Analysis, Design, and Software Architecture (BDSA)

Paolo Tell

Object Oriented Analysis

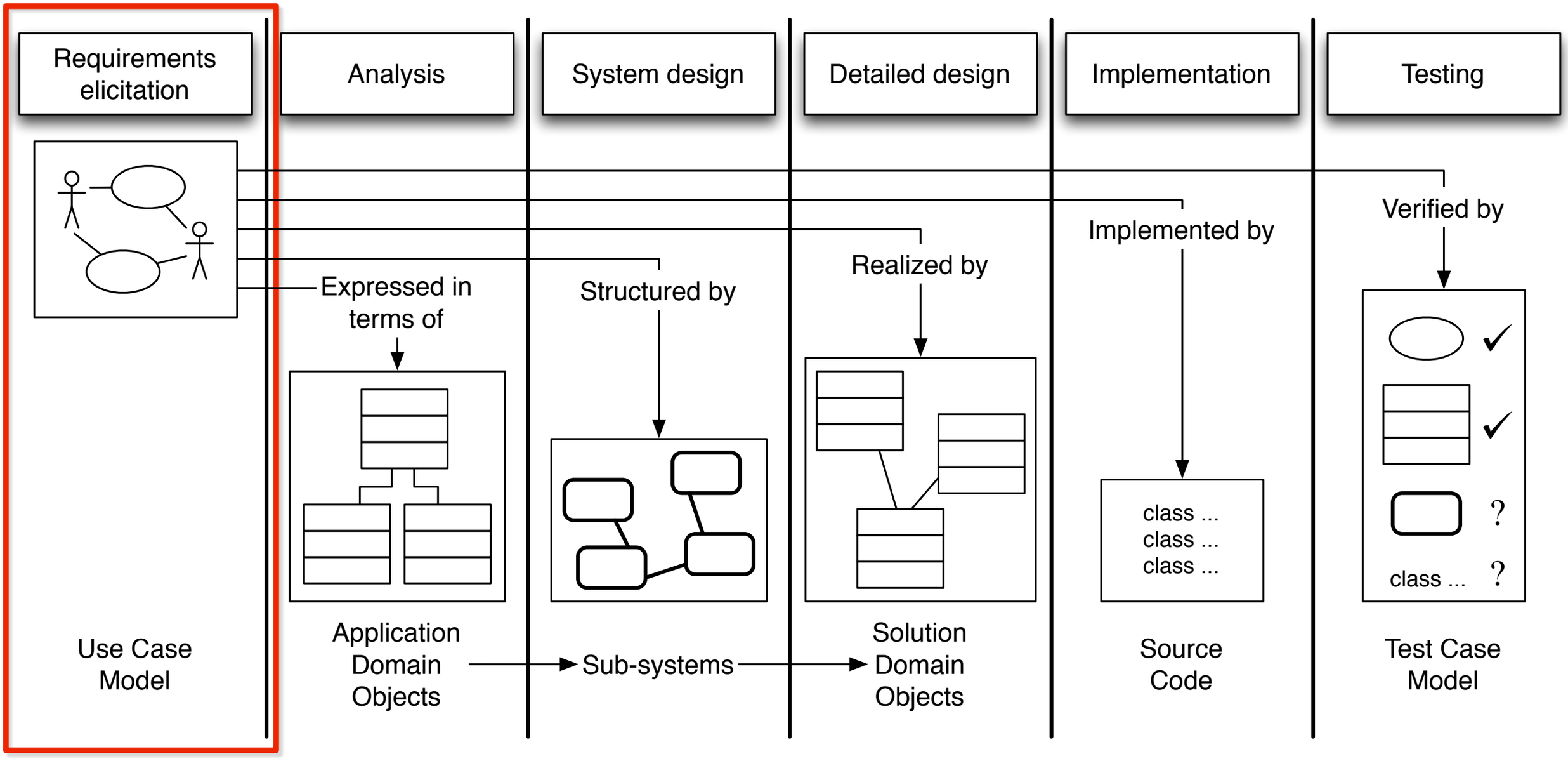
IT University
of Copenhagen

Outline

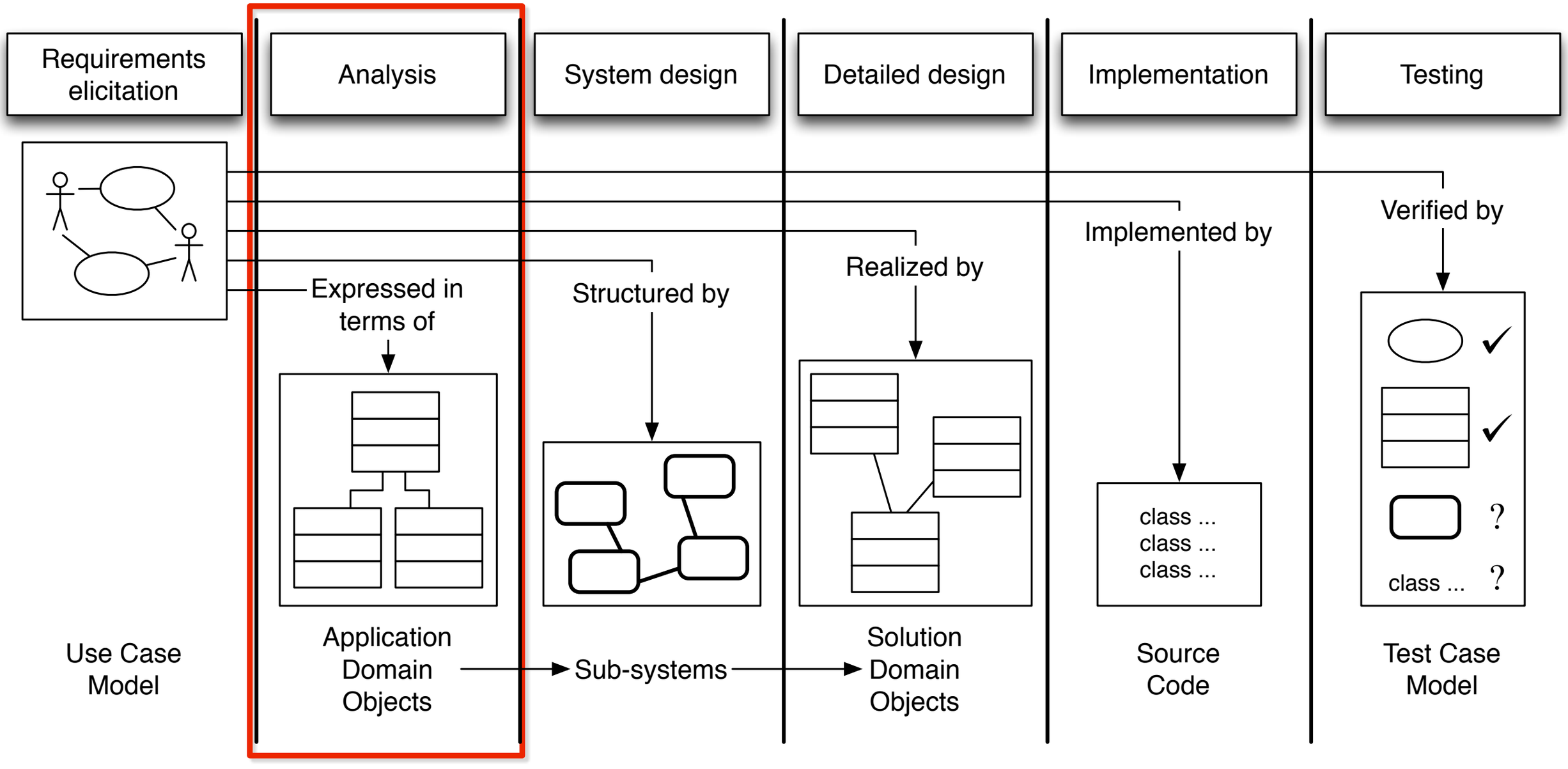
- Literature
  - [OOSE] ch. 5
- Topics covered:
  - Object-Oriented Analysis
    - Analysis model
    - UML diagrams for OOA
    - OOA activities
  - Managing Analysis
    - Documentation (RAD)
    - Client sign-off

OOA

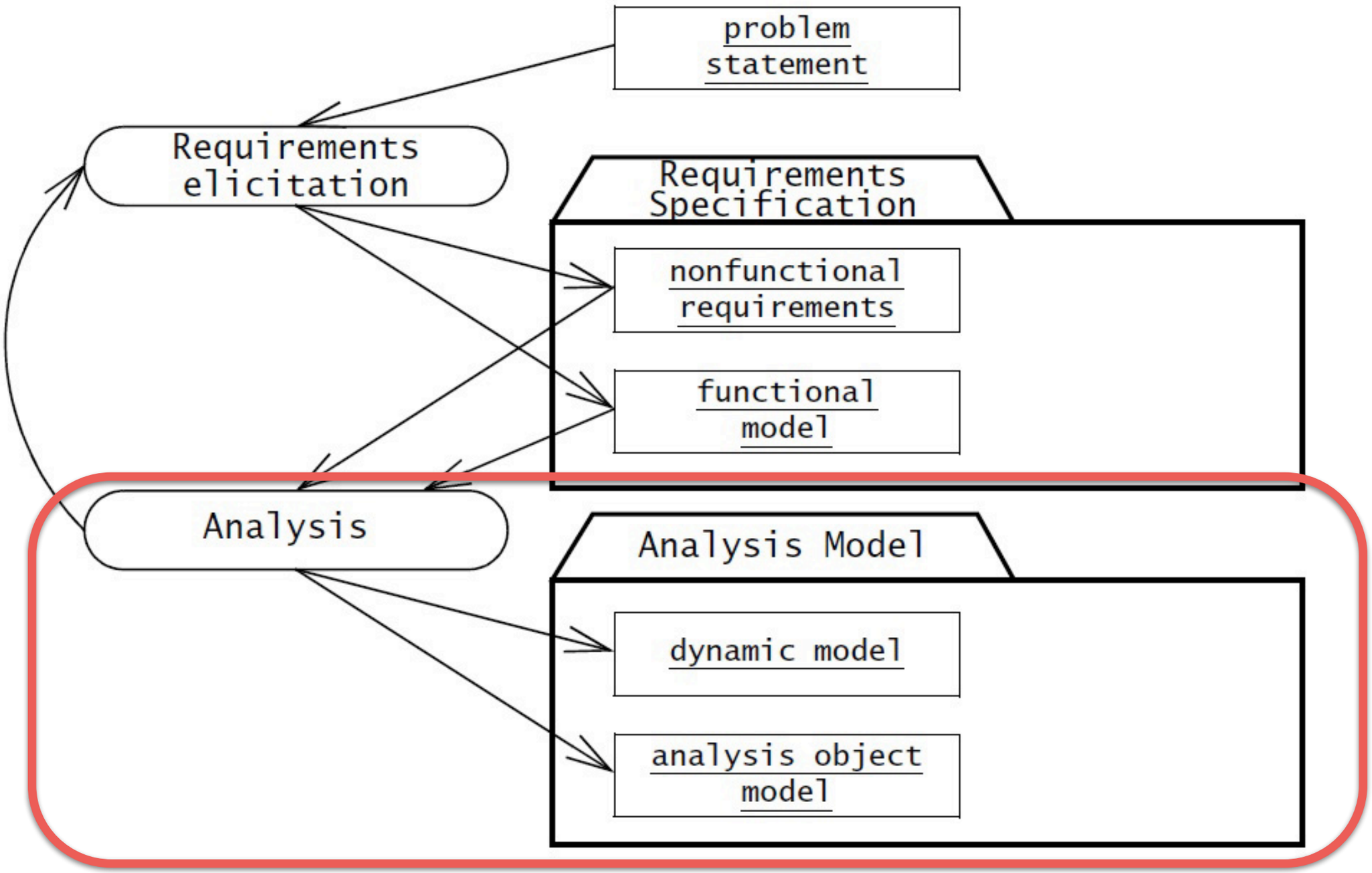
Software lifecycle activities



Software lifecycle activities



Requirements elicitation and analysis



Requirements elicitation and analysis

- Requirements elicitation:
  - definition of the system in terms understood by the client/customer/user.
    - "requirement specification".

- Analysis:
  - definition of the system in terms understood by the developer.
    - "technical specification", "analysis model".

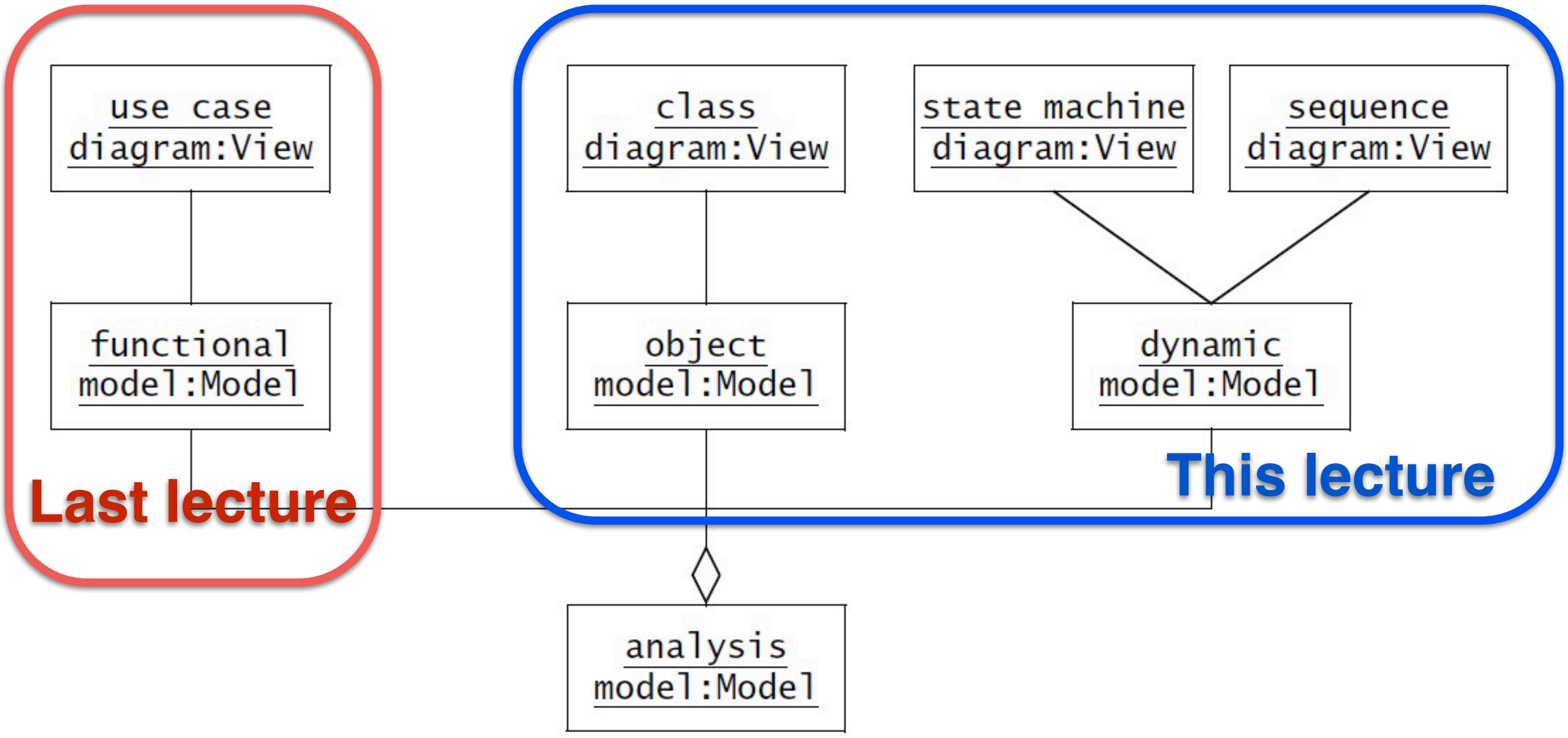
- OOA focuses on creating a model of the system:
  - may not be understandable by the client/customer/user.
  - leads to revisions of the requirements specification.

