Section 2.3
Analysis

1. Overview
2. Concepts
3. Activities
4. ARENA case study

2.3.1 Overview

- Learning outcomes
  - identify high-level objects and categorize them
    - entity, control, and boundary objects
  - construct the corresponding *object model* with:
    - associations (inheritance, aggregation)
    - directionality, multiplicity
    - UML class diagrams
  - construct the corresponding *dynamic model*
    - based on functional requirements
    - UML sequence diagrams, state machine diagrams, activity diagrams

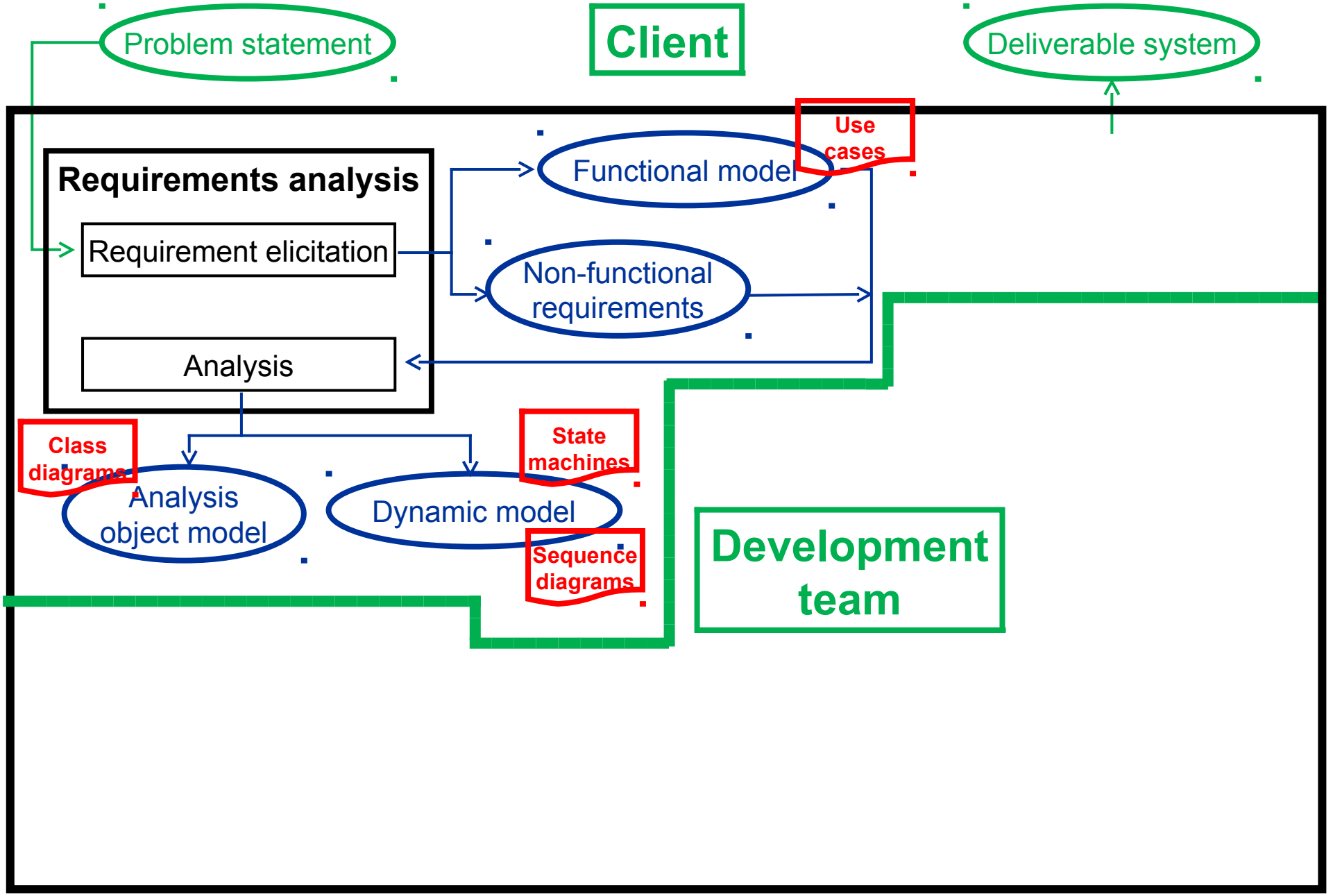
Overview (cont.)

- Characteristics of analysis phase:
  - highly iterative and incremental
  - output is not necessarily for client consumption

- Focus on:
  - structuring and formalizing the requirements
  - describing the application domain
  - representing the system from the user's point of view

- The *analysis model* is made up of all three:
  - functional model (use cases, scenarios)
  - analysis object model (class diagrams)
  - dynamic model (state machine and sequence diagrams)

Analysis Work Products



2.3.2 Analysis Concepts

- Analysis object model
- Generalization and specialization
- Dynamic model

Analysis Object Model

- What is the analysis object model?
  - the focus is on:
    - high-level concepts manipulated by the system
    - their properties
    - their relationships
- How is it represented?
  - UML class diagrams

Analysis Object Model (cont.)

- Categories of objects
  - entity objects
    - persistent information tracked by the system
  - boundary objects
    - interactions between actors and the system
  - control objects
    - in charge of realizing use cases

Analysis Object Model (cont.)

- Characteristics
  - encapsulation
  - we can separate:
    - interface (boundary objects)
    - functionality (entity and control objects)

- Why separate the different categories?
  - better for maintenance phase
  - facilitates modifiability, extensibility

- Naming conventions
  - class name should indicate the category

Generalization and Specialization

- Generalization
  - identifies abstract concepts from more specific ones
- Specialization
  - identifies more specific concepts from more general ones
- Generalization + Specialization == Inheritance

Dynamic Model

- What is the dynamic model?
  - it captures the system behaviour from the external point of view
- How is it represented?
  - UML sequence diagrams
    - shows interactions among *a set of objects* for *one use case*
  - UML state machine diagram
    - shows the behaviour of *one object*

2.3.3 Analysis Activities

- Identifying entity, boundary, control objects
- Identifying associations, aggregates, attributes
- Modelling inheritance relationships
- Mapping use cases to sequence diagrams
- Modelling state-dependent behaviour
- Reviewing the analysis model

Identifying Entity Objects

- Starting point:
  - look at every use case
  - find participating objects in use case

- Heuristics for potential mappings

Part of Speech	Model component
proper noun	instance
common noun	class
doing verb	operation
being verb	inheritance
having verb	aggregation
modal verb	constraint
adjective	attribute

Identifying Entity Objects (cont.)

- Strategy
  - examine every use case
  - identify:
    - recurring nouns
    - real-world entities and activities
    - data sources and data sinks
  - start with names used in the application domain

Identifying Entity Objects (cont.)

