Chapter 2

Software Modeling Language

© Pearson Education Limited 2005

Topics

- Structured modeling language
  - Data flow modeling
  - Entity-relationship modeling

- Object-oriented modeling language
  - Class diagrams
  - Use case diagrams
  - Interaction diagrams
  - Statechart diagrams
  - Activity diagrams
  - Implementation diagrams

© Pearson Education 2005
The theme

- Software engineering is about modeling
  - the outcome of software engineering – a program – is an executable model
- Software modeling requires a language as a means of:
  - communicating between stakeholders
  - expressing development processes and artifacts at multiple levels of abstraction
- The Unified Modeling Language (UML)
- UML profiles to cater for specific domains and technologies
- Model consists of one or more diagrams and any additional information stored in the project repository

Structured modeling language

- Structured programming
  - without goto statements,
  - loops and if statements as the main control constructs,
  - top-down approach to program design
- Structured programming → structured modeling
  (structured analysis and design)
  - expresses the monolithic and procedural character of Cobol-style systems of the past
  - functional decomposition - top-down function-oriented approach to software development
  - visualization techniques
    - Data Flow Diagrams (DFDs)
    - Entity-Relationship Diagrams (ERDs)
    - structure charts
Data Flow Diagrams (DFDs)

- One of the most popular modeling technique in the history of SE
- Mismatch with the object-oriented approach
- The cornerstone of DFDs is functional decomposition

**Context Diagram**

- Consists of:
  - one process only
  - a number of external entities
  - in- and out-flows between the process and external entities
- Determines the place of the system with regard to its environment
Level “0” diagram

- Called also the overview diagram

Level “1” diagram

- Flow balancing
- Data store
Entity-Relationship (ER) modeling

- **A data modeling technique**
- **Entity-Relationship Diagrams** (ERDs) define just three modeling elements – entities, relationships, and attributes
- An entity is a conceptual data structure, which represents a business fact or rule and which can be distinctly identified (usually)
- A relationship represents an association between entity instances from different entity sets and, in some important cases, from a single entity set
- An attribute is a data-value pair
  - single-valued
  - multi-valued attributes and composite attributes not normally supported

### ER crow’s foot notation

- Attributes in entities
  - name, identifier indication, type, mandatory indication
- Multiplicity of relationships
  - mandatory vs optional participation

**Movie**
- movie_code: N5, M
- movie_title: VA30
- director: VA20

**Actor**
- actor_code: N5, M
- actor_name: VA25

**Listed as**
- position: N2
Object-oriented modeling language

- The Unified Modeling Language (UML) “…is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.” (UML, 2003b, p.1-1)
- In the UML, visual modeling is an arrangement of so-called classifiers
  - A classifier is a model element that describes the system’s behavior or structure and that normally has a visual representation
    - Examples of classifiers include class, actor, use case, relationship
- In a working system, classifiers manifest as objects
  - An object is a piece of software that has
    - state - defined by its attribute values
    - behavior - defined by services (operations) that an object can perform
    - identity – to differentiate between objects (even with the same state and behavior)

Six kinds of diagrams

- state structure,
- use case,
- interaction,
- statechart,
- activity, and
- implementation diagrams.
Class diagrams

- **Class diagram:**
  - expresses static structures of models, called also state models
  - visualizes classes (and interfaces), their internal structure, and their relationships to other classes
- "**a class is the descriptor for a set of objects with similar structure, behavior, and relationships.**" (UML, 2003b, p.3-35)
- **Attribute** is a structural (typed) feature of a class
  - attribute ➔ data member, member variable, instance variable, field
- **Operation** is a behavioral feature of a class
  - operation ➔ member function, method

Modeling elements of class diagram

```
ClassB
attribute1
attribute2
operation1()
operation2()
0..n association1 0..1 ClassA
n association2 1..n
1..n aggregation
unidirectional association
1..n
ClassD
generalization
0
ClassC
```
Class diagram as static structure diagram

```
class actor
  actor_code : double
  actor_name : String

class listed_as
  position : double

class movie
  movie_code : double
  movie_title : String
  director : String

movie(movie_code : double, title : String, director : String)
addListedAs(l : ListedAs) : void
removeListedAs(l : ListedAs) : void
getMovieTitle() : String
setMovieTitle(property1 : String) : void
getMovieCode() : double
setMovieCode(property1 : double) : void
getDirector() : String
setDirector(property1 : String) : void
equals(o : Object) : boolean
```

Class design with state and behavior features

```
Movie
movieTitle : String
movieCode : double
director : String

Movie(movieCode : double, title : String, director : String)
addListedAs(l : ListedAs) : void
removeListedAs(l : ListedAs) : void
getMovieTitle() : String
setMovieTitle(property1 : String) : void
getMovieCode() : double
setMovieCode(property1 : double) : void
getDirector() : String
setDirector(property1 : String) : void
equals(o : Object) : boolean
```
**Design class diagram**