Object-oriented analysis

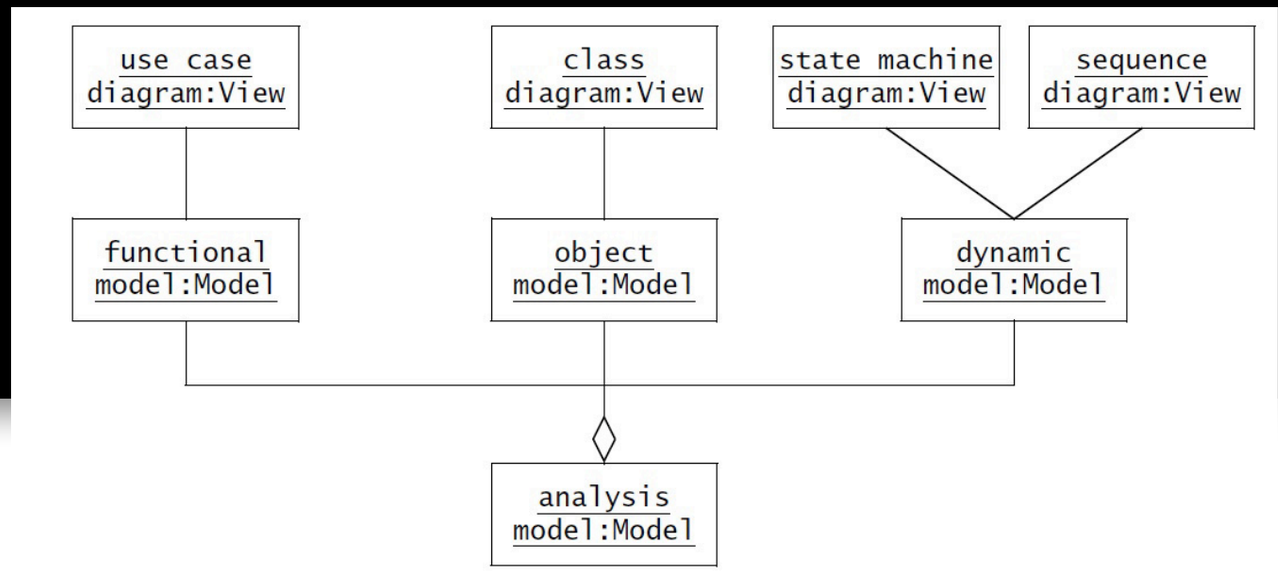
- OOA focuses on creating a model of the system:
  - that is complete, correct, consistent, and verifiable;
  - by structuring formalizing requirements;
  - which leads to revision of the requirements.

- Analysis model consists of:
  - functional model – use case model (last lecture);
  - analysis object model – class & object diagrams;
  - dynamic model – sequence & state machine diagrams.

Analysis model components



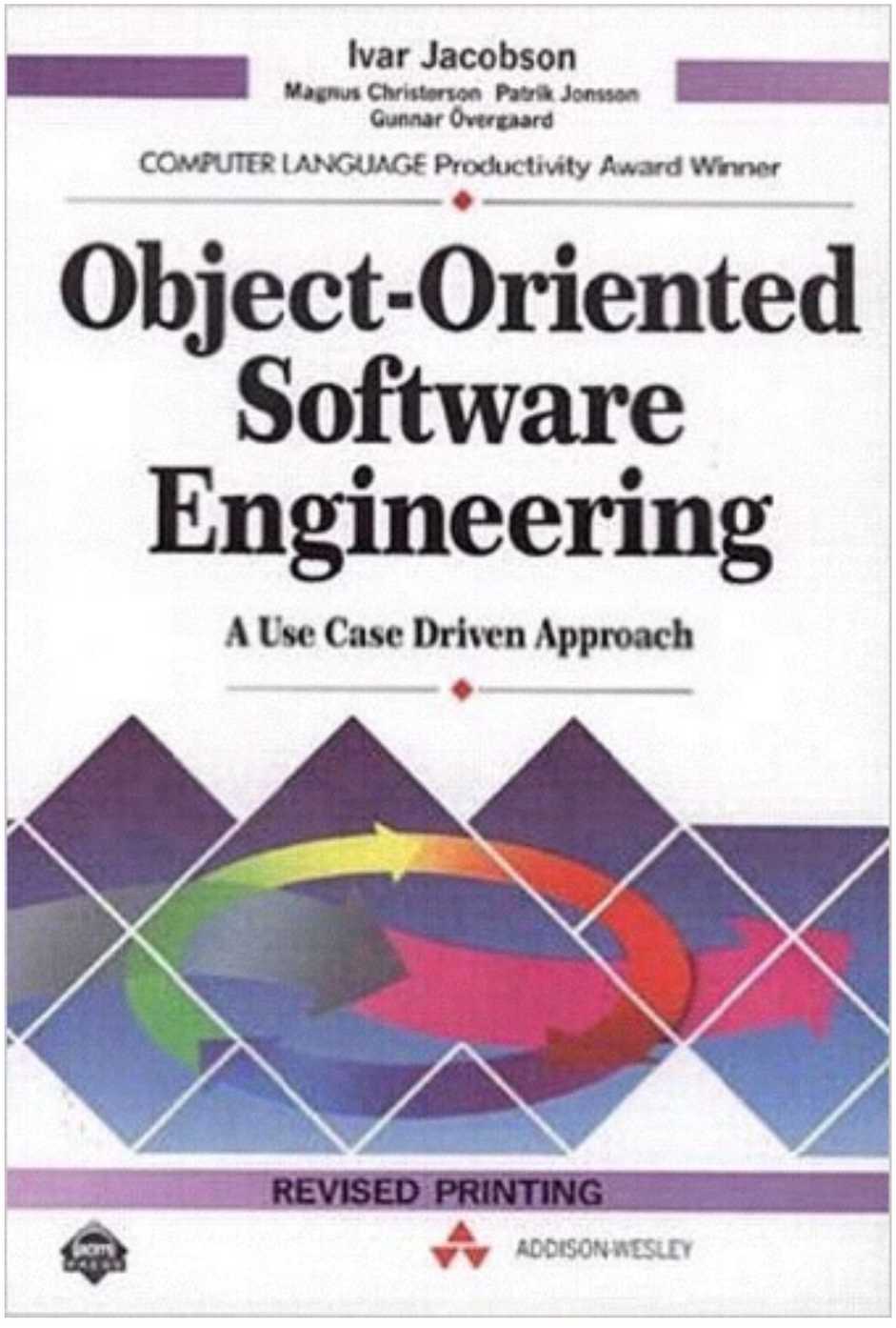
Analysis concepts



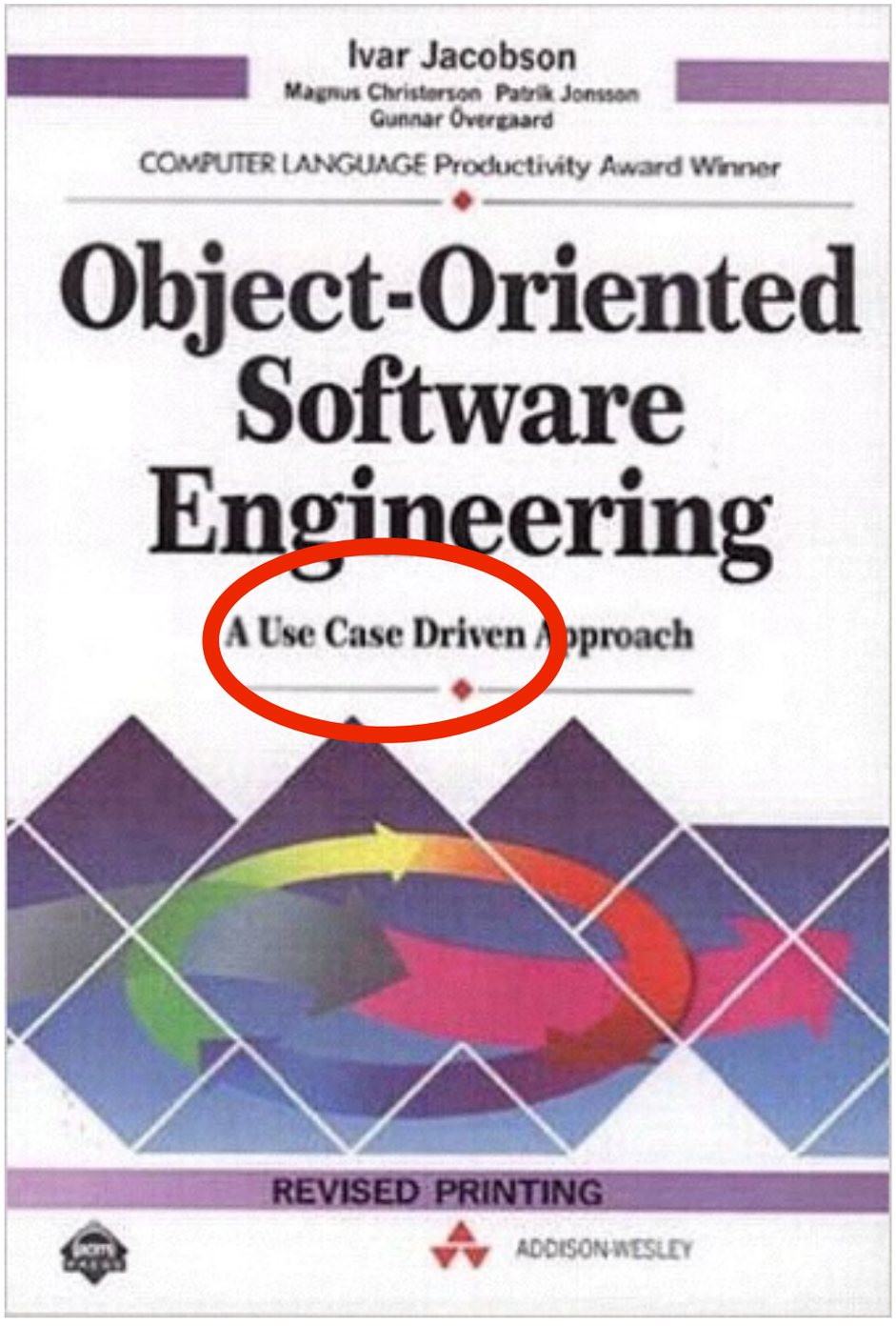
- Object-oriented analysis has two parts:
  - static model – object model
    - class diagrams
    - object diagrams
  - dynamic model
    - sequence diagrams
    - state machine diagrams

- User level concepts
  - important to remember to work on the domain (users') level
    - no technical implementation language, like
      - database, session, network, etc.
  - analysis diagram captures the "higher level" abstractions.