- Sample use case

Use case name	**`ViewClassList`**
Participating actors	Initiated by **`Instructor`** Communicates with **`RegistrarSystem`**
Flow of events	1. The **`Instructor`** selects the View Class List option. 2. The **`RegistrarSystem`** displays a list of all **courses** taught by the **`Instructor`**. 3. The **`Instructor`** selects the desired course. 4. The **`RegistrarSystem`** displays a list of **`Student`**s registered in the selected course.
Entry condition	• The **`Instructor`** must be logged in to UniCentral.
Exit condition	
Quality requirements	• The **`Instructor`** can keep working while his/her request for the class list is being processed. When the requested information becomes available, it should be displayed in another window. • Response time between the **`Instructor`** selecting the View Class List option and receiving the information should be no more than 20 seconds.
Traceability	• FR-17, NFR-23, NFR-28

Identifying Entity Objects (cont.)

- What is a *data dictionary*?
  - definition of objects, with attributes and associations
  - contains additional information not found in class diagrams

- Sample data dictionary

Entity Object	Attributes and Associations	Definition
`Course`	• instructor • students	Learning session offered by a department, at a given time. Each course has a list of students registered for the course, and an instructor assigned to teach it.
`Instructor`	• courses taught	Teacher of a given set of courses.
`RegistrarSystem`		An instance of the remote Registrar System.
`Student`	• courses	An individual who attends a university for learning. Students register for courses.

Identifying Boundary Objects

- What are boundary objects?
  - they represent the high-level system interface with the actors
- Purpose
  - to collect information from the actors
  - to translate it into the format required by control and entity objects
- Examples: GUI buttons, forms
  - not menu items or scroll bars
  - don’t get bogged down in the details of which widget

Identifying Boundary Objects (cont.)

- Strategy:
  - identify:
    - UI controls needed to initiate a use case
    - the forms required to gather information
    - the messages and notices used by the system
  - all interactions with actors must use boundary objects
  - use end user terminology from the application domain

Boundary Object	Definition
`ClassListNotice`	List of students displayed to the instructor.
`ViewClassListOption`	UI object that triggers an instructor's request to view the list of students registered in a given course.
`ViewReply`	List of students returned by RegistrarSystem.
`ViewRequest`	Request for list of students sent to RegistrarSystem.

Identifying Control Objects

- What are control objects?
  - usually manage the control flow of one use case
  - created at the beginning of the use case, destroyed at the end
  - a use case may need multiple control objects, one per actor

- Purpose
  - to coordinate boundary and entity objects
    - collect information from boundary objects
    - communicate it to the entity and control objects
  - to model the control flow of a use case

Identifying Control Objects (cont.)

- Strategy:
  - identify:
    - one control object per use case
    - one control object per actor in a use case
  - life span of a control object is the extent of the use case

Control Object	Definition
`RegistrarSystemControl`	Manages communications with RegistrarSystem.
`ViewClassListControl`	Initiates the View Course List control flow, contacts the RegistrarSystem to obtain the class list information, and displays it to the instructor.

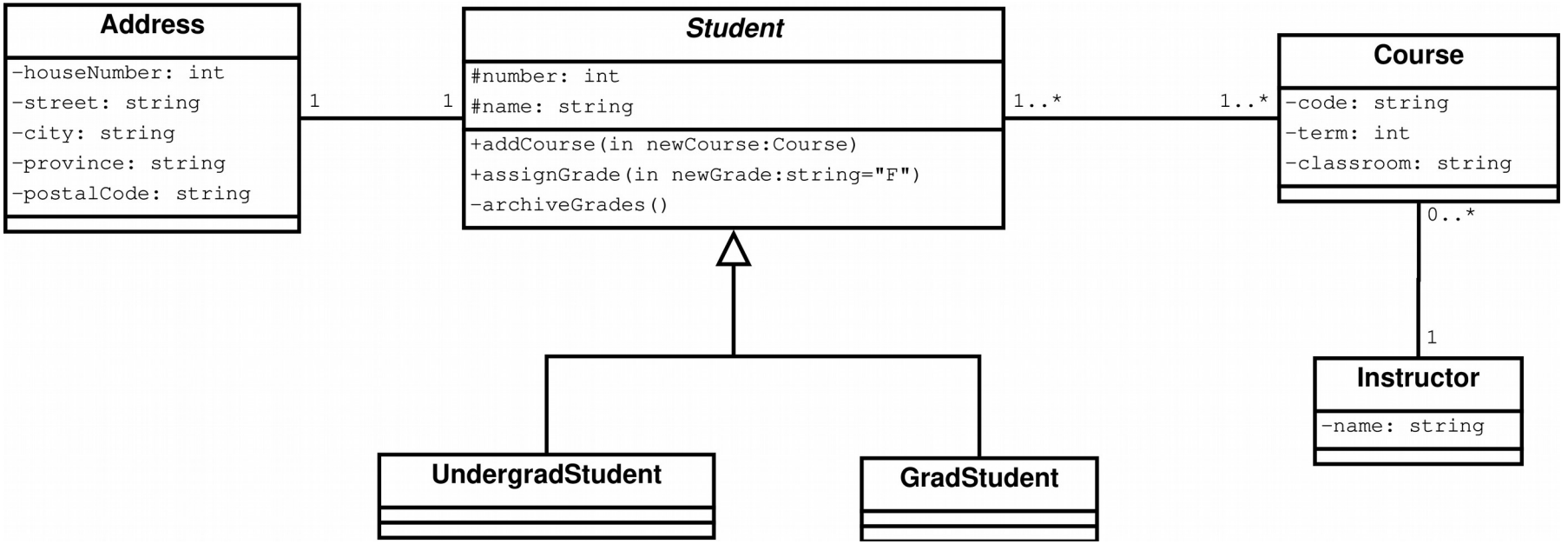
Identifying Associations

- What are associations?
  - relationships between objects
  - they describe the inter-dependencies of objects
  - they enable the discovery of special cases

- Properties of associations
  - name
    - optional, may not be unique
  - role at each end
    - optional, identifies the purpose of each class in the association
  - multiplicity
    - identifies the number of instances
    - may be a range

Identifying Associations (cont.)

- Strategy:
  - examine verb phrases
  - give names and roles in associations
  - identify attributes from qualifiers
  - minimize the number of associations
    - remove redundancy
    - eliminate derived associations



