Patterns in Software Engineering

Lecturer: Raman Ramsin

Lecture 1

Earlier Patterns

Software Patterns

- Software Patterns support reuse of software architecture and design.
  - Patterns capture the static and dynamic structures and collaborations of successful solutions to problems that arise when building applications in a particular domain.
- Patterns represent solutions to problems that arise when developing software within a particular context.
  - i.e., “Pattern == problem/solution pair in a context”

Patterns: A Chronological Perspective

- 1979: Christopher Alexander's "Timeless Way of Building"
  - Alexander studied ways to improve the process of designing buildings and urban areas.
- 1987: Cunningham and Beck use Alexander's ideas to develop a small pattern language for Smalltalk.
- 1990: The Gang of Four (Gamma, Helm, Johnson and Vlissides) begin work compiling a catalog of design patterns.
- 1991: Bruce Anderson gives first Patterns Workshop at OOPSLA.
- 1992: Peter Coad introduces his OO Patterns.
- 1993: Kent Beck and Grady Booch sponsor the first meeting of what is now known as the Hillside Group.
- 1994: First Pattern Languages of Programs (PLoP) conference.
- 1995: The Gang of Four (GoF) publish the *Design Patterns* book.

Software Design Patterns

- "A *design pattern* names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design."

- Design Patterns capture the static and dynamic structure and collaboration among key participants in software designs.
  - They are particularly useful for articulating how and why to resolve non-functional forces.
  - Patterns facilitate reuse of successful software architectures and designs.

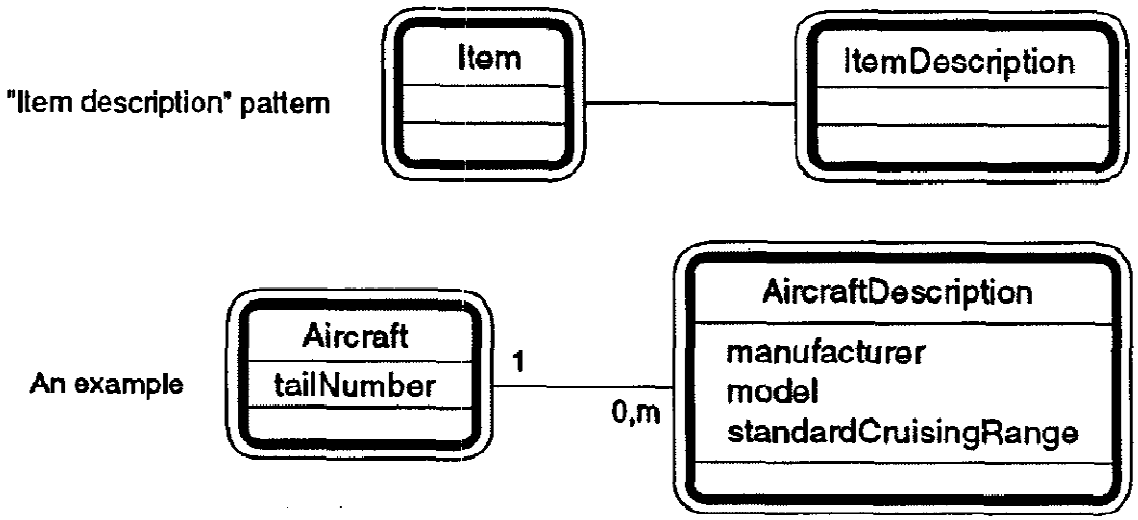
Coad's OO Patterns

- Seven basic Patterns:
  1. Item Description
  2. Time Association
  3. Event Logging
  4. Roles Played
  5. State over a Collection
  6. Behavior over a Collection
  7. Broadcast

Pattern 1: Item Description

- The item description pattern consists of an "item" object (i.e., an object of the class "item") and an "item description" object.
- An "item description" object has attribute *values* which may apply to more than one "item" object; an "item" object has its own individual assignment of attribute values.
- Use this pattern when some attribute *values* may apply to more than one object in a class.