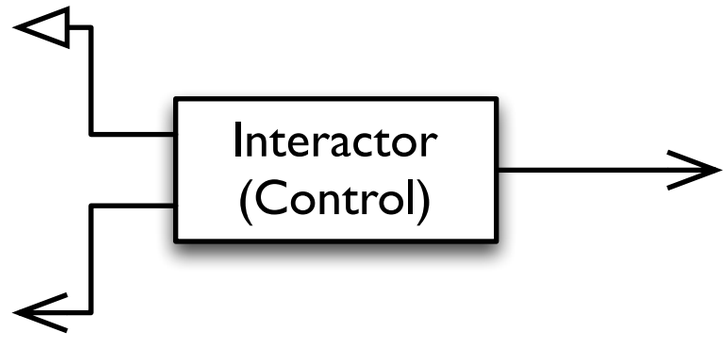
Ivar Jacobson



Ivar Jacobson

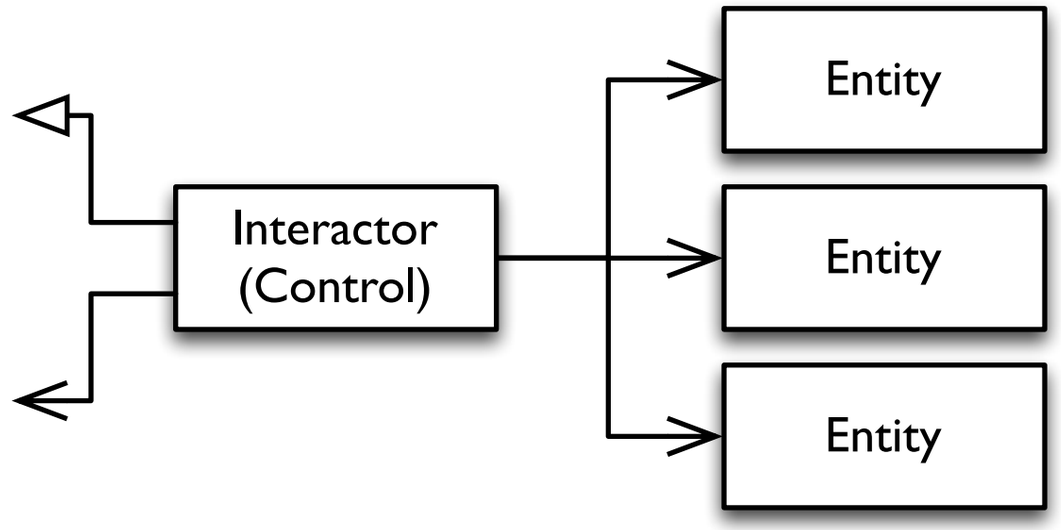


Interactor (Control) - Entity - Boundary



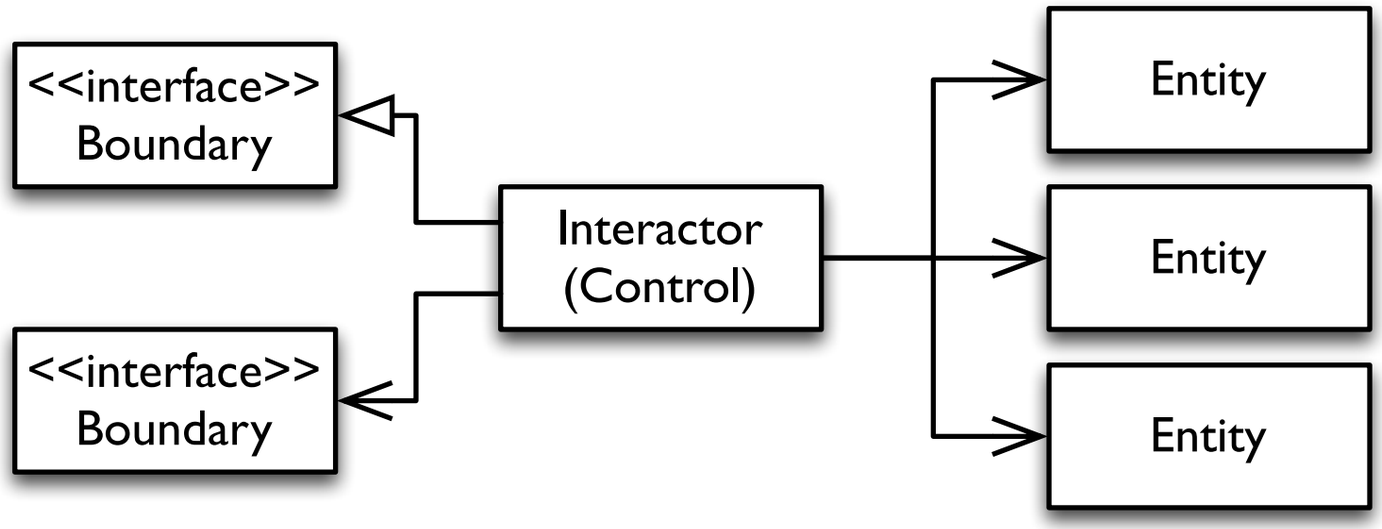
- They map the use cases.
- Application specific business rules.

Interactor (Control) - Entity - Boundary

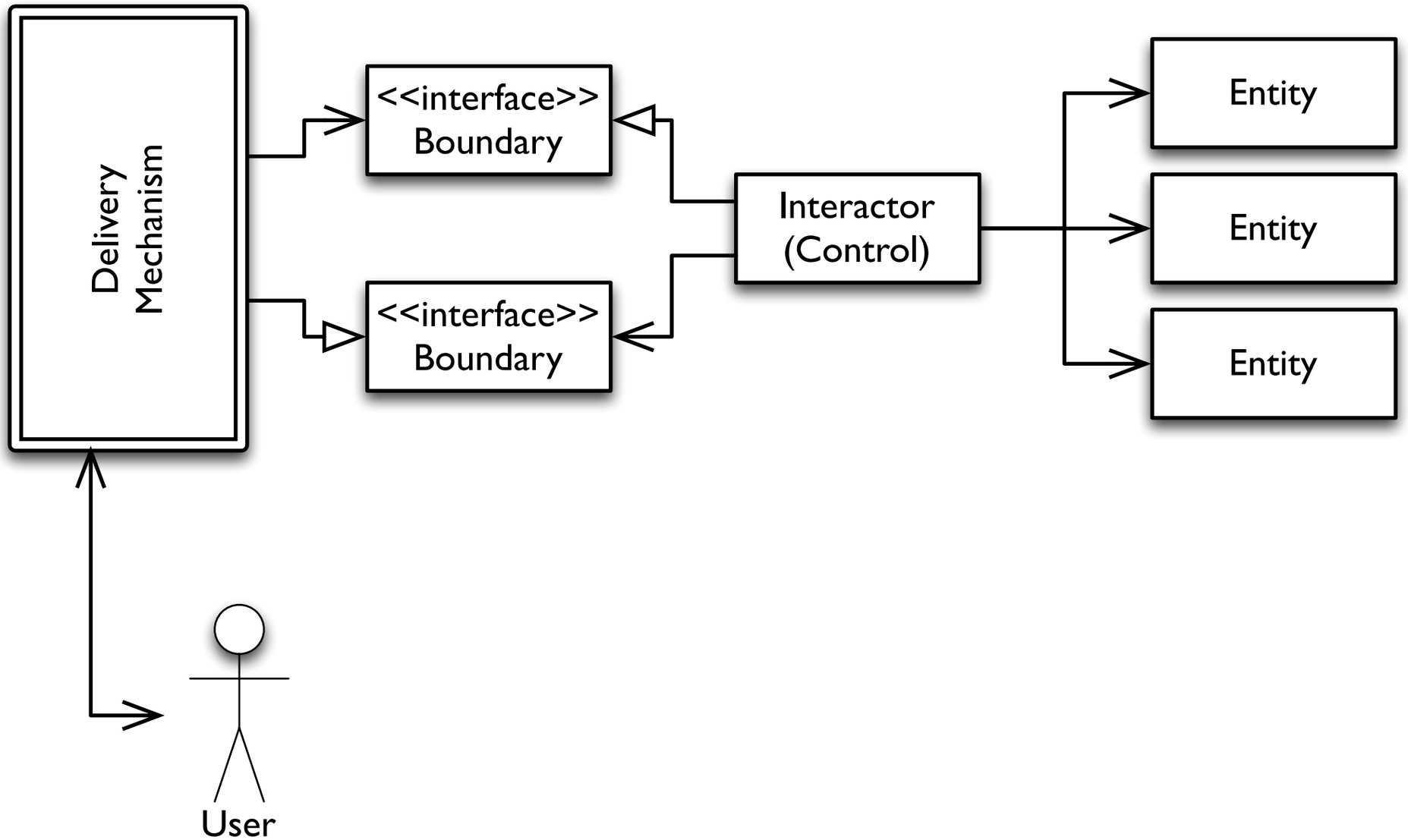


- Application <u>independent</u> business rules.

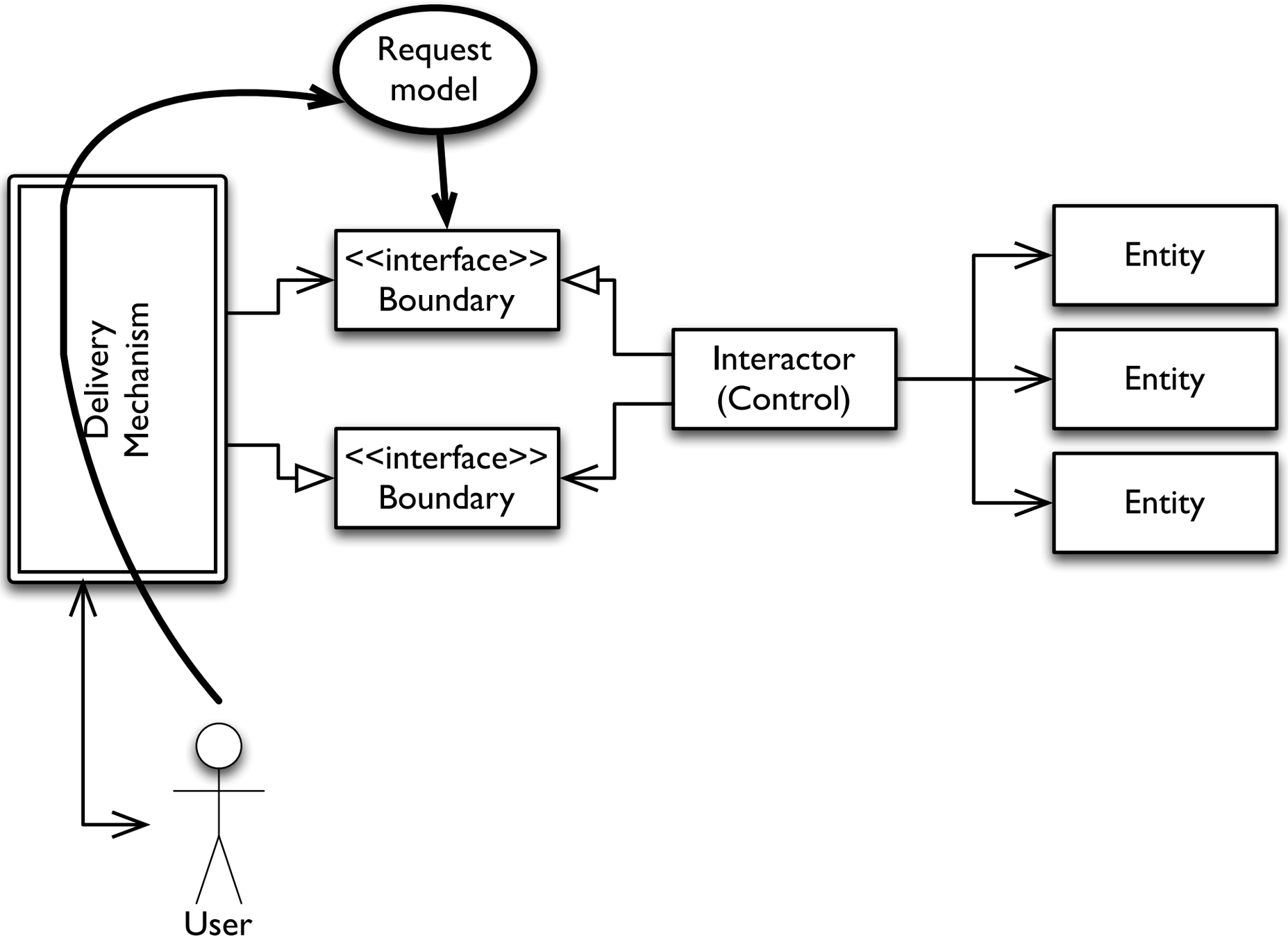
Interactor (Control) - Entity - Boundary



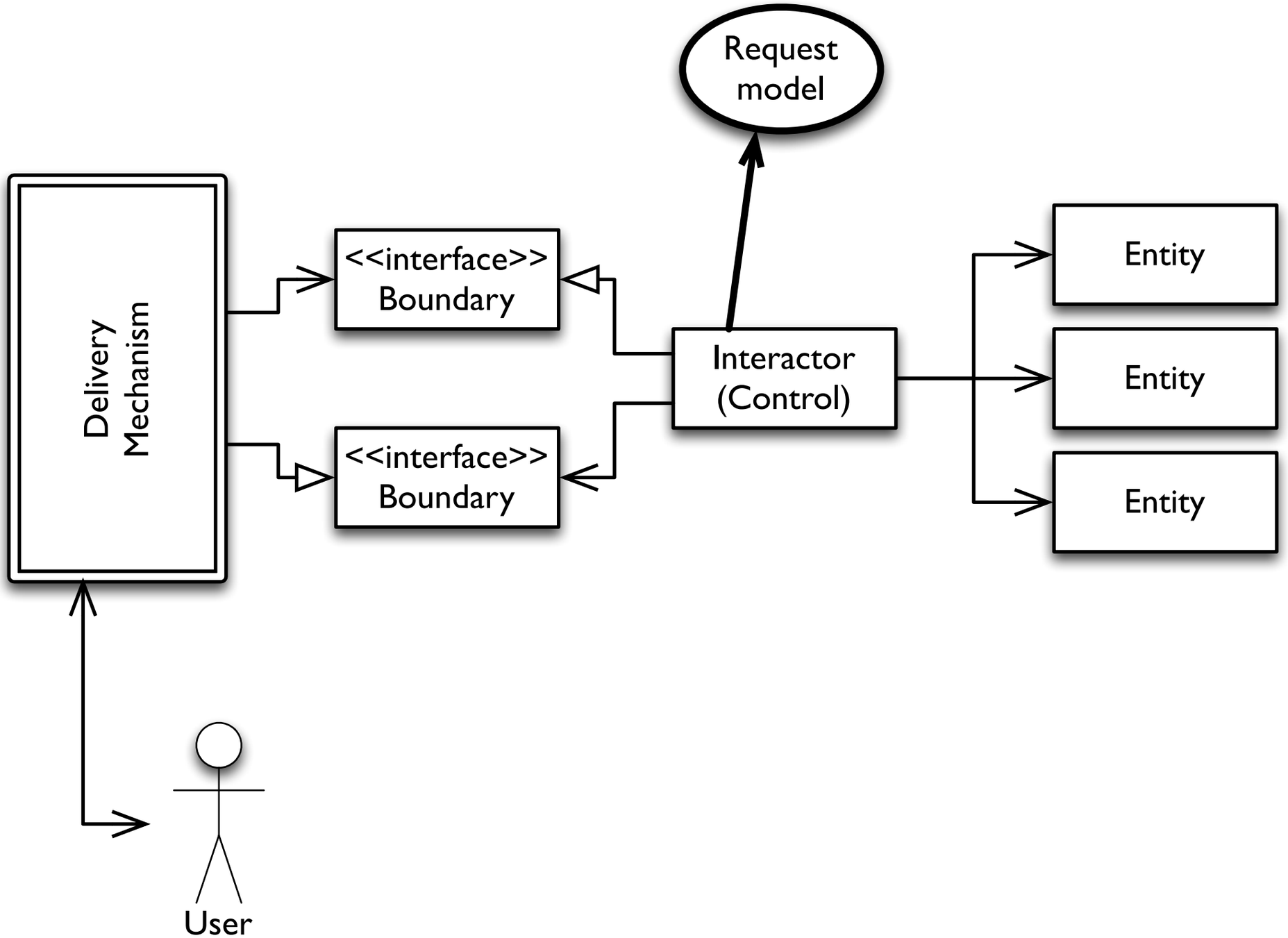
Interactor (Control) - Entity - Boundary