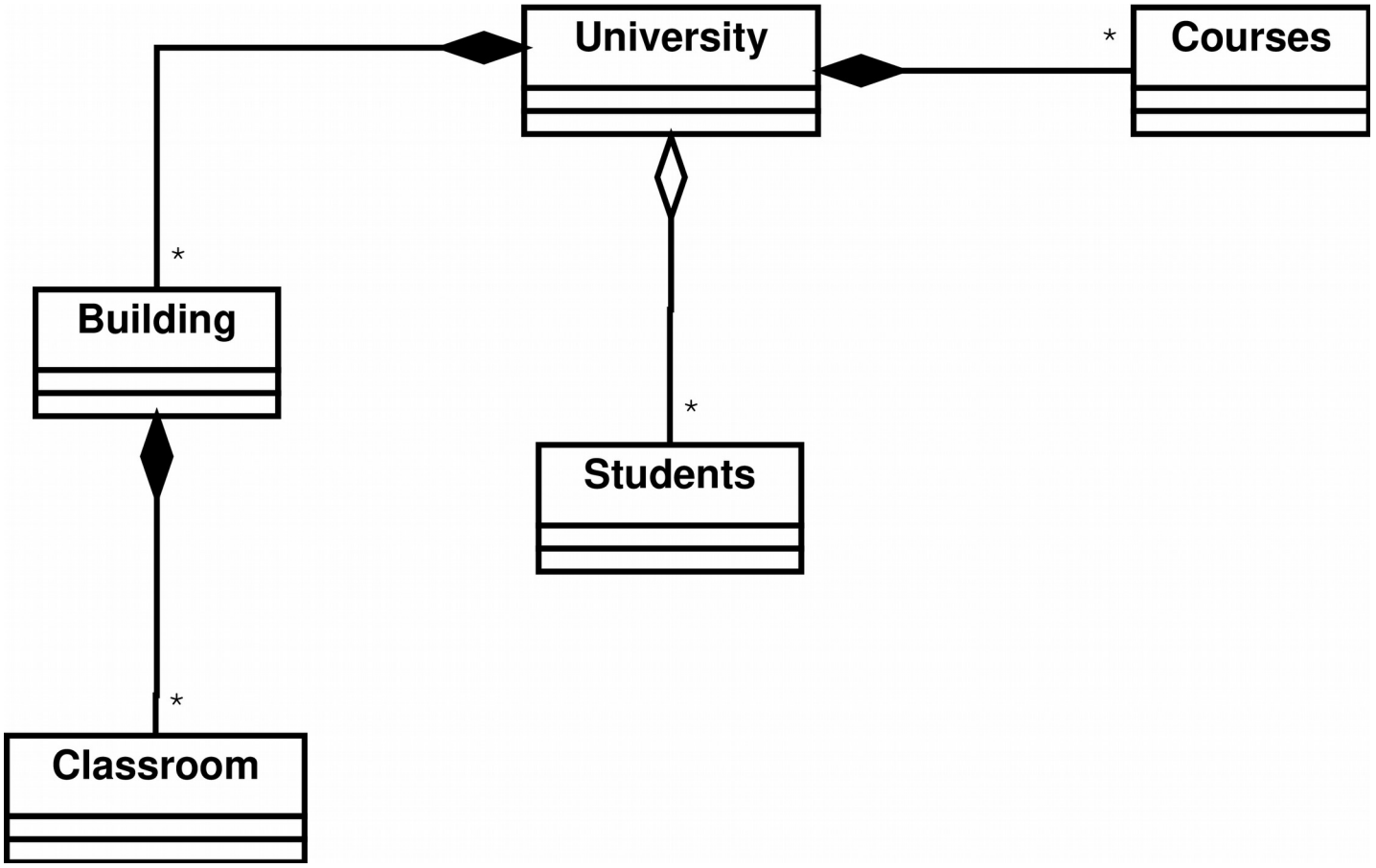
Identifying Aggregates

- What are aggregates?
  - “has-a” relationship between objects

- Two types of aggregation
  - composition
    - existence of part depends on existence of whole
    - part cannot belong to another whole
    - part cannot exist on its own
    - UML: 
  - shared aggregation
    - both objects can exist independently
    - UML: 

Identifying Aggregates (cont.)

- Example of aggregates



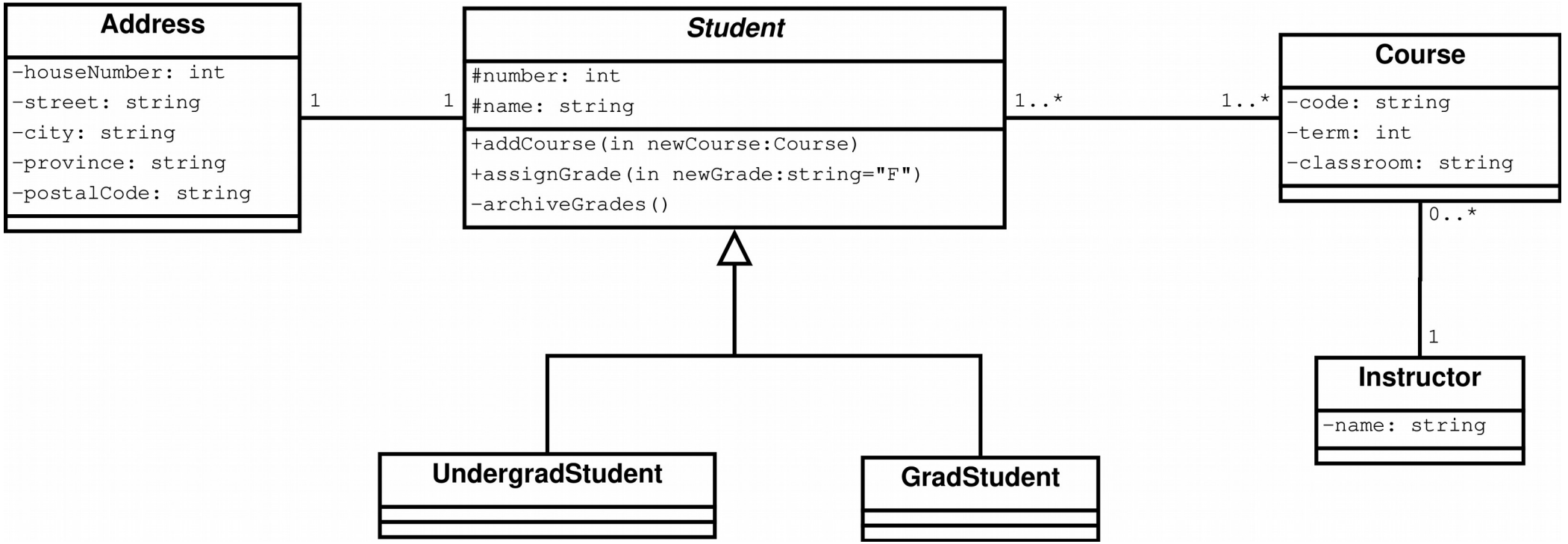
Identifying Attributes

- What are attributes?
  - properties of individual objects
  - should be relevant to the system
  - other objects are **never** attributes
    - they are represented as associations
- Characteristics of attributes
  - name, data type
- Strategy
  - examine possessive phrases in use cases
  - consider the stored state of entity objects

Modelling Inheritance Relationships

- Purpose
  - generalize class attributes and behaviour

- UML class diagrams
  - UML: △



Mapping Use Cases to Sequence Diagrams

- Purpose of mapping
  - to uncover new objects
  - to identify new object operations