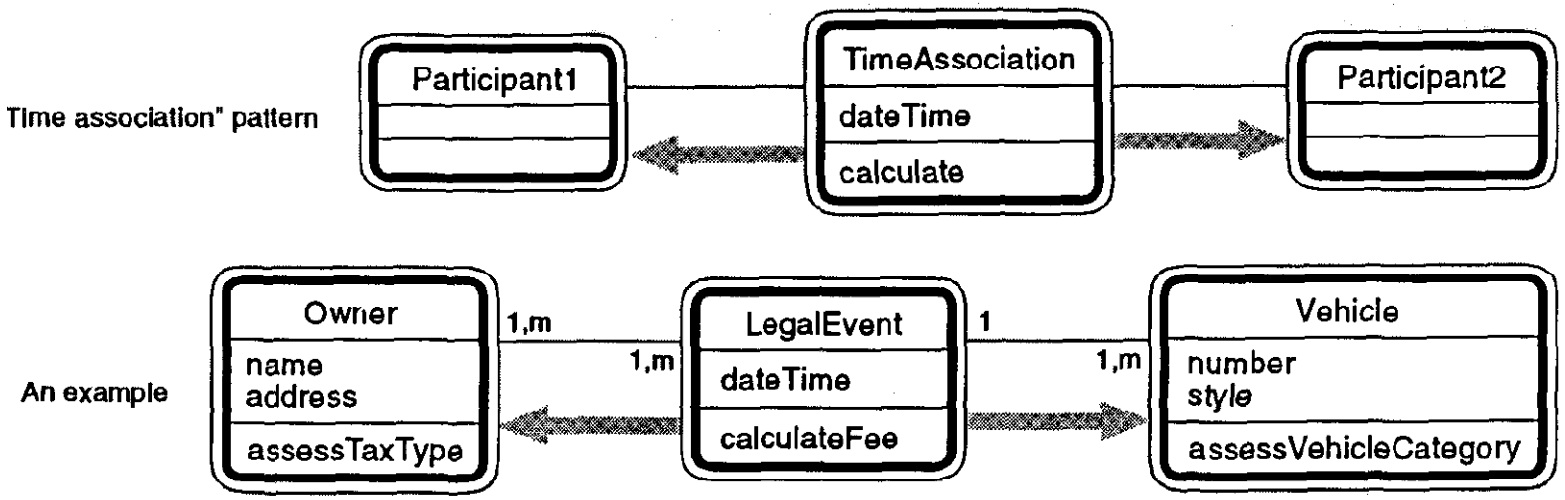
Item Description



Pattern 2: Time Association

- If one needs to express attributes or services regarding an association between two objects, then an object from "time association" is needed.
- A "time association" object often sends messages to its participating objects to get values or get a sub-calculation done on its behalf.
- Note that the association connection:
  - captures the association for future queries about these objects.
  - captures (for the sender) "to whom to send a message."
- Use this pattern whenever the system is responsible to know an association between two or more objects and to know or do something about that association.

