon

• Constraint
  – extension of the semantics of a UML element
  – drawn as a string enclosed in {brackets}
  – can be put in a note
Hints and Tips

• Use notes only for facts that can’t be expressed using the UML
• Use notes to specify work in progress
• Don’t use too many or large notes
• Standardize the use of stereotypes, tagged values and constraints
• Don’t use too many graphical stereotypes
• Use only general and simple additions
Part IV

Diagrams
Diagrams

• Simplification of reality
• Using a limited set of general-use building blocks
• Different types of diagrams are used to represent the system in different perspectives
• Good diagrams make the system more understandable
• Choosing the right diagrams forces you to ask the right questions and illuminate their implications
Terms and Concepts

• Structural diagrams
  – Used to visualize, specify, construct and document the static aspects of the system
  – Represents the “skeleton” of the system
Terms and Concepts

• Structural diagrams
  – Class diagram
    • a set of classes interfaces, collaborations and their relationships
    • used to render the static design view of a system
  – Object diagram
    • same as class diagram, but for instances
    • “snapshot” of the relations between objects in a hypothetical situation
Terms and Concepts

• Structural diagrams
  – Component diagrams
    • shows a set of software components and their relationships
    • used to render a static implementation view of the system
  – Deployment diagrams
    • shows a set of nodes and their relations
    • used to render the static deployment view of the system
    • more abstract version of component diagrams
Terms and Concepts

• Behavioral diagrams
  – Used to visualize, specify, construct and document the dynamic aspects of the system
  – Represents the behavior of the system
Terms and Concepts

• Behavioral diagrams
  – Use case diagrams
    • shows a set of use cases and actors and their relationships
    • shows the different views that are possible on the system
    • defines subsets of classes (and their interactions) that are needed to achieve a certain goal of the system
Terms and Concepts

• Behavioral diagrams
  – Sequence diagrams
    • defines the time ordering of messages exchanged between a set of objects
    • shows a set of objects and the messages sent and received between these objects towards the realization of a certain service
Terms and Concepts

• Behavioral diagrams
  – Collaboration diagrams
    • defines all the dynamic interactions between a set of objects
    • shows a set of objects, links among those objects, and messages sent and received by those objects in the general case (not related to a specific service)
  – Sequence and collaboration diagrams carry the same information but not in the same context
Terms and Concepts

• Behavioral diagrams
  – Statechart diagram
    • defines a behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events
    • emphasizes the event-ordered behavior of an object
    • used to model the behavior of an interface, class or collaboration between classes
Terms and Concepts

• Behavioral diagrams
  – Activity diagram
    • defines the flow of control among objects
    • shows the flow from activity to activity within the system
    • shows a set of activities, the sequential or branching flow between activities and objects that act or are acted upon in the activity
Common Modeling Techniques

• Modeling different views of a system
  – views are the different dimensions from which the system can be represented
  – different views expose different aspects of the problem or see the same aspect in a different context
  – must find the right set of views
  – must focus on different views separately, and then find a compromise for common parts
Common Modeling Techniques

• Modeling different views of a system
  – decide which views best express the architecture and expose all problems
  – for each of these views, chose which kind of diagrams you will use to capture its details
  – decide which of these will be part of the documentation
  – unused diagrams should not be thrown away
Common Modeling Techniques

• Modeling different levels of abstraction
  – different people involved in the development have different needs
  – the different kinds of diagrams propose a different view on the system
  – some people need abstract information, others need exact information
  – several versions of the same diagram, at various levels of abstraction, might be needed
Common Modeling Techniques

- Modeling different levels of abstraction
  - consider the different needs of the readers
  - create different abstraction levels for diagrams that are read by different readers
  - hide or reveal building blocks and relationships
  - hide or expand messages and transitions that are essential to understanding
  - reveal only adornments that are essential to understanding (e.g. classifying stereotypes)
Hints and Tips

• When you design a diagram
  – diagrams are a tool, they do not need to be cute
  – not all diagrams are meant to be preserved
  – find an goal and an audience for each diagram
    and show only the information needed to explain the solution to the intended audience
  – give meaningful names to all diagrams
  – group diagrams into packages
Hints and Tips

• A well structured diagram
  – is focused on communicating one aspect of the system
  – contains only those elements that are essential to understanding that aspect
  – provides details consistent with its level of abstraction
Hints and Tips

• When you draw a diagram
  – give it a name that communicates its purpose
  – lay out elements to minimize lines that cross
  – lay out related elements close to one another
  – use colors and notes to draw attention on important features
Part V

Class Diagrams
Class Diagrams

• Most common diagram used to model object-oriented systems
• Shows a set of classes, interfaces, and collaborations and their relationships
• Models the static design view of the system
• Involves modeling the vocabulary of the system requirements specifications
Terms and Concepts

• Class diagrams commonly contain
  – classes
  – interfaces
  – collaborations
  – relationships
  – notes and constraints
  – packages or subsystems
Terms and Concepts

• Common uses
  – model the static design view to support the functional requirements of the system
    • **model the vocabulary** of the system: first define classes and their responsibilities, and then refine towards attributes, operations
    • **model simple collaborations**: define the interactions between classes towards the definition of a common behavior
    • **model a database schema**: data elements are attributes, tables are classes, and relationships are…relationships
Common Modeling Techniques

• Modeling simple collaborations
  – identify the mechanism to model.
  – for each mechanism, identify the classes, interfaces, and other collaborations that participate in this collaboration; as well as their relationships
  – use scenarios to walk through these things
  – populate these elements with their required contents (attributes and operations) first starting with responsibilities
Example

```
Motor

move(Direction, Speed)
stop()
resetCounter()
status()

SteeringMotor 1

PropulsionMotor 1

PathAgent
Responsibilities
-- seek path
-- avoid obstacles

1

Driver

CollisionSensor *
```
Common Modeling Techniques

• Modeling a logical database schema
  – identify the classes whose state must transcend the lifetime of their application
  – create a class diagram for these and mark them as <<persistent>> with a tagged value
  – define their attributes and associations (and their cardinalities)
  – define data-specific operations on these classes
Example of Database Schema
Hints and Tips

• A well-structured class diagram
  – is focused on communicating only one aspect of the system’s static design
  – provides only the details that are consistent with its associated level of abstraction
  – is not minimalist
Hints and Tips

• When you draw a class diagram
  – give it a name that communicates its purpose
  – minimize line crossing
  – use colors and notes to emphasize important aspects
  – lay out related elements close to one another