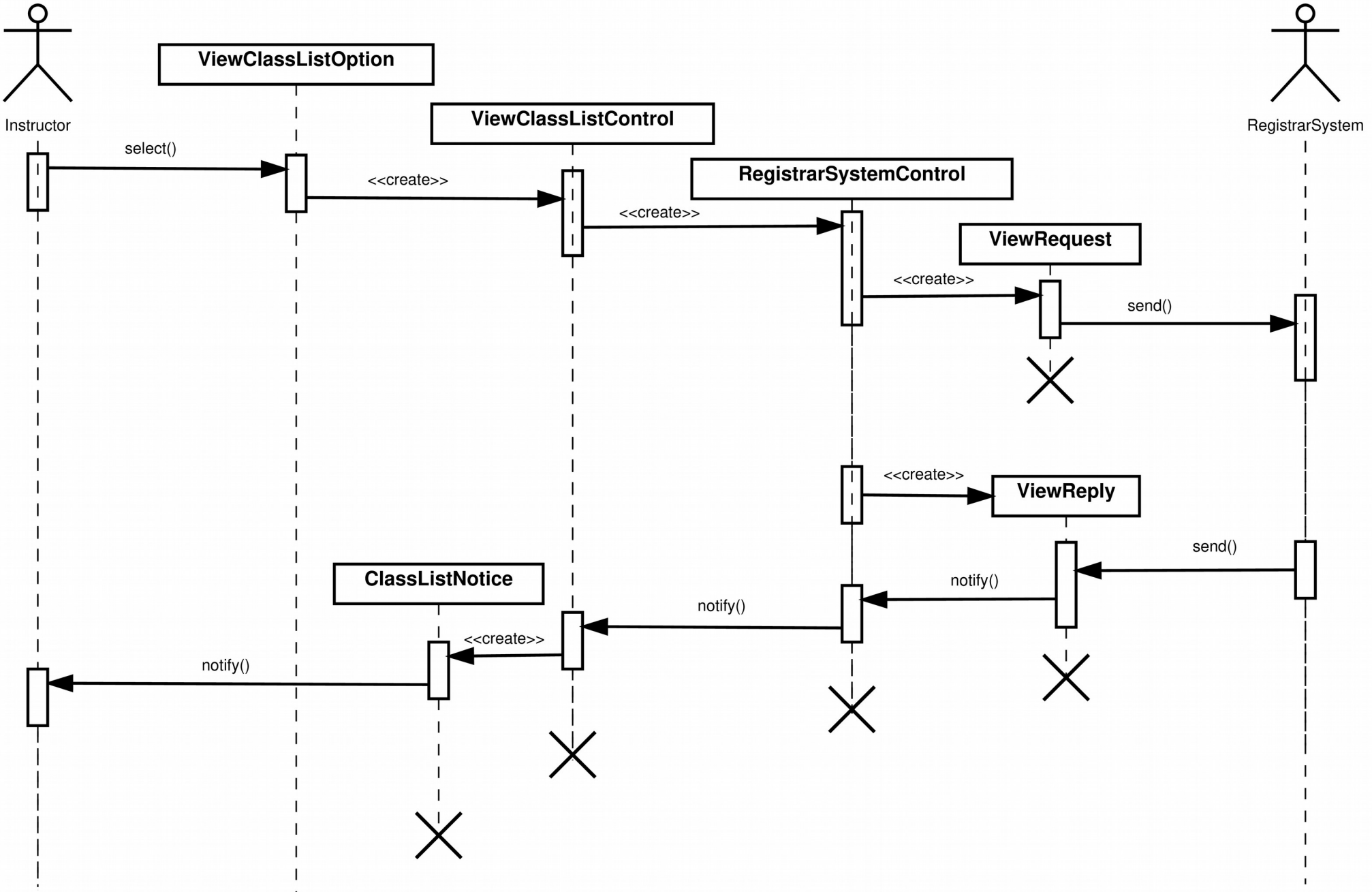
- Characteristics of sequence diagrams
  - each is linked to one use case
  - it ties a use case to its participating objects
  - it shows how use case behaviour is distributed among objects
  - it models the interactions between objects
  - the emphasis is on high-level behaviour, not on implementation
    - we are still **very** far from the code

Mapping Use Cases to Sequence Diagrams (cont.)

- Strategy:
  - first column is the initiating actor
  - second column is the boundary object used by initiating actor
  - third column is the control object managing the use case
  - principal control object is created by the initiating boundary obj.
  - other control objects may be created by principal control obj.
  - further boundary objects are created by the control objects
  - entity objects are accessed by the control or boundary objects

- Entity objects **never** access boundary or control objects
  - why?

Use Cases to Sequence Diagrams



Modelling State-Dependent Behaviour

- Purpose
  - to capture the behaviour of a single object
  - to identify new behaviour

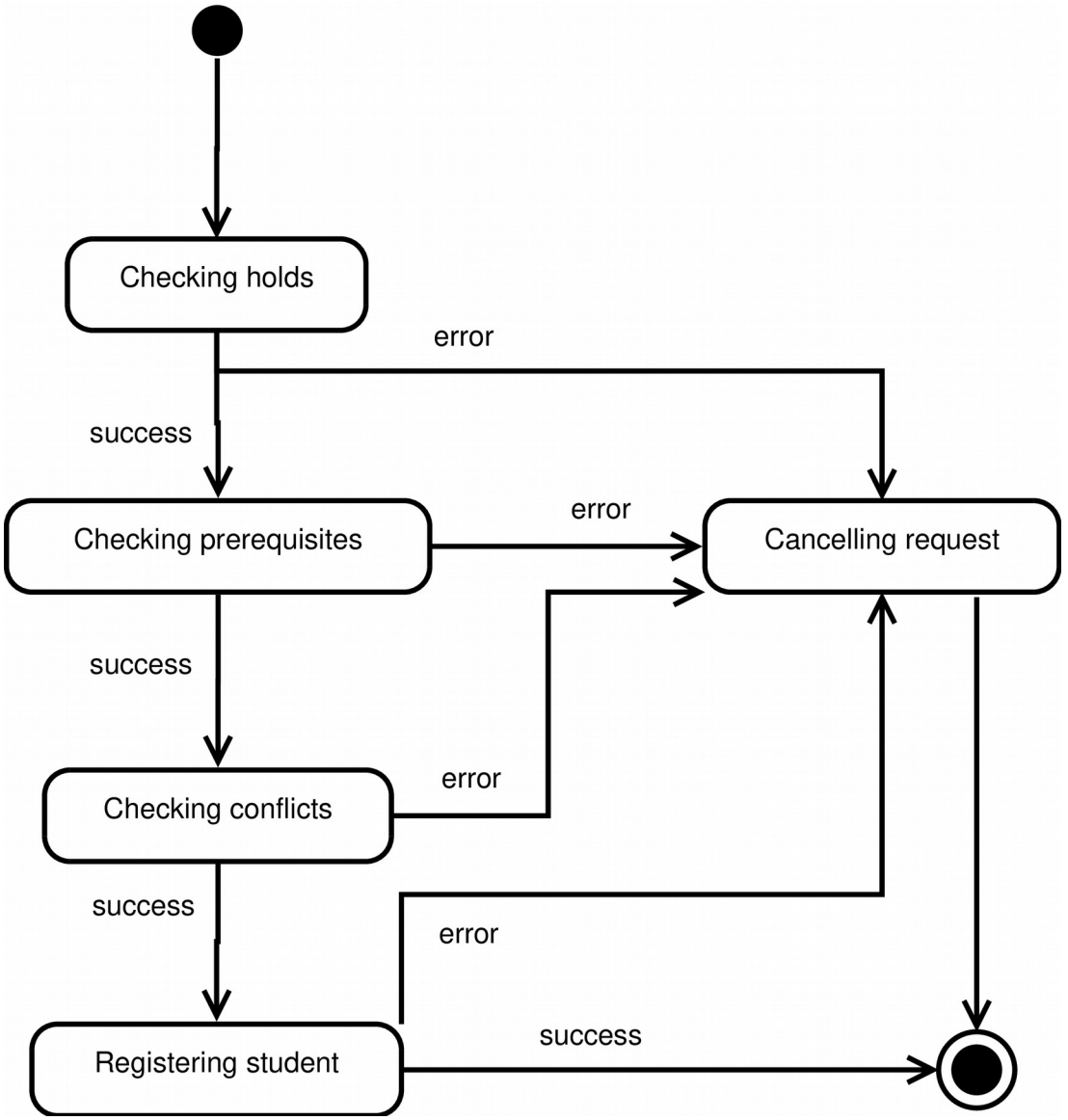
- State machine diagrams
  - when to use:
    - not necessary for every object
    - only for long-lived objects with state-dependent behaviour
  - what objects:
    - almost always for control objects
    - sometimes for entity objects
    - almost never for boundary objects