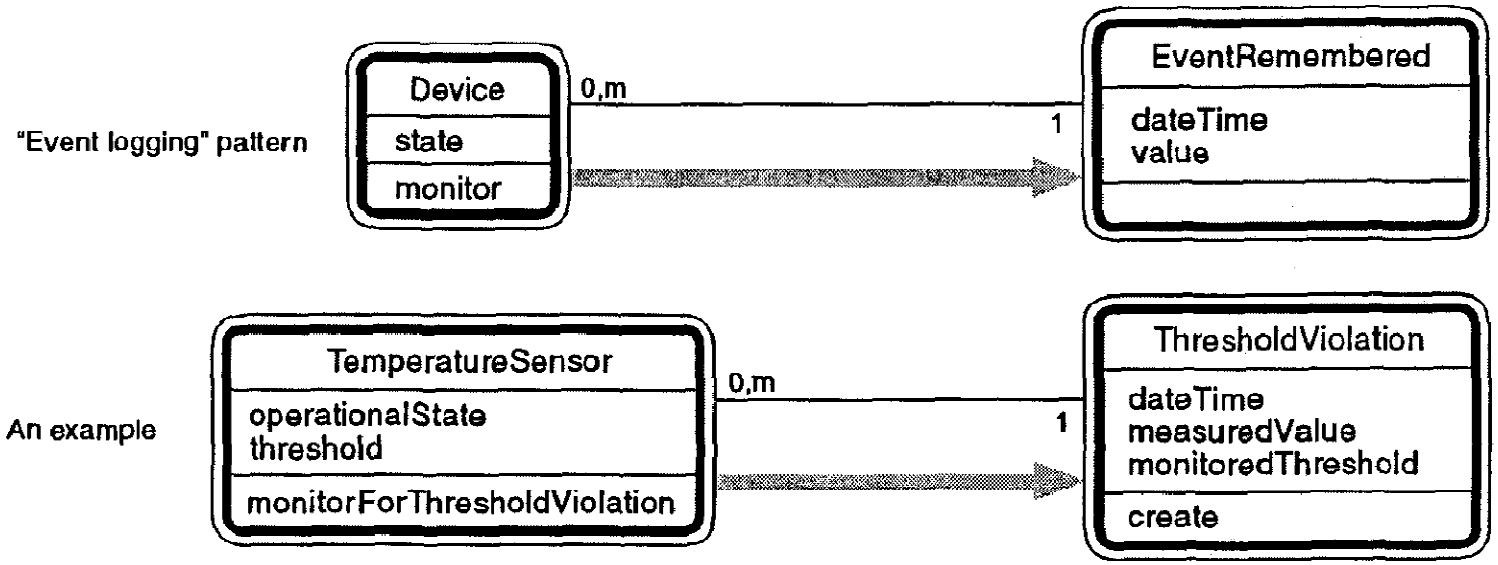
Time Association



Pattern 3: Event Logging

- A "device" object monitors an external device; the object
  - is responsible for detecting that an event has occurred;
  - is responsible for initiating a response to the event.
- Part of the response may be to log the event's occurrence; when this is the case, a "device" object sends the message "create" to the "event remembered" class to create an object with historical values.
- A "device" object may know about some number of "event remembered" objects; an "event remembered" object must know about a corresponding "device" object.
- Use whenever an event is detected, and you need to log its occurrence to support after-the-fact analysis or to meet legal requirements.