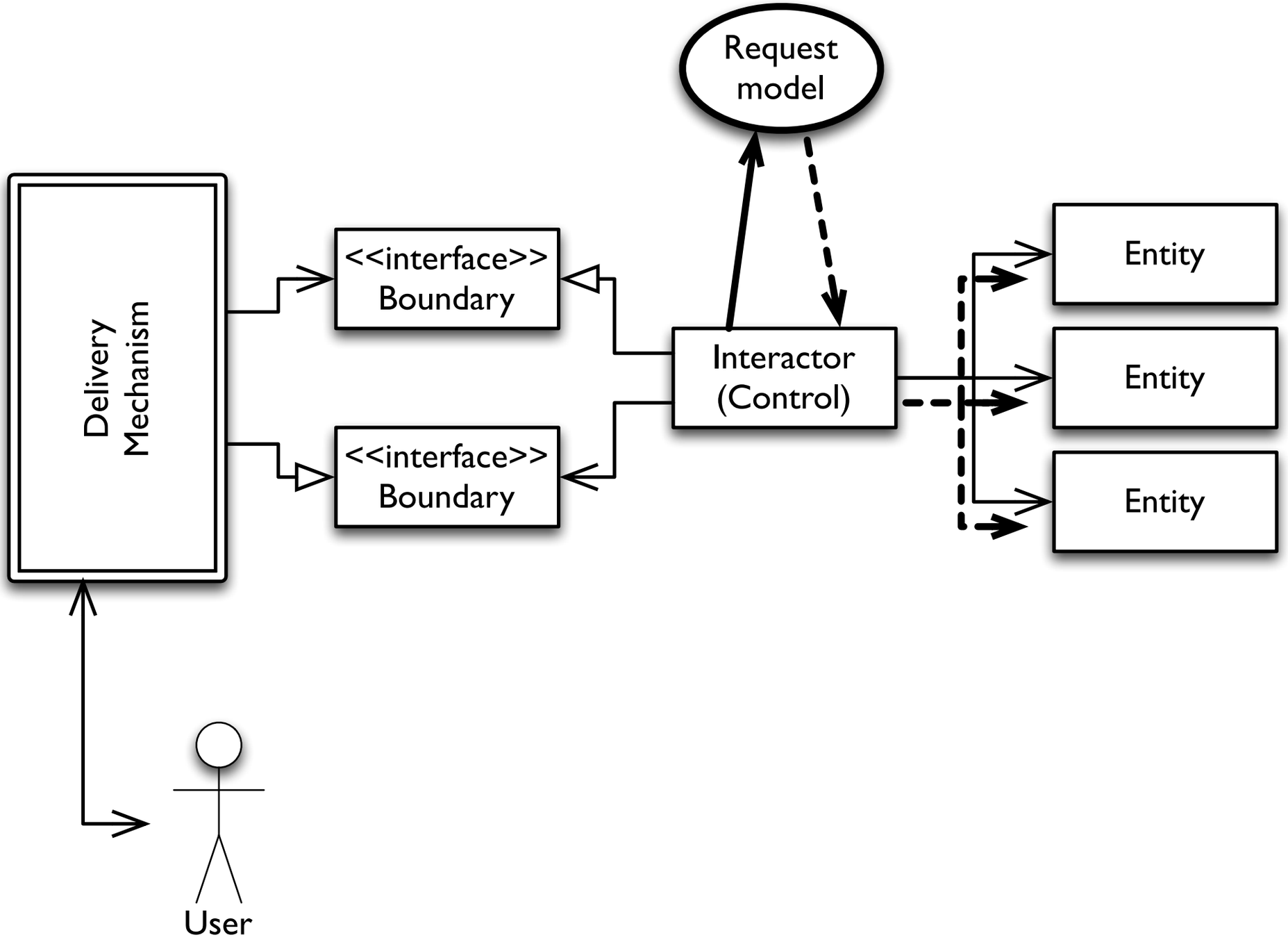
Interactor (Control) - Entity - Boundary



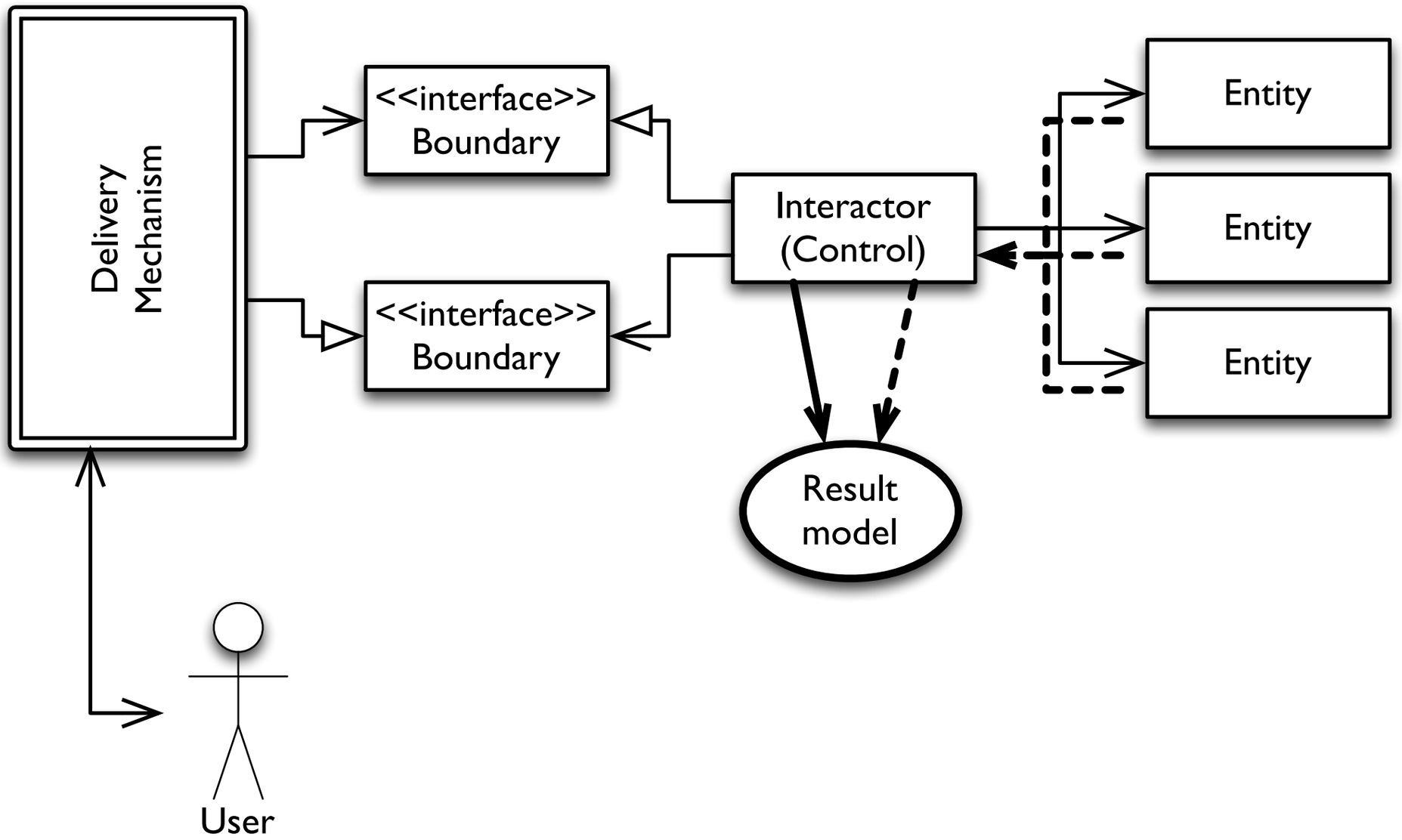
Interactor (Control) - Entity - Boundary



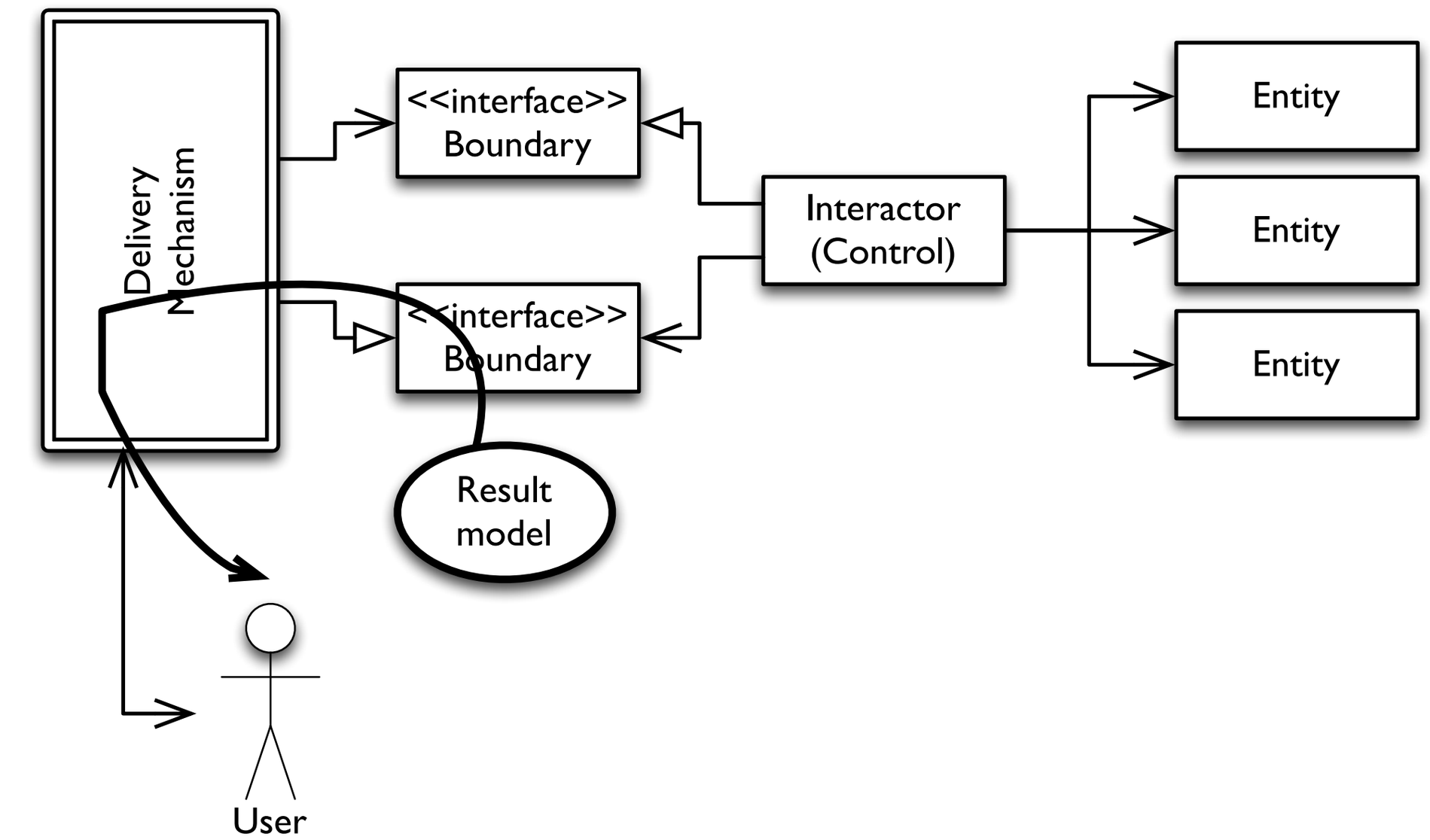
Interactor (Control) - Entity - Boundary



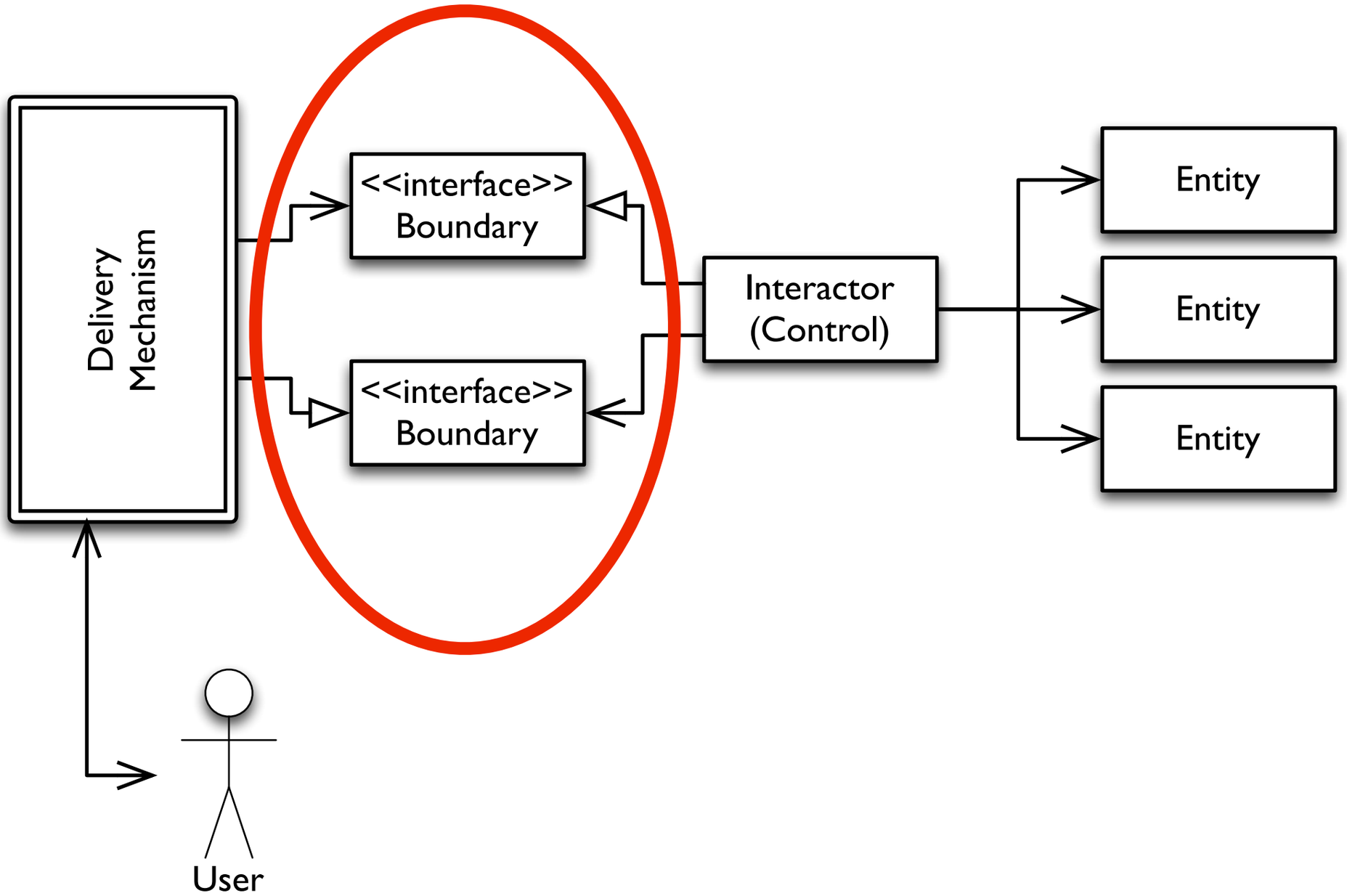
Interactor (Control) - Entity - Boundary



Interactor (Control) - Entity - Boundary



This is the power of OO languages!
… but another lecture.