Modelling State-Dependent Behaviour (cont.)

- Strategy:
  - for each entity, boundary, control object:
    - think about whether the state needs to be stored
    - multiple and complex changes in state should be captured

Modelling State-Dependent Behaviour (cont.)

- Example: RegisterCourseControl object



Modelling State-Dependent Behaviour (cont.)

- Example: *cuLearn* assignment submission

Reviewing the Analysis Model

- This activity serves to ensure model:
  - correctness
  - completeness
  - consistency
  - verifiability / realism

Reviewing the Analysis Model (cont.)

- Correctness:
  - glossary of entity objects is understandable
  - abstract classes correspond to user-level concepts
  - descriptions match user's definitions
  - entity and boundary objects are named with noun phrases
  - use cases and control objects are named with verb phrases
  - all error cases are described and handled

Reviewing the Analysis Model (cont.)

- Completeness
  - each object:
    - is needed by a use case
    - indicates which use case(s) create/modify/destroy it
    - is accessed by a boundary object
  - each attribute:
    - indicates when it it set
    - indicates the data type
    - is a qualifier

Reviewing the Analysis Model (cont.)

- Completeness (cont.)
  - each association:
    - indicates when it is traversed
    - justifies multiplicity
    - indicates whether one-to-many and many-to-many are qualified
      - are there special cases?
  - each control object:
    - has the necessary associations to access objects in its use case

Reviewing the Analysis Model (cont.)

- Consistency
  - no multiple classes or use cases with the same name
  - similar names mean similar concepts
  - objects with similar attributes and behaviour are in the same inheritance hierarchy
- Verifiability / realism
  - prototypes have been built for novel features
  - performance and reliability requirements have been verified using prototypes on target hardware

2.3.4 ARENA Case Study

Name	AnnounceTournament
Flow of events	1. The **LeagueOwner** requests the creation of a **tournament**. 2. The system checks if the LeagueOwner has exceeded the **number of tournaments** in the **league** or in the **arena**. If not, the system presents the LeagueOwner with a form. 3. The LeagueOwner specifies a **name**, **application start and end dates** during which Players can apply to the tournament, **start and end dates** for conducting the tournament, and a **maximum number of Players**. 4. The system asks the LeagueOwner whether an exclusive sponsorship should be sought and, if yes, presents a **list of Advertisers** who expressed the desire to be **exclusive sponsors**. 5. If the LeagueOwner decides to seek an exclusive sponsor, he selects a subset of the **names** of the **proposed sponsors**. 6. The system notifies the selected sponsors about the upcoming tournament and the **flat fee** for exclusive sponsorships. 7. The system communicates their **answers** to the LeagueOwner. 8. If there are interested sponsors, the LeagueOwner selects one of them. 9. The system records the **name** of the exclusive sponsor and charges the flat fee for sponsorships to the **Advertiser's account**. From now on, all **advertisement banners** associated with the tournament are provided by the exclusive sponsor only. 10. If no sponsors were selected (either because no Advertisers were interested or the LeagueOwner did not select any), the advertisement banners are selected at random and charged to each Advertiser's account on a per unit basis. 11. Once the sponsorship issues is closed, the system prompts the LeagueOwner with a **list of groups of Players, Spectators, and Advertisers** that could be interested in the new tournament. 12. The LeagueOwner selects which groups to notify. 13. The system creates a home page in the arena for the tournament. This page is used as an entry point to the tournament (e.g., to provide interested Players with a form to apply for the tournament, and to interest Spectators into watching **matches**). 14. At the application start date, the system notifies each interested user by sending them a link to the main tournament page. The Players can then apply for the tournament with the ApplyForTournament use case until the application end date.

Figure 5-23 Applying Abbott's heuristics for identifying entity objects in the AnnounceTournament use case. The first occurrence of a noun phrase is emphasized in **bold**.

Table 5-6 Entity objects participating in the `AnnounceTournament` use case identified from noun phrases in the use case. "(?)" denote areas of uncertainty that lead to the questions in Figure 5-24.

Entity Object	Attributes & Associations	Definition
`Account`	• balance • history of charges (?) • history of payments (?)	An `Account` represents the amount currently owed by an `Advertiser`, a history of charges, and payments.
`Advertiser`	• name • leagues of interest for exclusive sponsorships (?) • sponsored tournaments • account	Actor interested in displaying advertisement banners during the `Matches`.
`Advertisement`	• associated game (?)	Image provided by an `Advertiser` for display during matches.
`Arena`	• max number of tournaments • flat fee for sponsorships (?) • leagues *(implied)* • interest groups *(implied)*	An instantiation of the `ARENA` system.
`Game`		A `Game` is a competition among a number of `Players` that is conducted according to a set of rules. In `ARENA`, the term `Game` refers to a piece of software that enforces the set of rules, tracks the progress of each `Player`, and decides the winner.
`InterestGroup`	• list of players, spectators, or advertisers • games and leagues of interests *(implied)*	`InterestGroups` are lists of users in the `ARENA` which share an interest (e.g, for a game or a league). `InterestGroups` are used as mailing lists for notifying potential actors of new events.
`League`	• max number of tournament • game	A `League` represents a community for running `Tournaments`. A `League` is associated with a specific `Game` and `TournamentStyle`. `Players` registered with the `League` accumulate points according to the `ExpertRating` of the `League`.

ARENA Case Study (cont.)