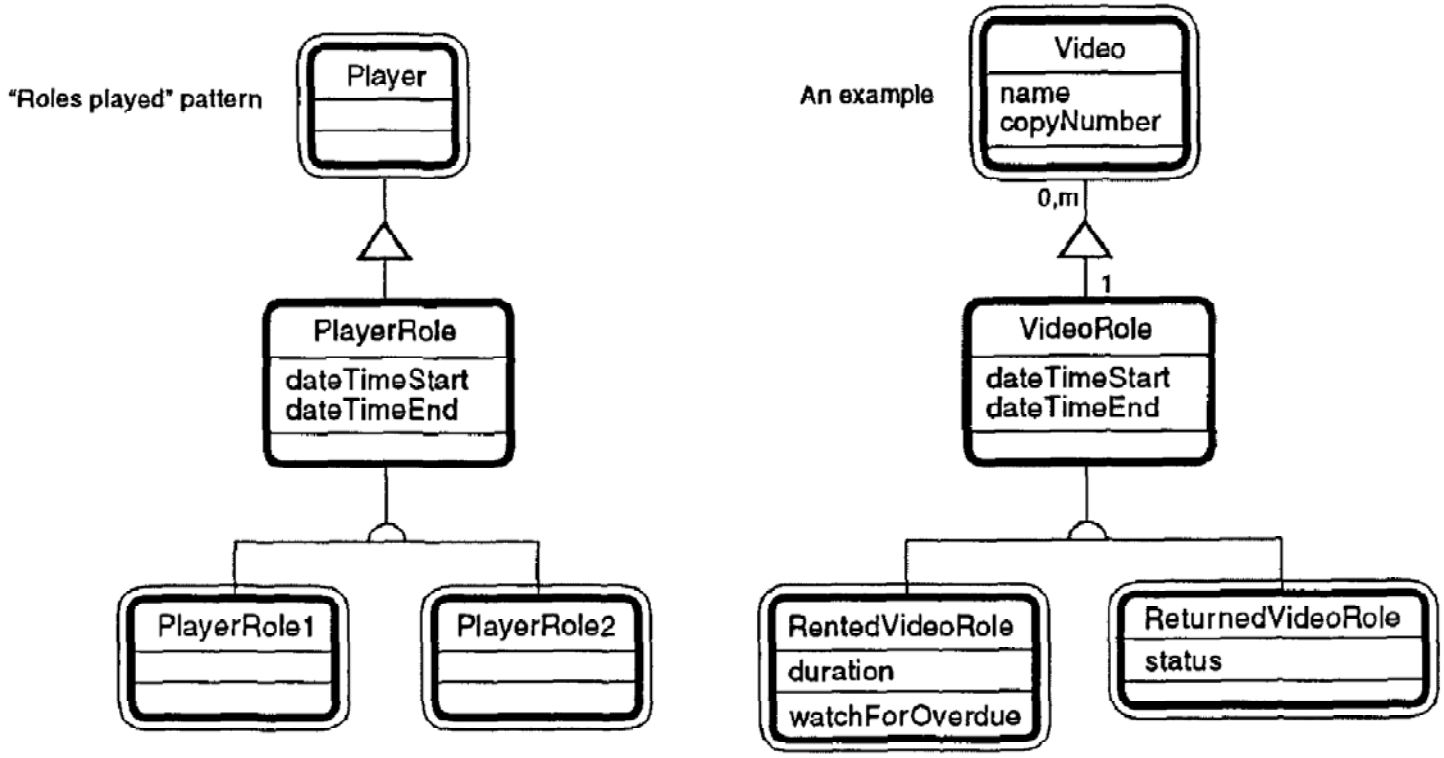
Event Logging



Pattern 4: Roles Played

- A "player" object has attribute values and services that apply over time. A player object is always a player object.
- At times, a player object "wears different hats," playing one or more roles.
- Often, starting and ending times are common to all such roles.
- Use this pattern:
  - whenever you have a player object which remains the same old player object, but has different attributes and services, depending on the "hats" the player may wear.
  - to model large numbers of roles, combinations of roles, and changes in roles; this approach is more concise and flexible than attempting to use multiple inheritance in this situation.

Roles Played

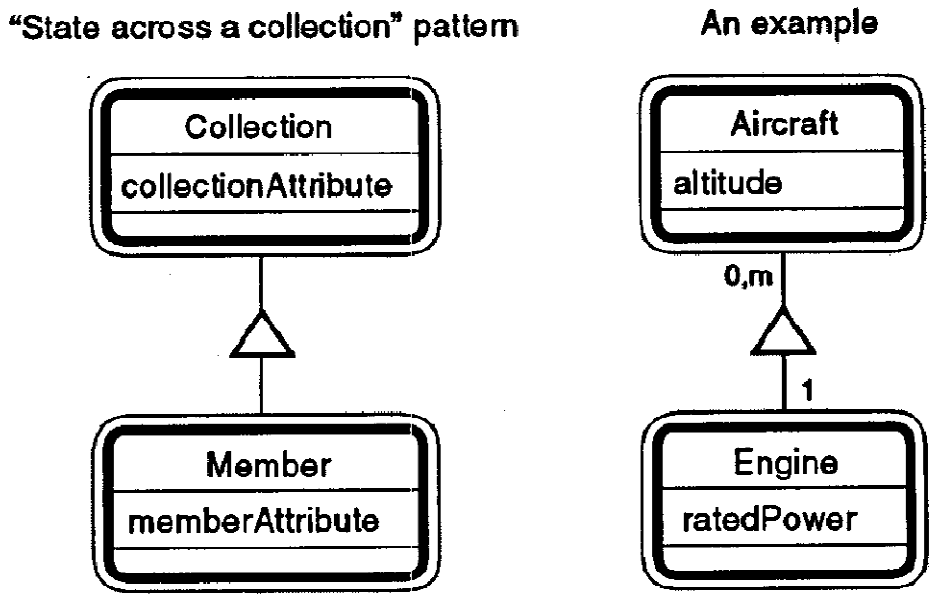


Pattern 5: State over a Collection

- A "collection" object knows its state; this state applies to the collection and may also apply to its parts, and each "member" object has its own state, too.

- Use this pattern whenever there is whole-part in a business domain or implementation domain, and one or more attributes apply to the whole (the collection).