### MovieSearcher

- MOVIE ONLY: int = 1
- ACTOR ONLY: int = 2
- MOVIE AND ACTOR: int = 3

- MovieSearcher
- getMovies
- displayMoviesByDirector
- displayMoviesByLeadingActors
- displayMoviesByLeadingActorsByStoredProcedure
- displayMoviesByActor
- displayMoviesByActorCode
- displayMoviesByTitle
- displayActorsByMovieTitle
- displayActorsByMovieCode
- retrieveAll
- display

### Connection

- Connection
- readAll
- query
- closeResult
- getLeadingActorsByQuery
- getLeadingActorsByStoredProcedure
- searchMoviesByStoredProcedure

### ListedAs

- position: double

- ListedAs
  - getPosition
  - setPosition
  - getMovie
  - setMovie
  - getActor
  - setActor

### Actor and Movie

### Collection (from util)

### Movie

- movieTitle: String
- movieCode: double
- director: String
Use case diagrams

- The main analysis-level behavior modeling technique in UML
- The power of use case diagrams does not rest in graphical diagrams. The real power of use case diagrams is in textual specifications of use cases stored in the repository.
- Use case represents a major piece of system functionality.
- Actor is a role that somebody or something plays with regard to a use case.
  - Actor communicates with a use case (via «communicate» relationship) and expects from it some feedback – a value or observable result.

Modeling elements of use case diagram

- Actor 1 communicates with Use Case 1 and Use Case 3 via «communicate» relationship.
- Actor 2 communicates with Use Case 2 via «communicate» relationship.
- Use Case 2 is associated with Use Case 4 via «include» relationship.
- Use Case 2 is generalized into Use Case 3.
- Use Case 2 is extended by Use Case 5.
Sequence diagrams

- The first kind of interaction diagrams
- The second are collaboration diagrams
- Interaction diagrams are the main design-level behavior modeling technique in UML
- Sequence diagram is a graphical visualization of sequences of messages between objects
  - placing messages one under another shows this
  - optional numbering of messages also indicates the sequence.
- Object receiving a message activates the relevant method.
- The time when the flow of control is focused in an object is called activation
  - shown as narrow rectangle on object lifeline.
Collaboration (communication) diagram

2.1. query(sql : String)

searcher : MovieSearcher

1. Connection()
2. readAll()

conn : Connection

2.1.1. createStatement()
2.1.2. Movie(movieCode : double, title : String, director : String)
2.1.3. Actor(code : double, name : String)

: Connection

2.1.1.1. executeQuery(s : String)
2.1.4. ListedAs(m : Movie, a : Actor, position : double)

: Statement

: ListedAs

Statechart diagrams

- Not specific to object-oriented modeling
- Capture **states** of an object and **actions** that lead to state transitions on that object
- Drawn for each class, which has interesting state changes worthy of modeling
- **State** of an object (class instance) changes when the values of some of its attributes change
- **States** have durations – they correspond to intervals of time between two **transitions**

event-signature [guard-condition] /action-expression
**Statechart diagram**

- **Movie Details**
  - Known
  - Purchased for Screening

- **Screening**
  - Schedule for Screening
  - Withdrawn from Screens

- **Schedule for Screening**
  - Withdraw

- **Withdrawn from Screens**
  - screen it [movie arrived]

- **Activity diagrams**

  - **Activity diagram** is a state machine that represents a computation, i.e. the performance of actions, and such that the transitions are triggered by the completion of the actions.

  - Typically, an activity diagram is attached to the implementation of an operation or a use case.

  - **Action states** are computations that should not be interrupted by external events or have any outgoing transitions based on explicit events.

  - **Outgoing transitions** from an action state are the result of completing the activity of that state.
Activity diagram

Implementation diagrams

- Models for physical implementation of the system
- Show system components, their structure and dependencies and how they are deployed on computer nodes
- Two kinds of diagrams:
  - component diagrams
  - deployment diagrams
- Component diagrams show structure of components, including their interface and implementation dependencies
- Deployment diagrams show the runtime deployment of the system on computer nodes
Component diagrams

- “A component diagram shows the dependencies among software components, including the classifiers that specify them (for example, implementation classes) and the artifacts that implement them; such as, source code files, binary code files, executable files, scripts.” (UML, 2003b, p.3-169)
- “A component represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.” (UML, 2003b, p.3-174)
- Component implicitly exposes a set of interfaces, which represent services provided by elements residing on the component
- Components may be connected to other components by physical containment (direct nesting of a component in its enclosing component) → «reside» relationship or «implement» relationship
- Packages can be used in a component diagram to illustrate the grouping of the components
The Unified Modeling Language (UML) is the standard modeling language for modern object-oriented software systems. The language for structured modeling includes Data Flow Diagrams (DFDs), Entity-Relationship Diagrams (ERDs), and structure charts. The UML range of diagrams includes class diagrams, use case diagrams, interaction diagrams, statechart diagrams, activity diagrams, and implementation diagrams. Object-oriented UML modeling is centered on class diagrams but is driven by use case diagrams. Interaction diagrams are the main design-level behavior modeling technique in UML. Statechart diagrams capture states of objects and actions that lead to state transitions on the objects. Activity diagram is a state machine that represents a computation. Implementation diagrams (component diagrams and deployment diagrams) are models for physical implementation of the system.