Table 5-6 *Continued.*

Entity Object	Attributes & Associations	Definition
`LeagueOwner`	• name *(implied)*	The actor creating a `League` and responsible for organizing `Tournaments` within the `League`.
`Match`	• tournament • players	A `Match` is a contest between two or more `Players` within the scope of a `Game`. The outcome of a `Match` can be a single winner and a set of losers or a tie (in which their are no winners or losers). Some `TournamentStyles` may disallow ties.
`Player`	• name *(implied)*	
`Tournament`	• name • application start date • application end date • play start date • play end date • max number of players • exclusive sponsor	A `Tournament` is a series of `Matches` among a set of `Players`. `Tournaments` end with a single winner. The way `Players` accumulate points and `Matches` are scheduled is dictated by the `League` in which the `Tournament` is organized.

ARENA Case Study (cont.)

Table 5-7 Boundary objects participating in the AnnounceTournament use case.

Boundary Object	Definition
TournamentForm	Form used by the LeagueOwner to specify the properties of a Tournament during creation or editing.
RequestSponsorshipForm	Form used by the LeagueOwner to request sponsorships from interested Advertisers.
SponsorshipRequest	Notice received by Advertisers requesting sponsorship.
SponsorshipReply	Notice received by LeagueOwner indicating whether an Advertiser wants the exclusive sponsorship of the tournament.
SelectExclusiveSponsorForm	Form used by the LeagueOwner to close the sponsorship issue.
NotifyInterestGroupsForm	Form used by the LeagueOwner to notify interested users.
InterestGroupNotice	Notice received by interested users about the creation of a new Tournament.

ARENA Case Study (cont.)

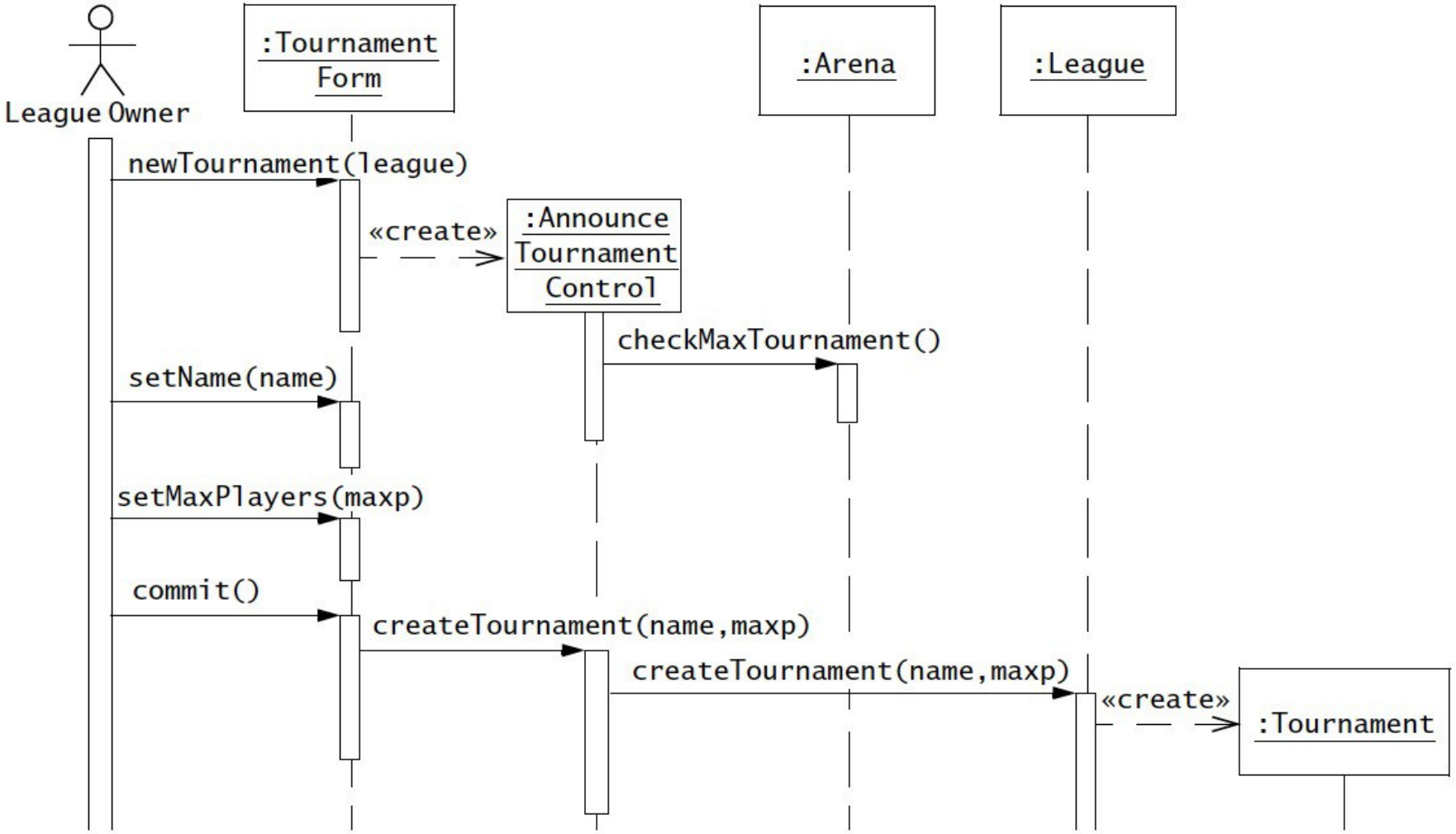


Figure 5-26 UML sequence diagram for AnnounceTournament, tournament creation workflow.

ARENA Case Study (cont.)