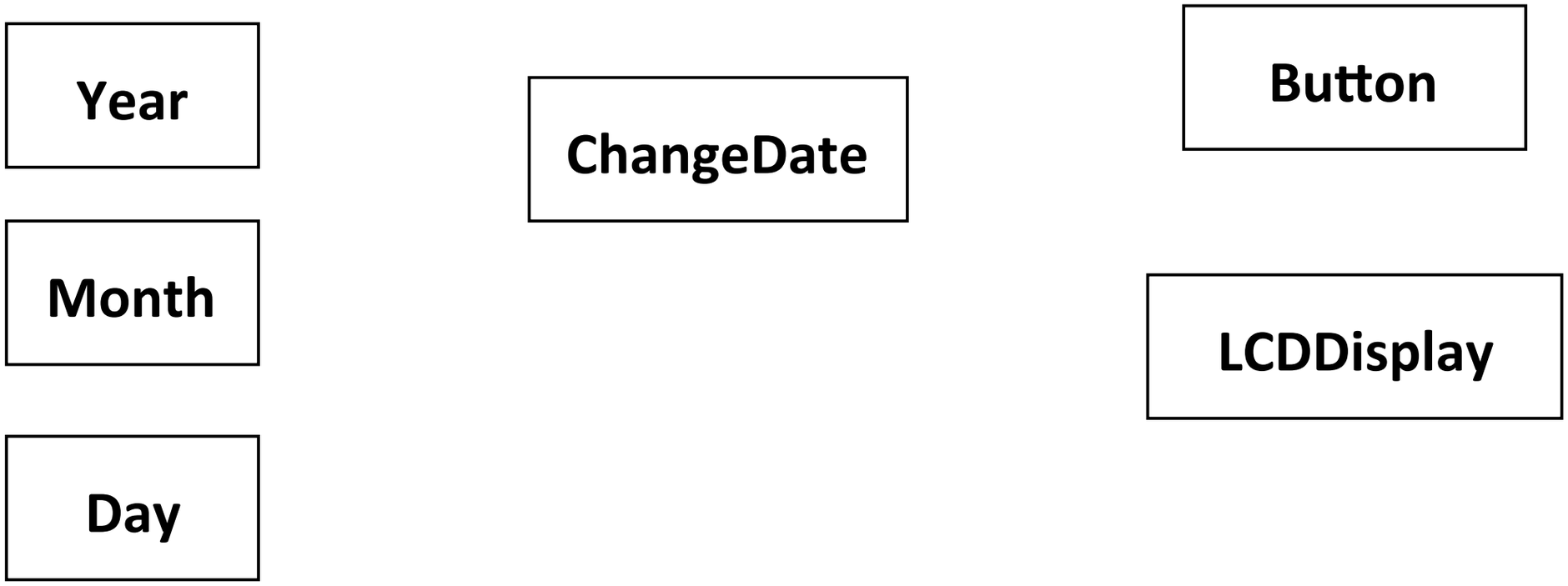
Interactor (Control) - Entity - Boundary

- Interactor (Control) objects
  - represent the control tasks performed by the system;
  - they (often) map use cases;
  - they are application specific business rules.

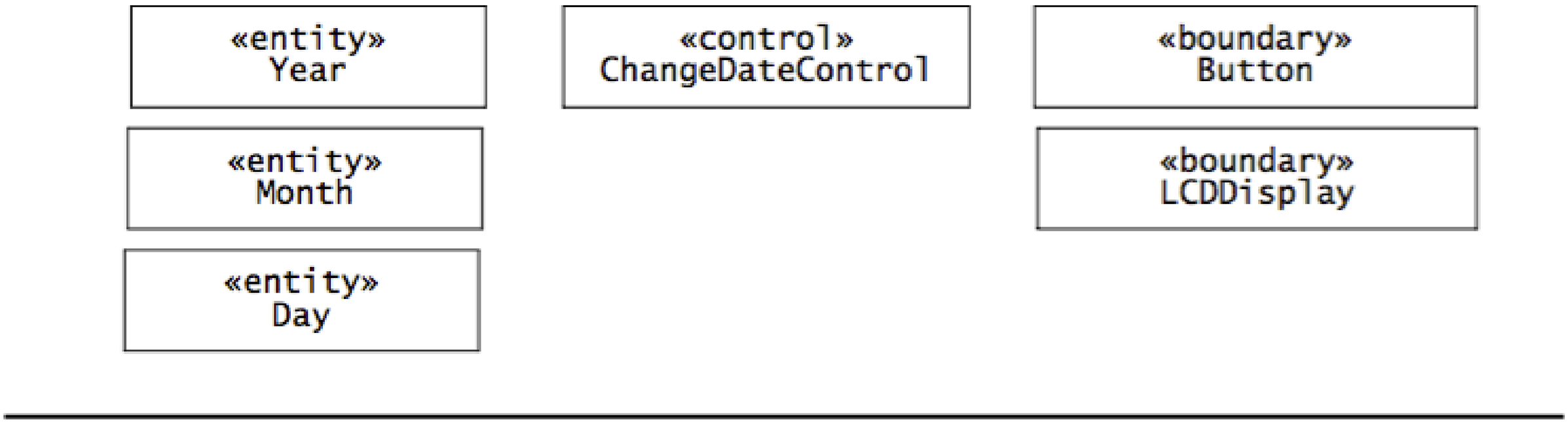
- Entity objects
  - represent the persistent information tracked by the system;
  - application domain objects also called “Business objects”;
  - they are application independent business rules.

- Boundary objects
  - Represent the interaction between the user and the system.

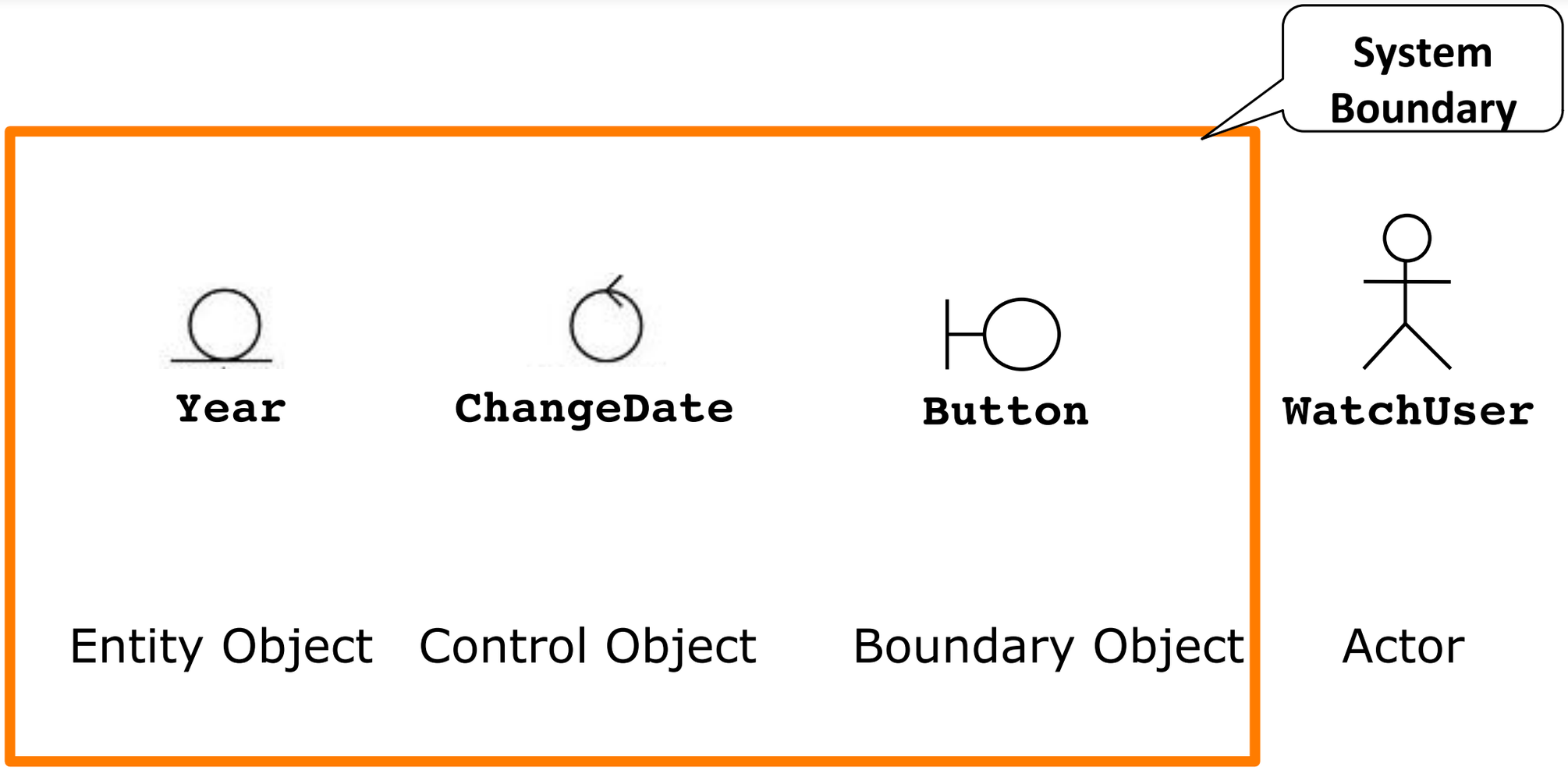
Example: 2BWatch modeling



Naming object types in UML

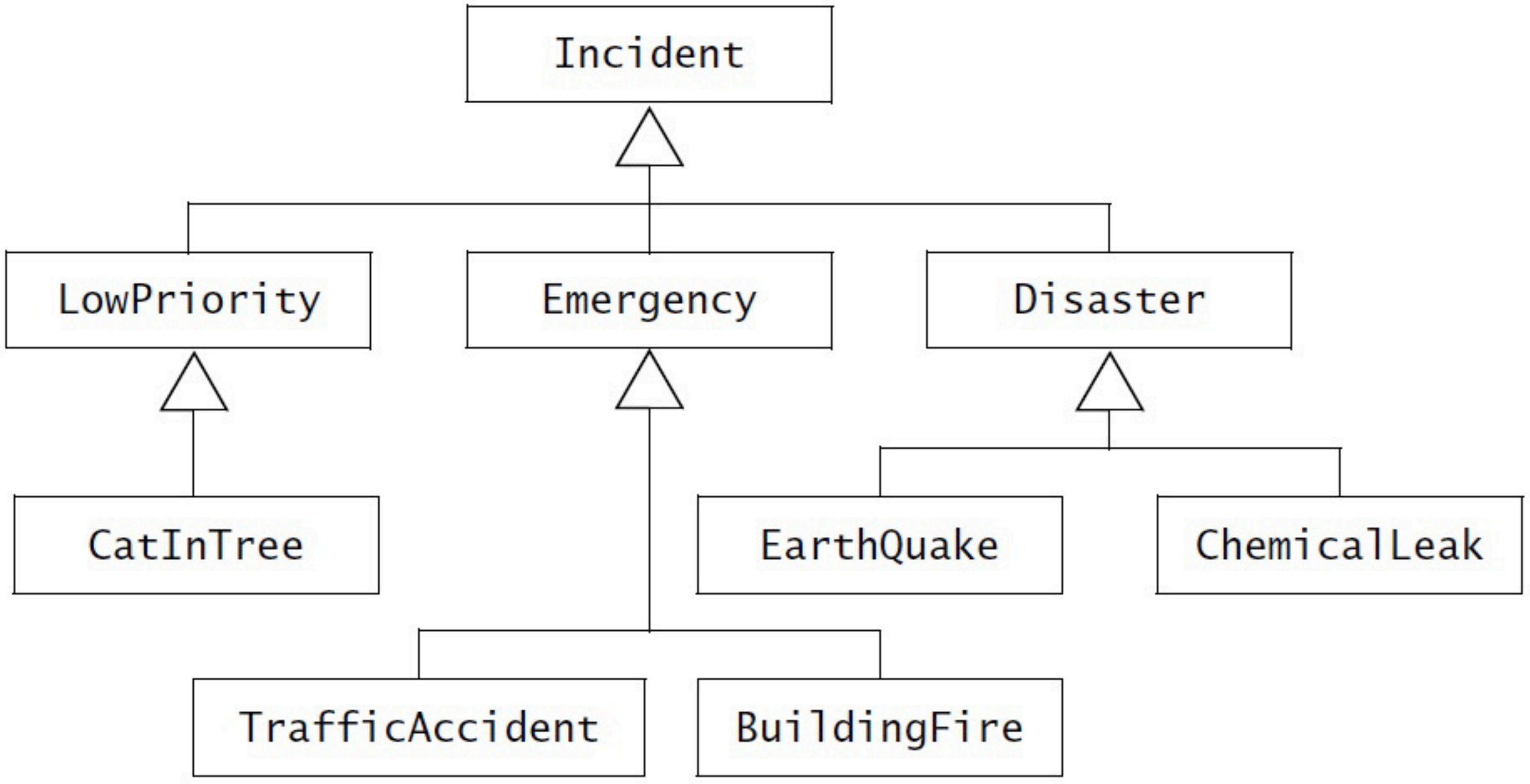


Icons for object types



- We can also use icons to identify a stereotype
  - when the stereotype is applied to a UML model element, the icon is displayed beside or above the name.

Generalization and specialization



Analysis activities:
from use cases to objects

Analysis activities

- Static modeling
  - Identifying interactor (control) objects
  - Identifying entity objects
  - Identifying boundary objects
- Dynamic modeling
  - Mapping use cases to objects using sequence diagrams
  - Modeling interactions among objects with CRC cards
- Linking and refining
  - Identifying associations
  - Identifying aggregates
  - Identifying attributes
  - Modeling state-dependent behavior of individual objects
  - Modeling inheritance relationships
- Reviewing the Analysis Model

Static modeling

Identifying entity objects

Table 5-1 Abbott's heuristics for mapping parts of speech to model components [Abbott, 1983].