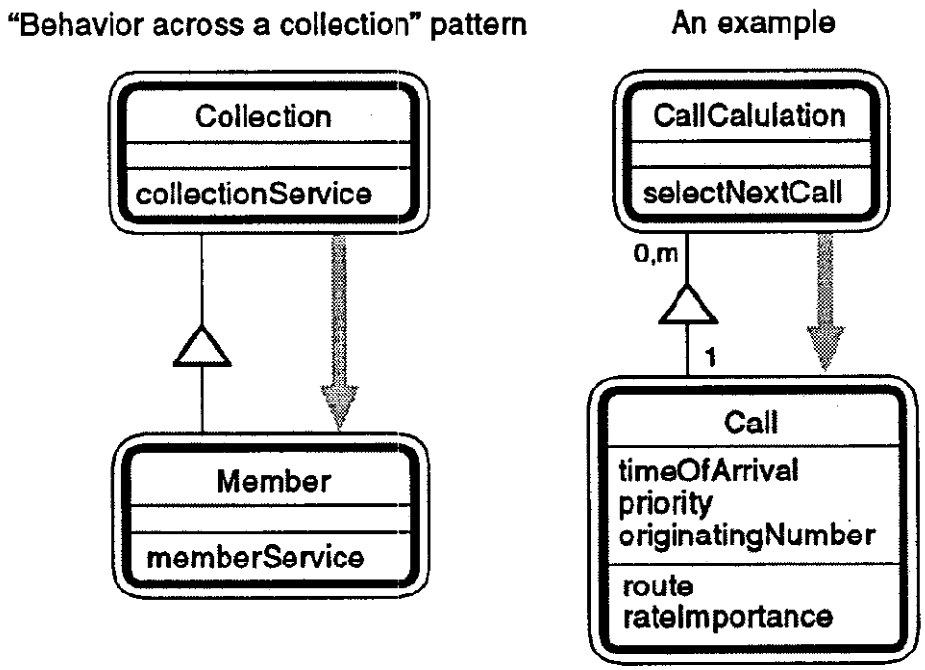
State over a Collection



Pattern 6: Behavior over a Collection

- A "collection" object has behavior that applies across an entire collection of its "member" objects.
- Each "member" object performs actions, knowing (by means of its attributes) how to perform, without needed coordination with other "member" objects.
- Use this pattern whenever there is whole-part in a domain, and a behavior (i.e., one or more services) applies across the whole collection.
- Caution: make the member objects do as much as they can with what they know; only put behavior that really applies across the collection up in the "collection" object.

Behavior over a Collection

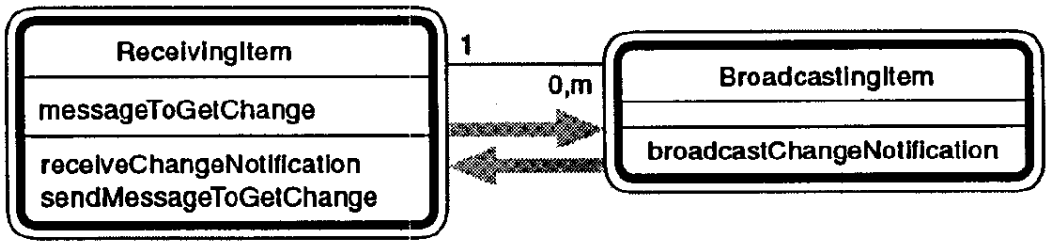


Pattern 7: Broadcast

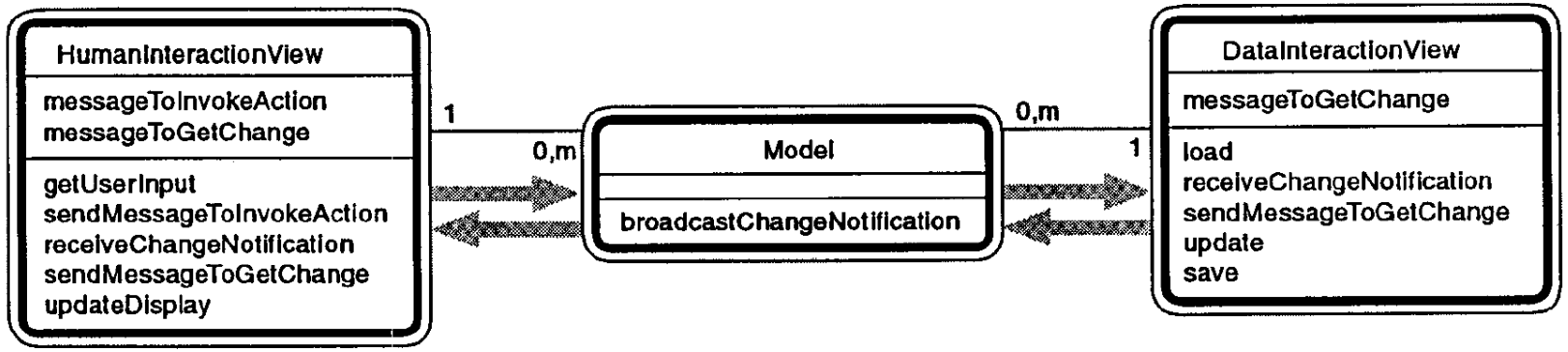
- This pattern is used to communicate complex changes between one major section of an OOA/OOD model with another major section.
  1. Whenever it changes, a "broadcasting item" object broadcasts a change notification to the "receiving item" objects that it knows about.
  2. A notified "receiving item" object then sends a message to the "broadcasting item" to get the change.
  3. Once it gets the change, a "receiving item" object takes whatever action is necessary in light of the change.
- Use this pattern to establish interactions between major OOA/OOD parts in a way that the two sections stay cleanly separated.
- Use this pattern to separate business domain classes from human-interaction and data-management classes.