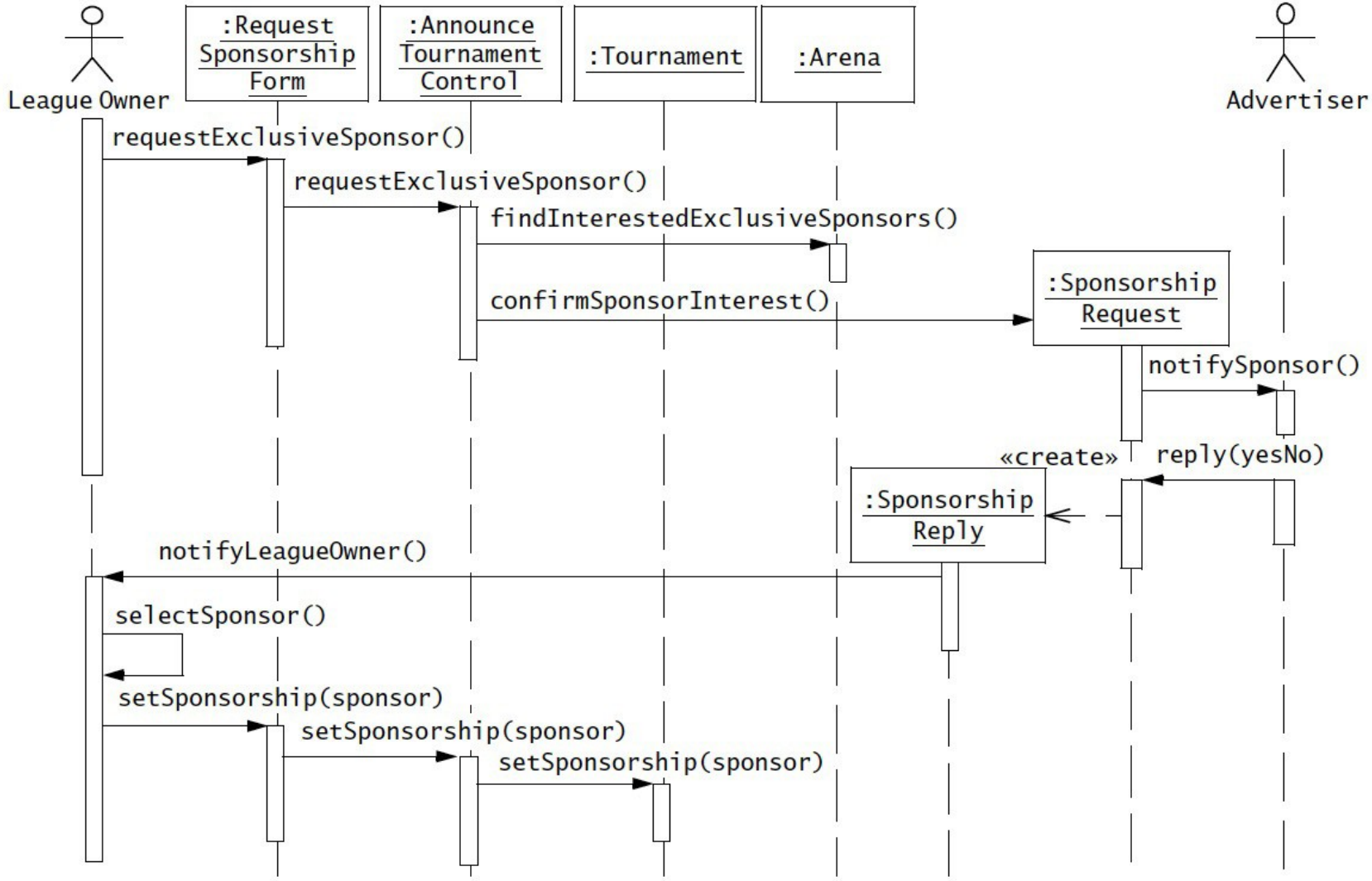


Figure 5-27 UML sequence diagram for AnnounceTournament use case, sponsorship workflow.

ARENA Case Study (cont.)

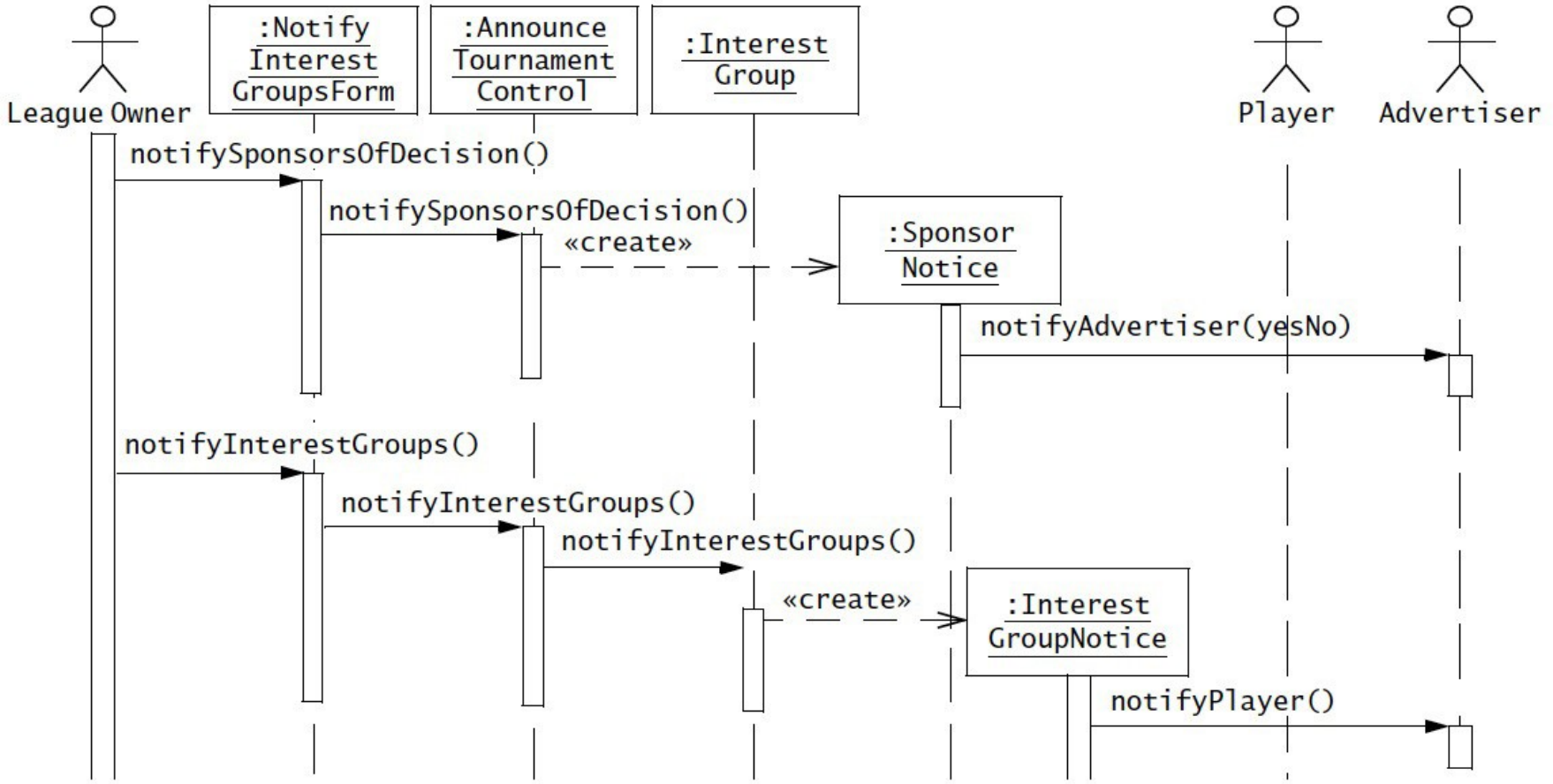


Figure 5-28 UML sequence diagram for AnnounceTournament use case, interest group workflow.

ARENA Case Study (cont.)