Part of speech	Model component	Examples
Proper noun	Instance	Alice
Common noun	Class	Field officer
Doing verb	Operation	Creates, submits, selects
Being verb	Inheritance	Is a kind of, is one of either
Having verb	Aggregation	Has, consists of, includes
Modal verb	Constraints	Must be
Adjective	Attribute	Incident description

Finding participating objects in use cases

- Pick a use case and look at flow of events
- Do a textual analysis (noun-verb analysis):
  - nouns are candidates for objects/classes;
  - verbs are candidates for operations;
  - this is also called Abbott's technique.
- After objects/classes are found, identify their types:
  - identify real world entities that the system needs to keep track of (FieldOfficer: entity object);
  - identify real world procedures that the system needs to keep track of (EmergencyPlan: control object);
  - identify interface artifacts (PoliceStation: boundary object).

Identifying interactor (control) objects

- Interactor (control) objects represent the control tasks performed by the system.
- Heuristics:
  - make one control object pr. use case;
  - make one control object pr. actor in the use case;
  - life span of the control object should cover;
    - the extent of the use case, or
    - the extent of a user session.

Identifying entity objects

- Entity objects represent the persistent information tracked by the system
- Heuristics:
  - terms that the developer or users need to clarify to understand the use case;
  - recurring nouns in the use cases;
  - real-world entities that the system needs to track/store;
  - real-world activities that the system needs to track;
  - data sources or sinks.

Identifying boundary objects

- Boundary objects represent the system interface with the actors
- Heuristics:
  - user interface controls the user needs to initiate the use case;
  - forms that the user need to enter data into the system;
  - notices and messages from the system to the user;
  - actor terminals (if more than one);
  - always use the end-users' terms for describing interfaces.

Exercise: extracting from use cases

Use case name	`ReportEmergency`
Entry condition	1. The `FieldOfficer` activates the "Report Emergency" function of her terminal.
Flow of events	2. `FRIEND` responds by presenting a form to the officer. The form includes an emergency type menu (general emergency, fire, transportation), a location, incident description, resource request, and hazardous material fields. 3. The `FieldOfficer` completes the form by specifying minimally the emergency type and description fields. The `FieldOfficer` may also describe possible responses to the emergency situation and request specific resources. Once the form is completed, the `FieldOfficer` submits the form by pressing the "Send Report" button, at which point, the `Dispatcher` is notified. 4. The `Dispatcher` reviews the information submitted by the `FieldOfficer` and creates an `Incident` in the database by invoking the `OpenIncident` use case. All the information contained in the `FieldOfficer`'s form is automatically included in the incident. The `Dispatcher` selects a response by allocating resources to the incident (with the `AllocateResources` use case) and acknowledges the emergency report by sending a `FRIENDgram` to the `FieldOfficer`.
Exit condition	5. The `FieldOfficer` receives the acknowledgment and the selected response.

Interactor (Control) objects

Table 5-4 Control objects for the ReportEmergency use case.

ReportEmergencyControl	Manages the ReportEmergency reporting function on the FieldOfficerStation. This object is created when the FieldOfficer selects the "Report Emergency" button. It then creates an EmergencyReportForm and presents it to the FieldOfficer. After submitting the form, this object then collects the information from the form, creates an EmergencyReport, and forwards it to the Dispatcher. The control object then waits for an acknowledgment to come back from the DispatcherStation. When the acknowledgment is received, the ReportEmergencyControl object creates an AcknowledgmentNotice and displays it to the FieldOfficer.
ManageEmergencyControl	Manages the ReportEmergency reporting function on the DispatcherStation. This object is created when an EmergencyReport is received. It then creates an IncidentForm and displays it to the Dispatcher. Once the Dispatcher has created an Incident, allocated Resources, and submitted an acknowledgment, ManageEmergencyControl forwards the acknowledgment to the FieldOfficerStation.

Entity objects

Table 5-2 Entity objects for the `ReportEmergency` use case.

`Dispatcher`	Police officer who manages `Incidents`. A `Dispatcher` opens, documents, and closes `Incidents` in response to `Emergency Reports` and other communication with `FieldOfficers`. `Dispatchers` are identified by badge numbers.
`EmergencyReport`	Initial report about an `Incident` from a `FieldOfficer` to a `Dispatcher`. An `EmergencyReport` usually triggers the creation of an `Incident` by the `Dispatcher`. An `EmergencyReport` is composed of an emergency level, a type (fire, road accident, other), a location, and a description.
`FieldOfficer`	Police or fire officer on duty. A `FieldOfficer` can be allocated to, at most, one `Incident` at a time. `FieldOfficers` are identified by badge numbers.
`Incident`	Situation requiring attention from a `FieldOfficer`. An `Incident` may be reported in the system by a `FieldOfficer` or anybody else external to the system. An `Incident` is composed of a description, a response, a status (open, closed, documented), a location, and a number of `FieldOfficers`.

Boundary objects

Table 5-3 Boundary objects for the `ReportEmergency` use case.

`AcknowledgmentNotice`	Notice used for displaying the `Dispatcher`'s acknowledgment to the `FieldOfficer`.
`DispatcherStation`	Computer used by the `Dispatcher`.
`ReportEmergencyButton`	Button used by a `FieldOfficer` to initiate the `ReportEmergency` use case.
`EmergencyReportForm`	Form used for the input of the `ReportEmergency`. This form is presented to the `FieldOfficer` on the `FieldOfficerStation` when the "Report Emergency" function is selected. The `EmergencyReportForm` contains fields for specifying all attributes of an emergency report and a button (or other control) for submitting the completed form.
`FieldOfficerStation`	Mobile computer used by the `FieldOfficer`.
`IncidentForm`	Form used for the creation of `Incidents`. This form is presented to the `Dispatcher` on the `DispatcherStation` when the `EmergencyReport` is received. The `Dispatcher` also uses this form to allocate resources and to acknowledge the `FieldOfficer`'s report.

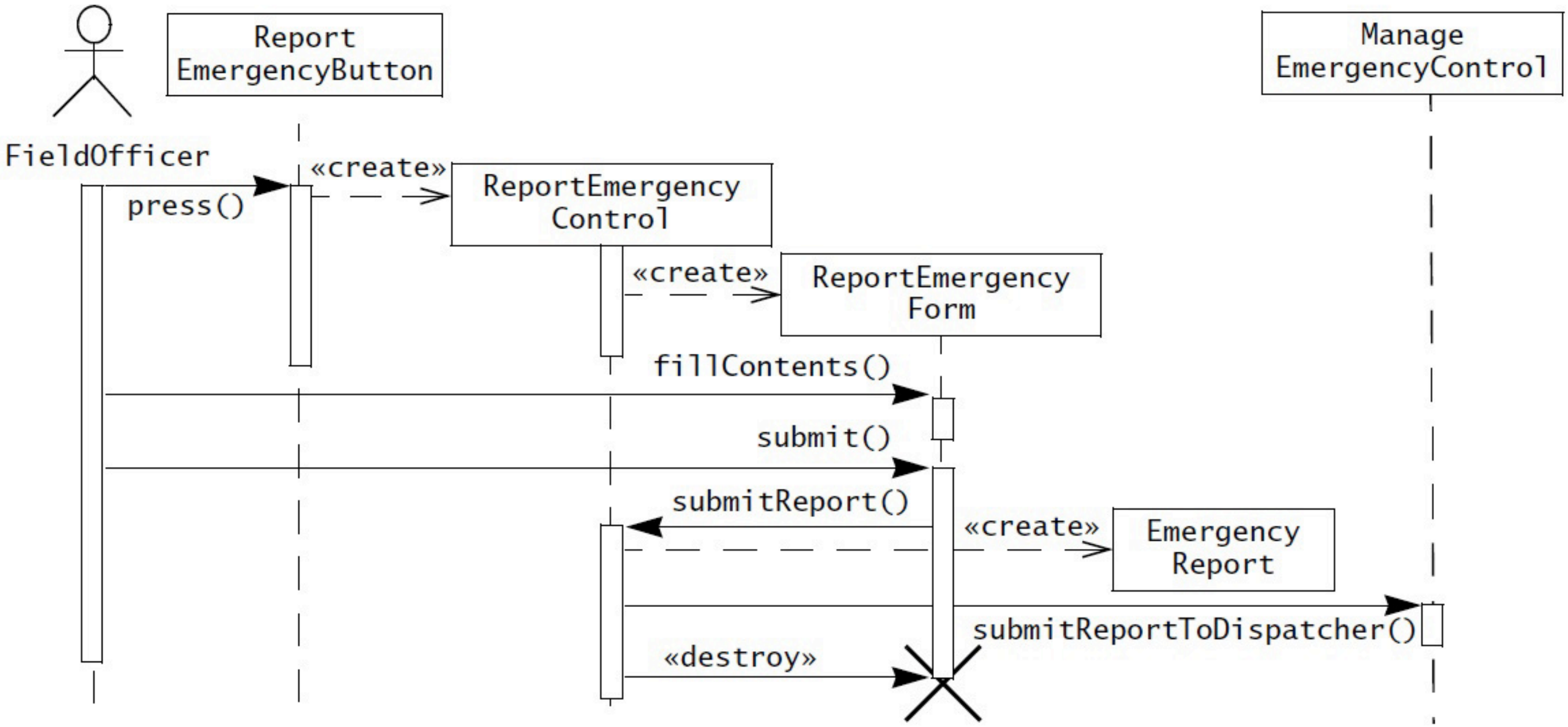
Dynamic modeling

Sequence diagrams

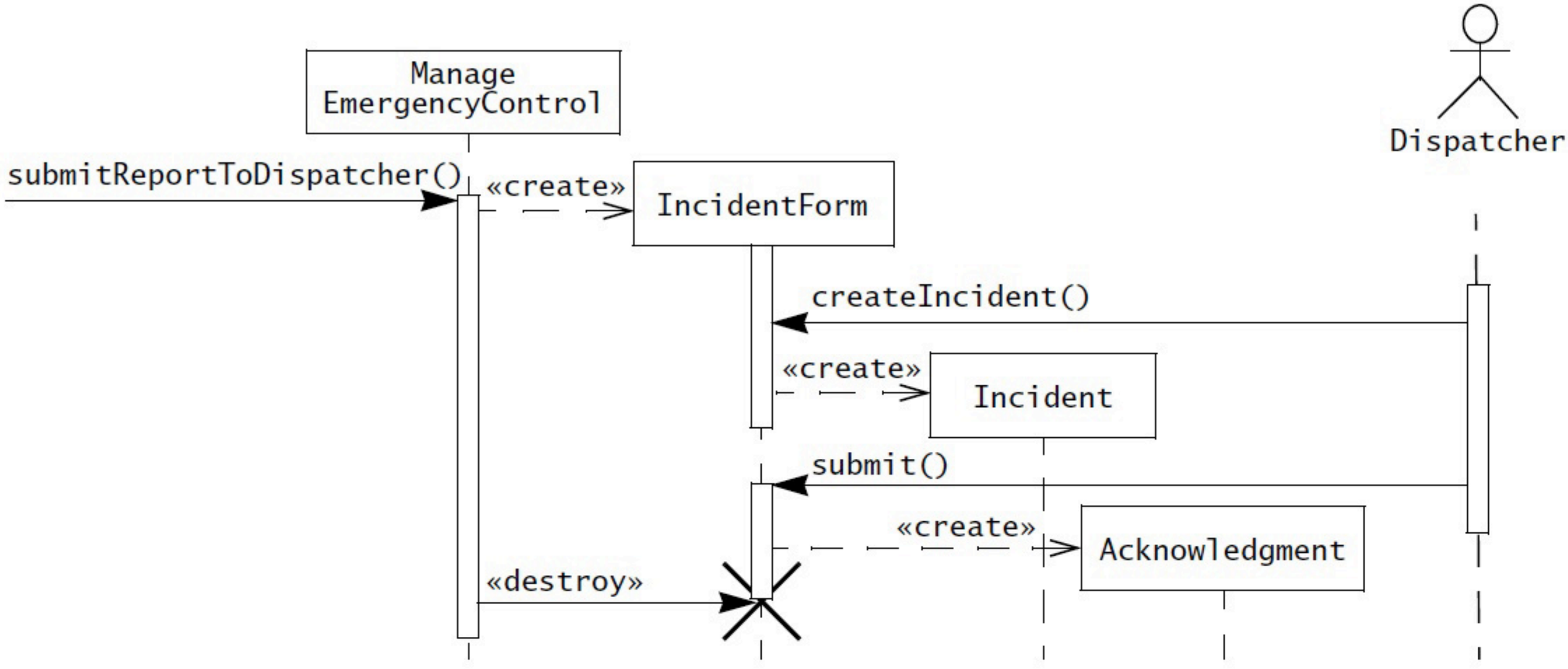
- Sequence diagram ties use cases with objects.
  - Describes the dynamic behavior between several objects over time.
  - Good for real-time specifications.

- Heuristics:
  - the first column corresponds to the initiating actor of the use case;
  - the second column should be a boundary object;
  - the third column should be the control object that manage the use case;
  - control objects are created by boundary objects;
  - boundary objects are created by control objects;
  - entity objects are accessed by control and boundary objects;
  - entity objects never access boundary or control objects.