Broadcast

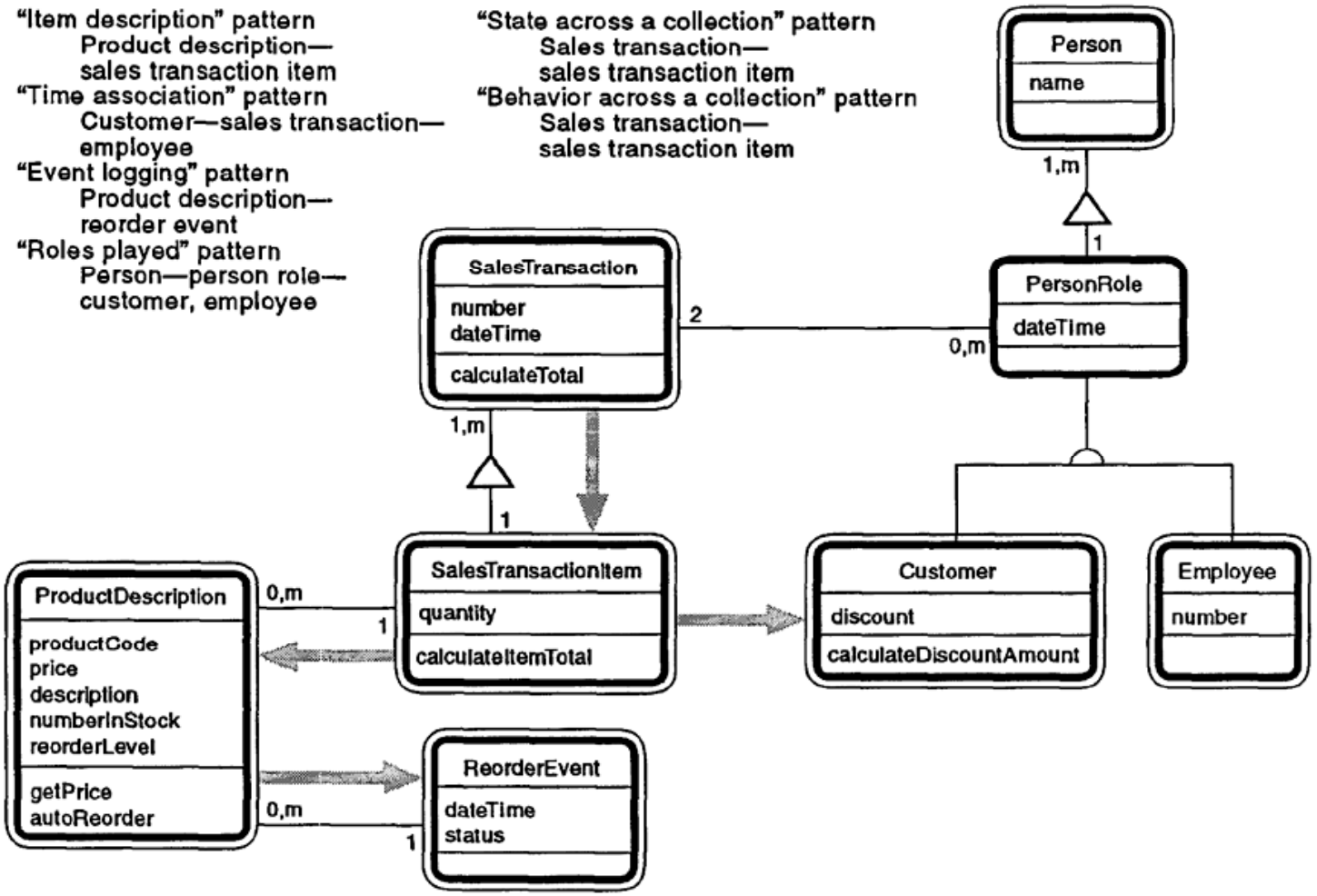
"Broadcast" pattern



An example



Coad Patterns: Example Model



Reference

- Coad, P., Object-oriented patterns, *Communications of the ACM 33*, 9 (September), 152-159, 1992.