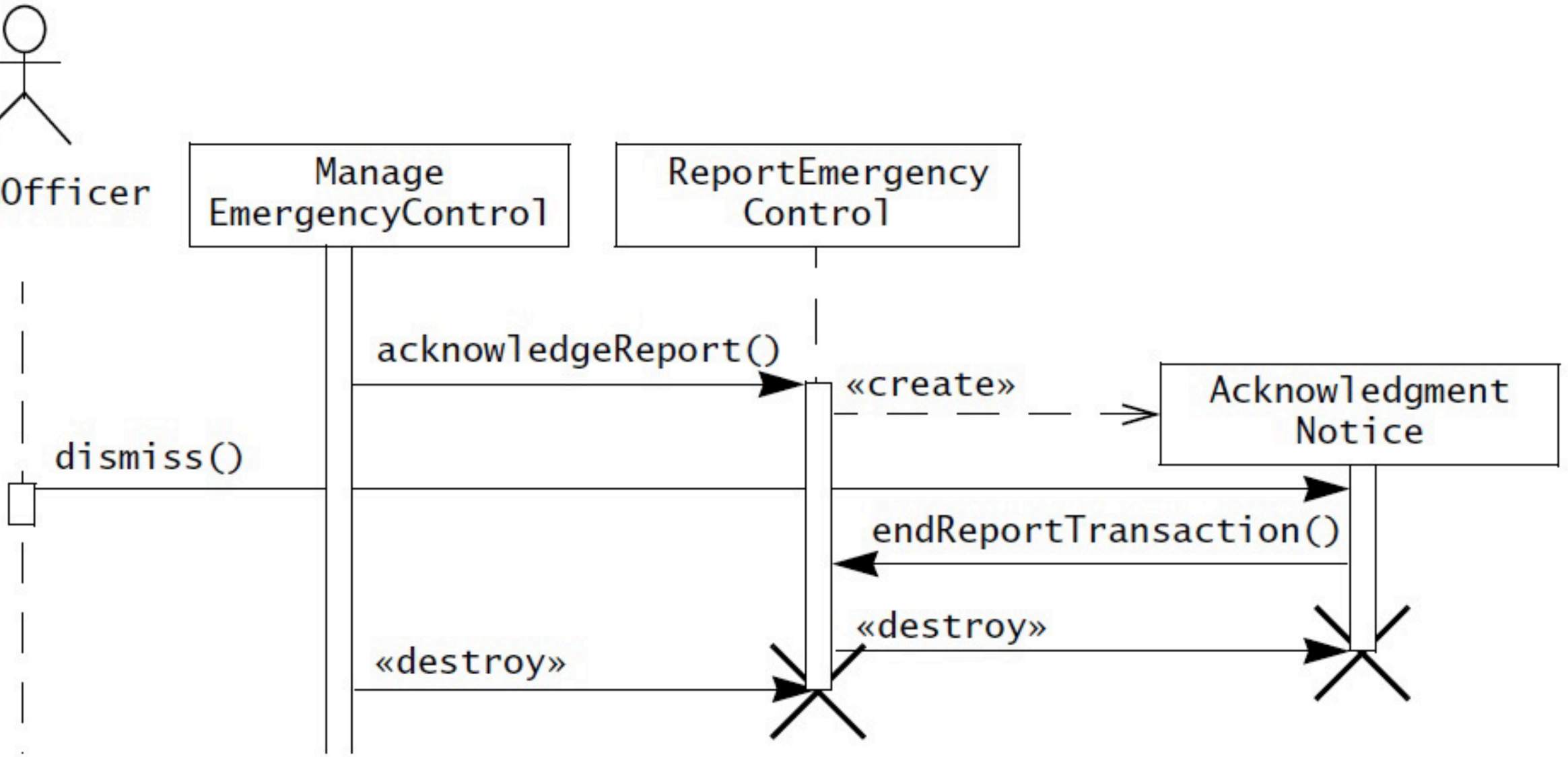
ReportEmergency use case I



ReportEmergency use case II

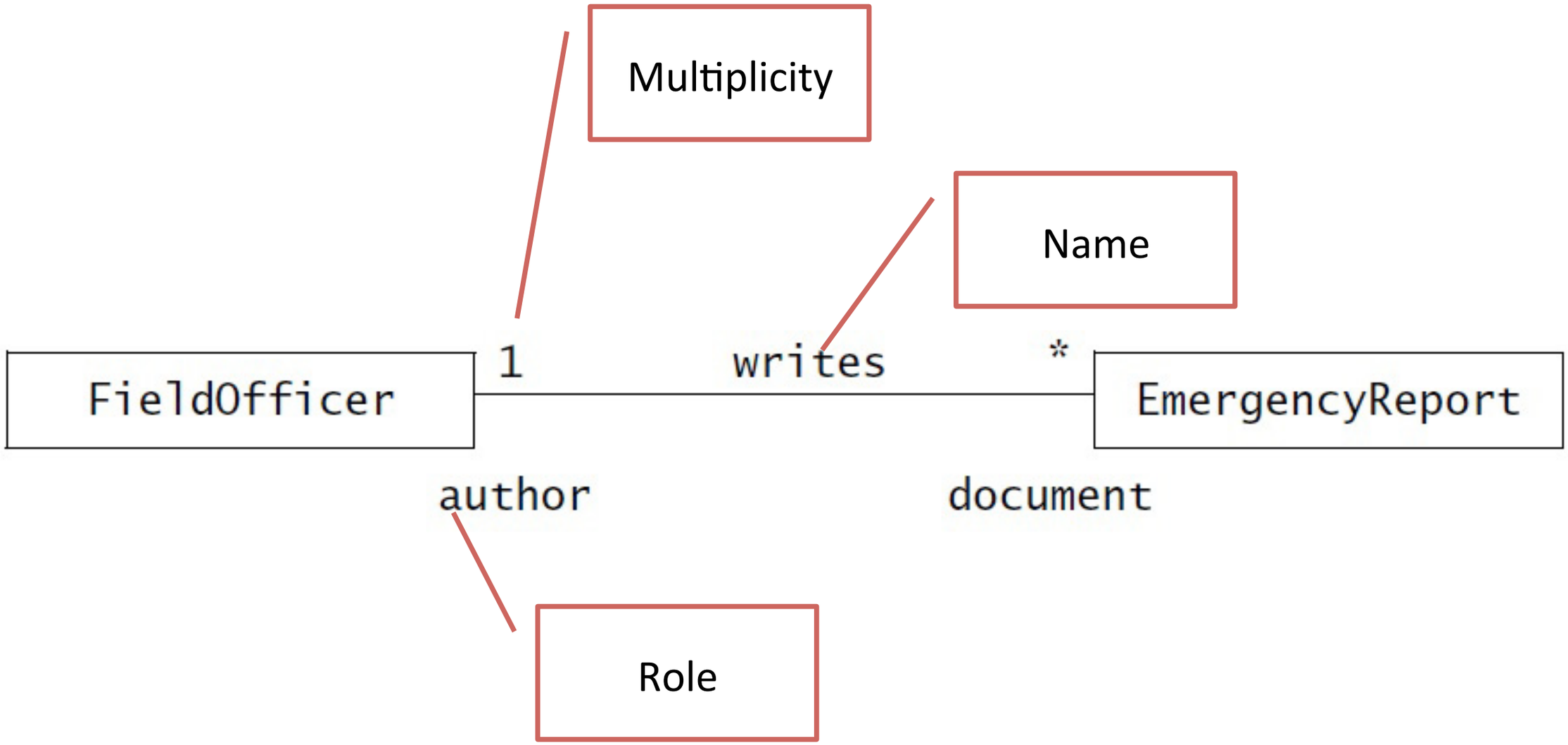


ReportEmergency use case III

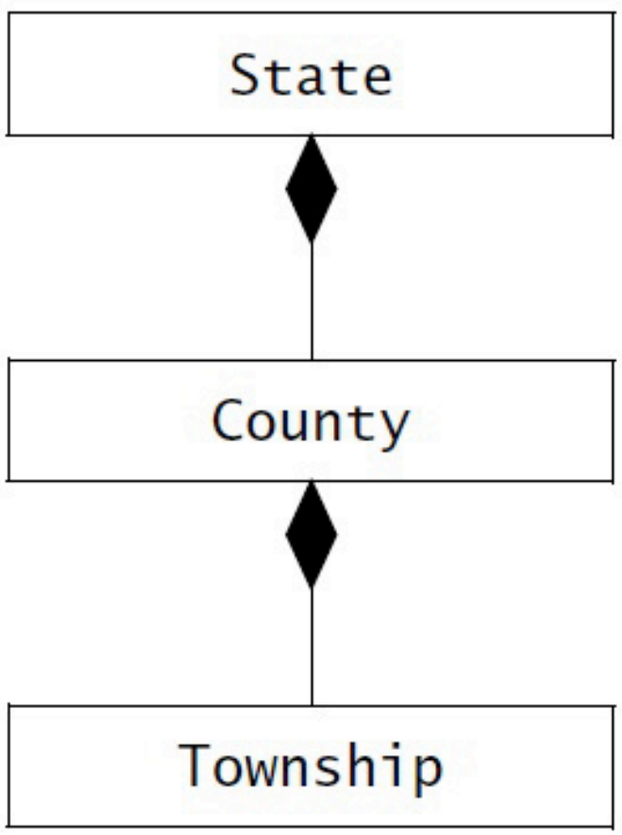


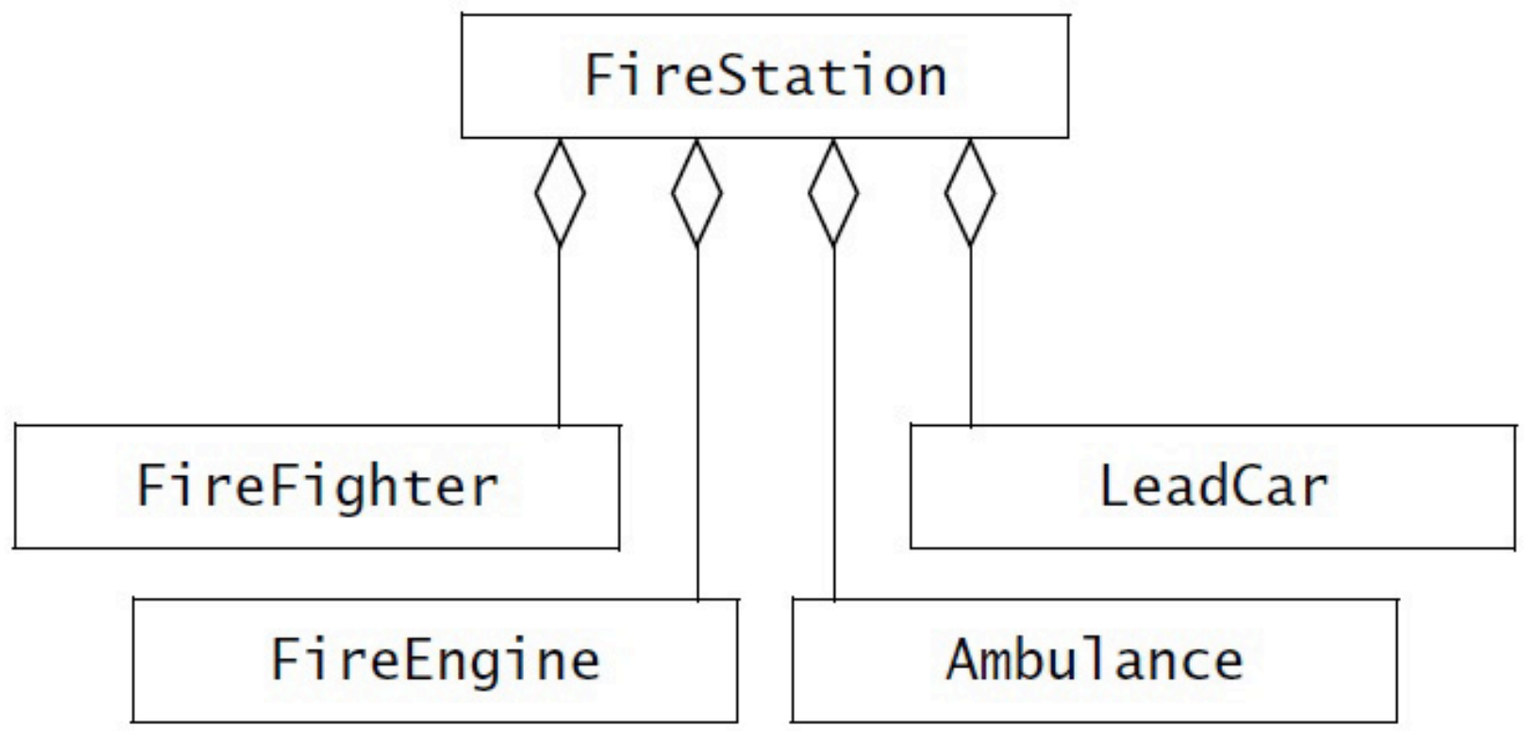
Linking and refining

Identifying associations

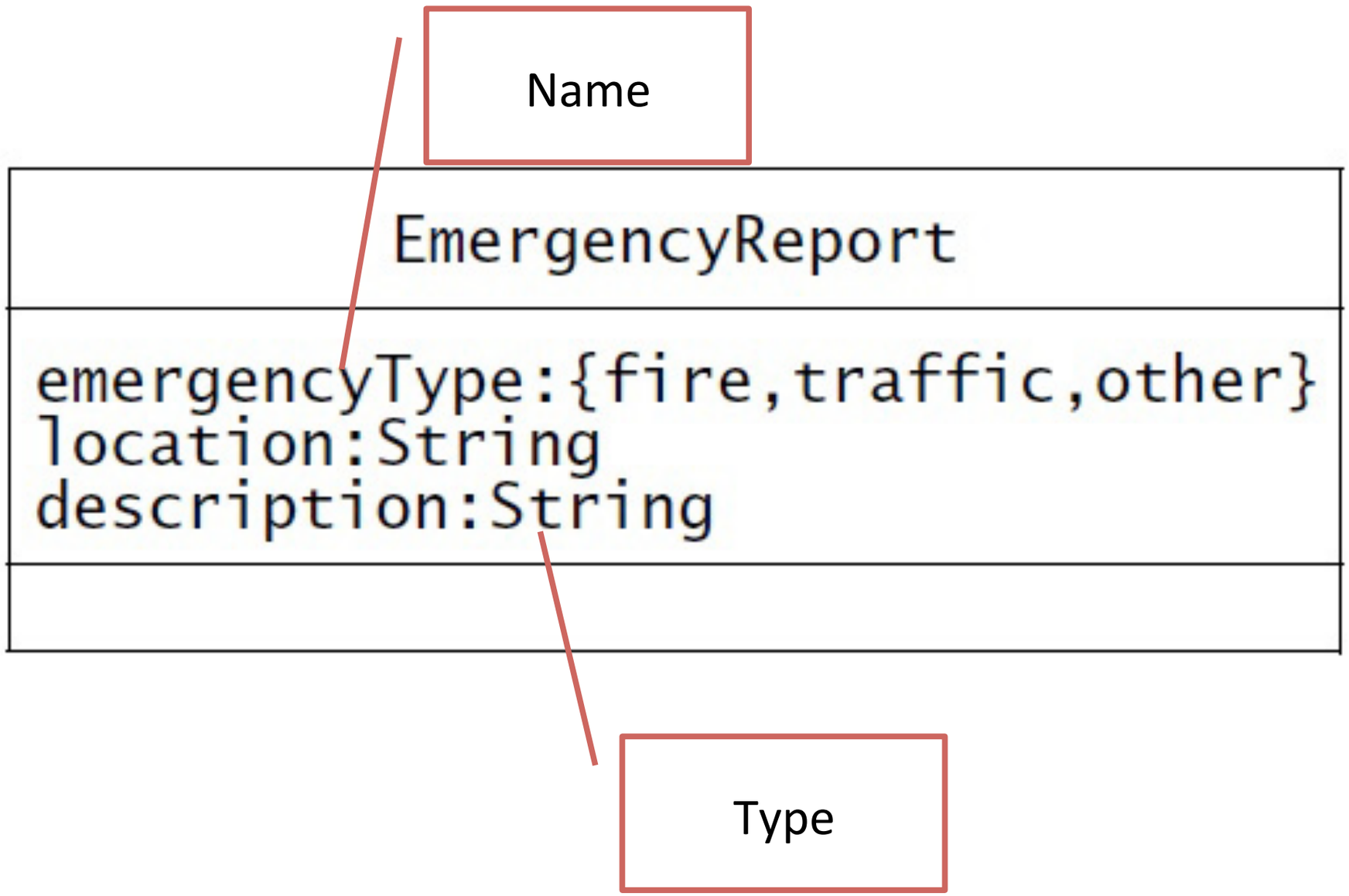


Identifying aggregates





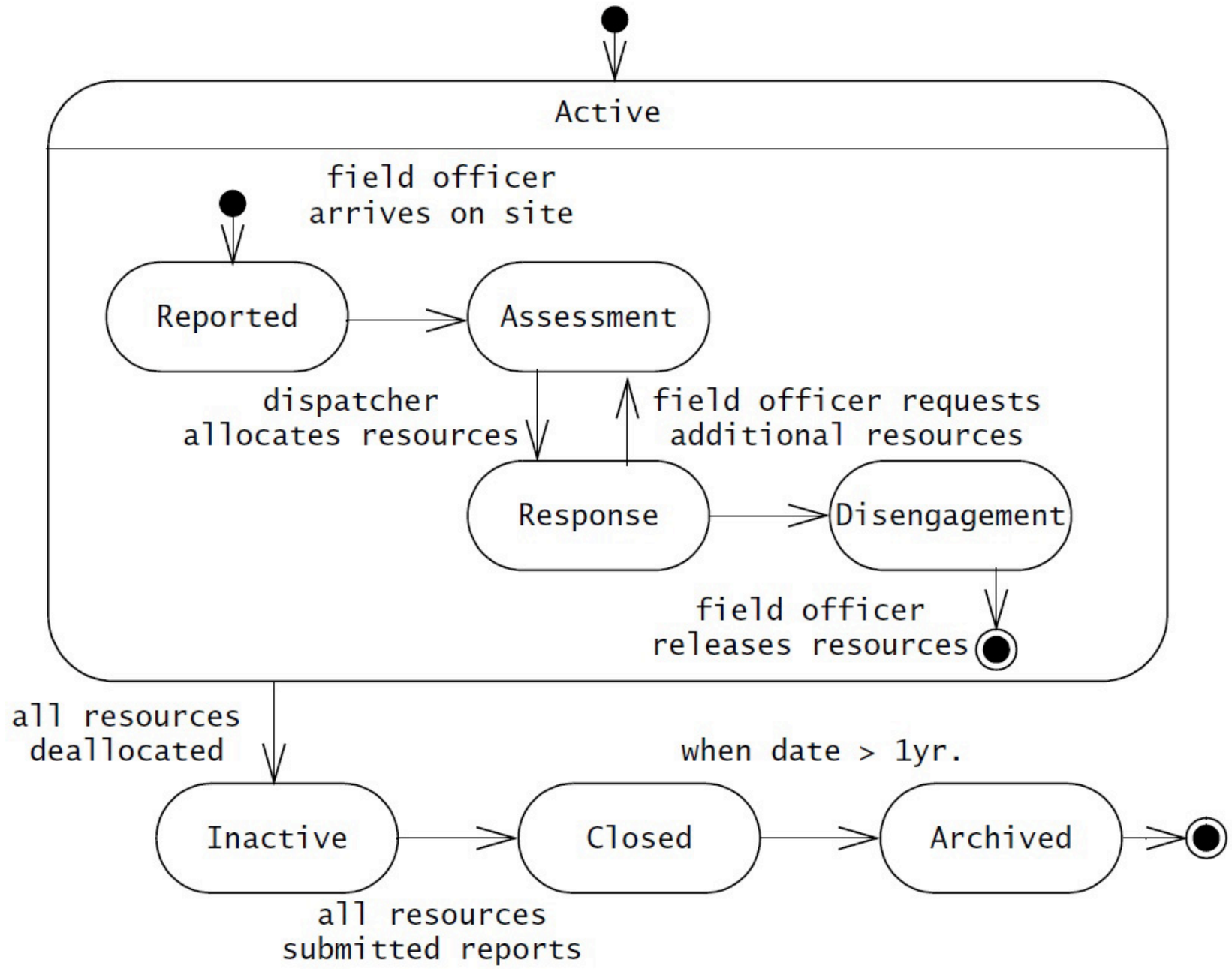
Identifying attributes



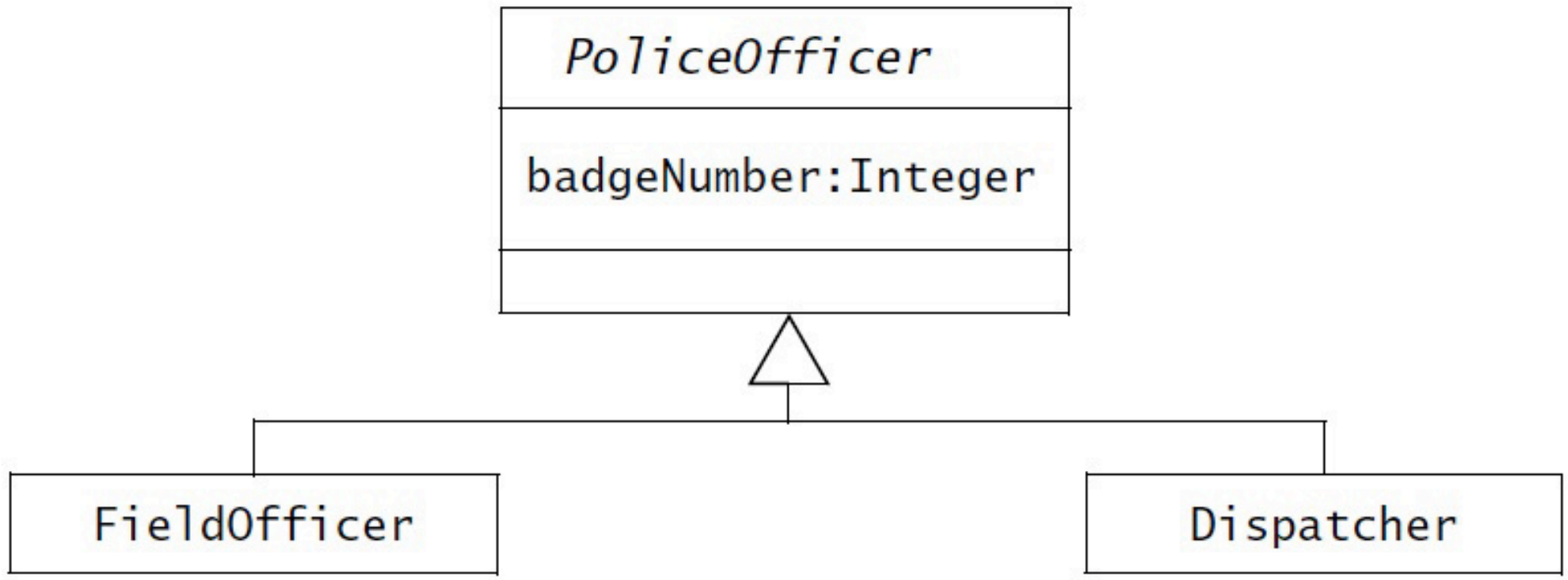
Modeling state-dependent behavior of an obejct

- Two UML diagrams types for describing dynamic models:
  - interaction diagrams describe the dynamic behavior between objects;
  - state chart diagrams describe the dynamic behavior of a single object.

State diagram

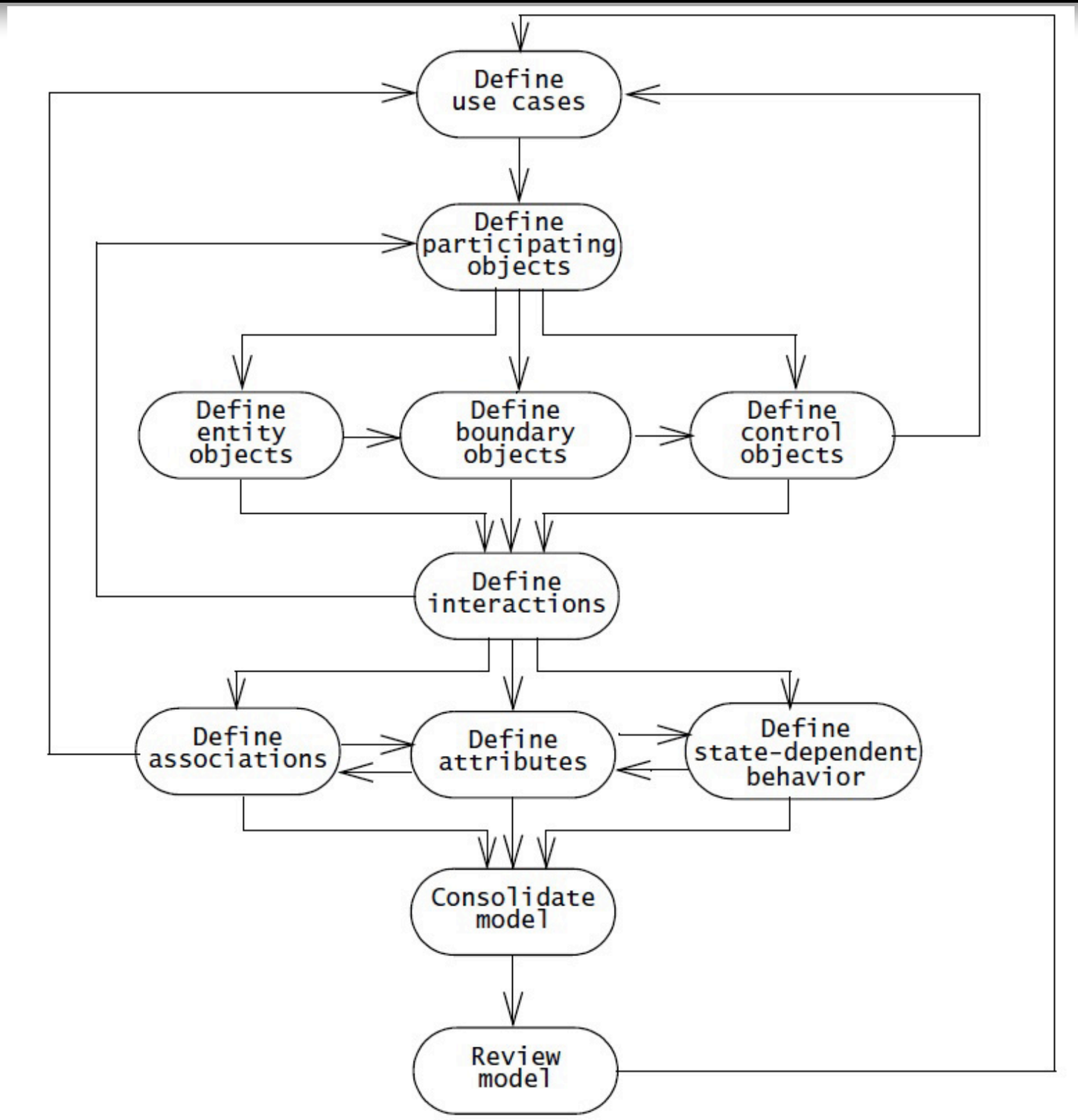


Modeling inheritance



- Inheritance is used to eliminate redundancy from the analysis model
  - if classes share attributes / behavior.

Analysis activities



Managing analysis

Requirements Analysis Document

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Scope of the system
   - 1.3 Objectives and success criteria of the project
   - 1.4 Definitions, acronyms, and abbreviations
   - 1.5 References
   - 1.6 Overview
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
     - 3.3.1 Usability
     - 3.3.2 Reliability
     - 3.3.3 Performance
     - 3.3.4 Supportability
     - 3.3.5 Implementation
     - 3.3.6 Interface
     - 3.3.7 Packaging
     - 3.3.8 Legal
   - 3.4 System models
     - 3.4.1 Scenarios
     - 3.4.2 Use case model
     - *3.4.3 Object model*
     - *3.4.4 Dynamic model*
     - 3.4.5 User interface—navigational paths and screen mock-ups
4. Glossary

Requirements Analysis Document

1. Introduction
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
   - 3.4 System models
     - 3.4.1 Scenarios
     - 3.4.2 Use case model
     - 3.4.3 Object model
       - 3.4.3.1 Data dictionary
       - 3.4.3.2 Class diagrams
     - 3.4.4 Dynamic models
     - 3.4.5 User interface—navigational paths and screen mock-ups
4. Glossary

Figure 4-16 Outline of the Requirements Analysis Document (RAD). Sections in *italics* are completed during analysis (see next chapter).

Roles in OOA

- End user
  - application domain expert, generates information/requirements.
- Client
  - define scope / priorities – often the guy with the MONEY!
- Analyst
  - application domain expert modeling the current and future systems.
- Architect
  - unifies use cases, object models, and dynamic models into one coherent description/model.
- Document Editor
  - low-level editing and document management.
- Configuration Manager
  - maintains revision history and manage changes.
- Reviewer
  - validates the RAD for correctness, completeness, consistency, and clarity.

Client sign-off

- The client has to formally accept and approved the analysis model and the RAD.
- Agree on:
  - functions and features;
  - priorities;
  - revision process;
  - acceptance criteria;
  - schedule;
  - budget.
- This is REALLY, REALLY IMPORTANT – and often forgotten, which again causes a lot of problems later in the process!!!!

Concluding

Key Points I

- Static modeling
  - Identifying interactor (control) objects
  - Identifying entity objects
  - Identifying boundary objects

- Dynamic modeling
  - Mapping use cases to objects using sequence diagrams
  - Modeling interactions among objects with CRC cards

- Linking and refining
  - Identifying associations
  - Identifying aggregates
  - Identifying attributes
  - Modeling state-dependent behavior of individual objects
  - Modeling inheritance relationships

- Reviewing the Analysis Model

Outline

- Literature
  - [OOSE] ch. 5
- Topics covered:
  - Object-Oriented Analysis
    - Analysis model
    - UML diagrams for OOA
    - OOA activities
  - Managing Analysis
    - Documentation (RAD)
    - Client sign-off