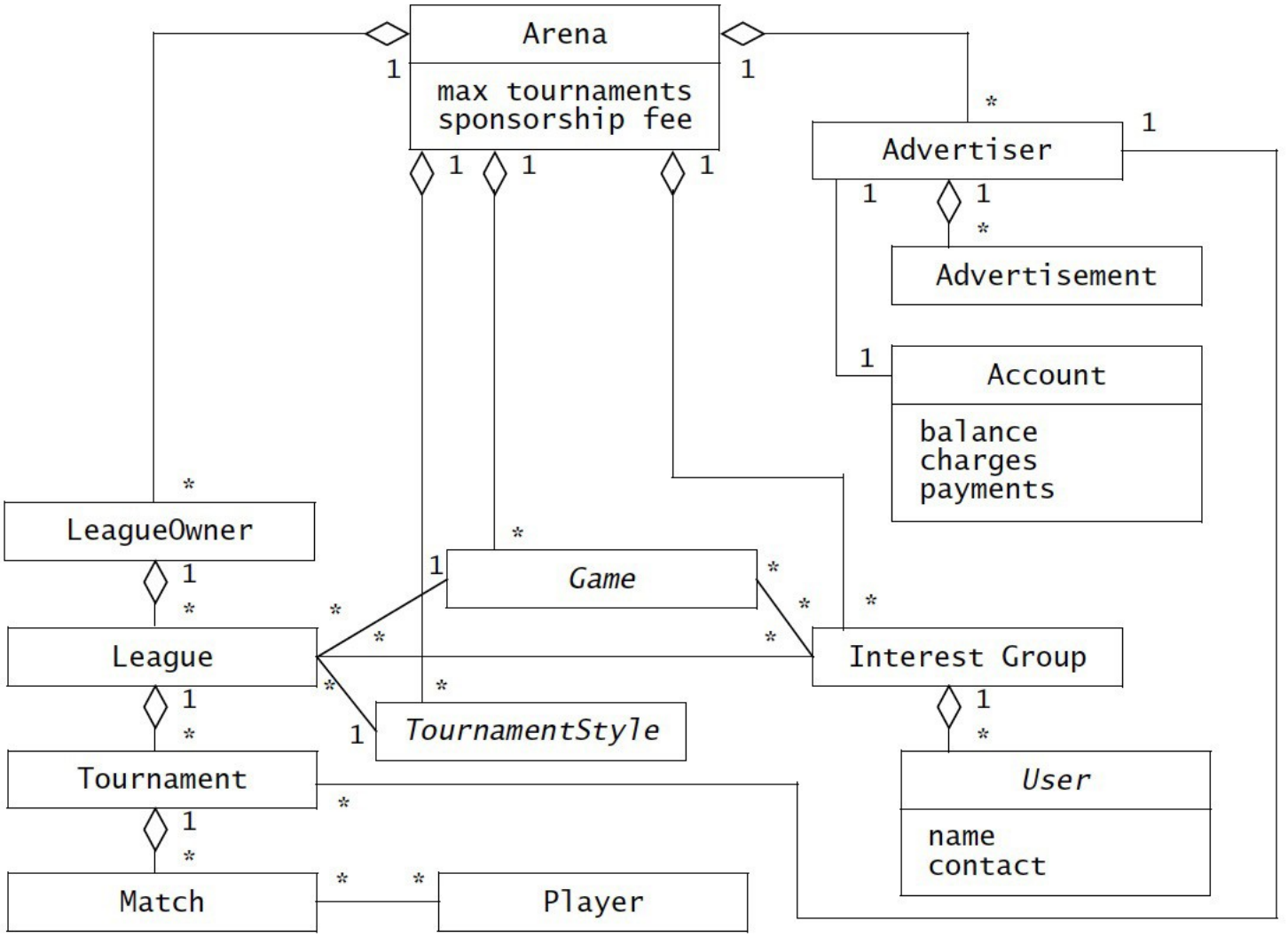


Figure 5-29 Entity objects identified after analyzing the AnnounceTournament use case.

ARENA Case Study (cont.)

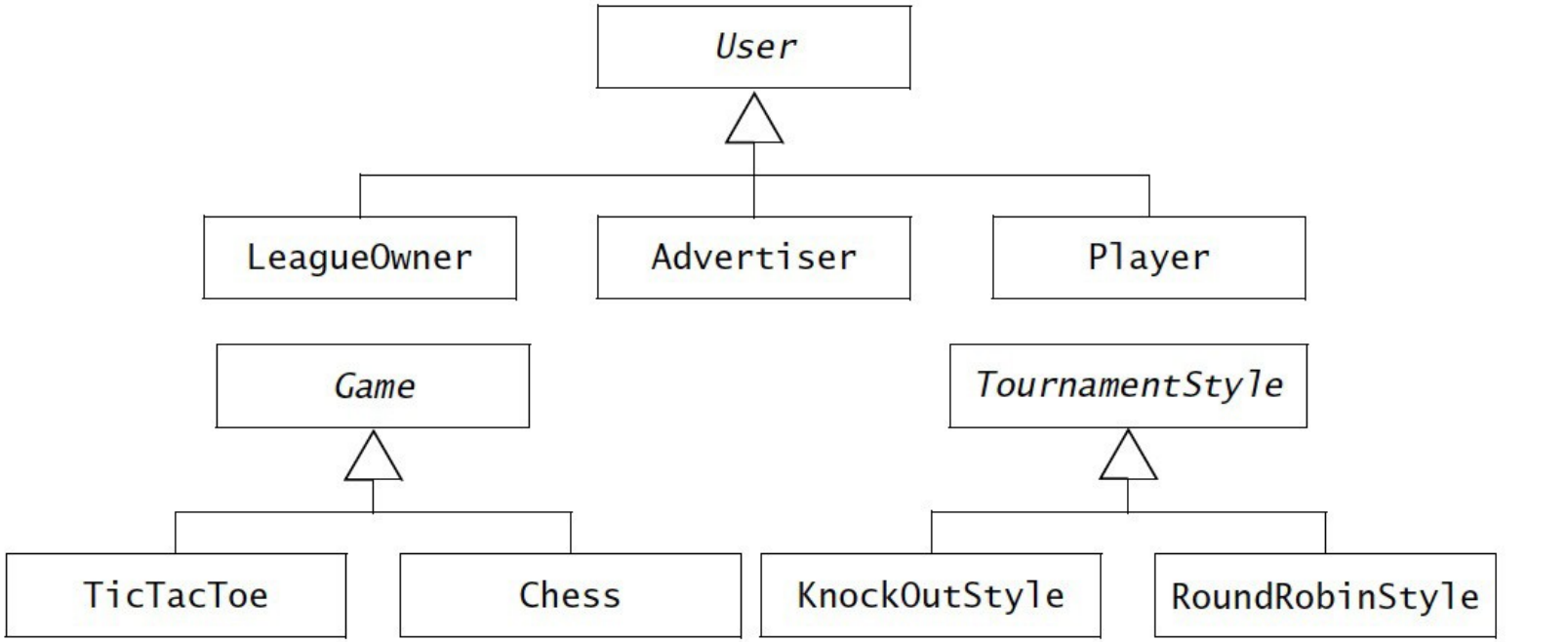


Figure 5-30 Inheritance hierarchy among entity objects of the AnnounceTournament use case.

ARENA Case Study (cont.)

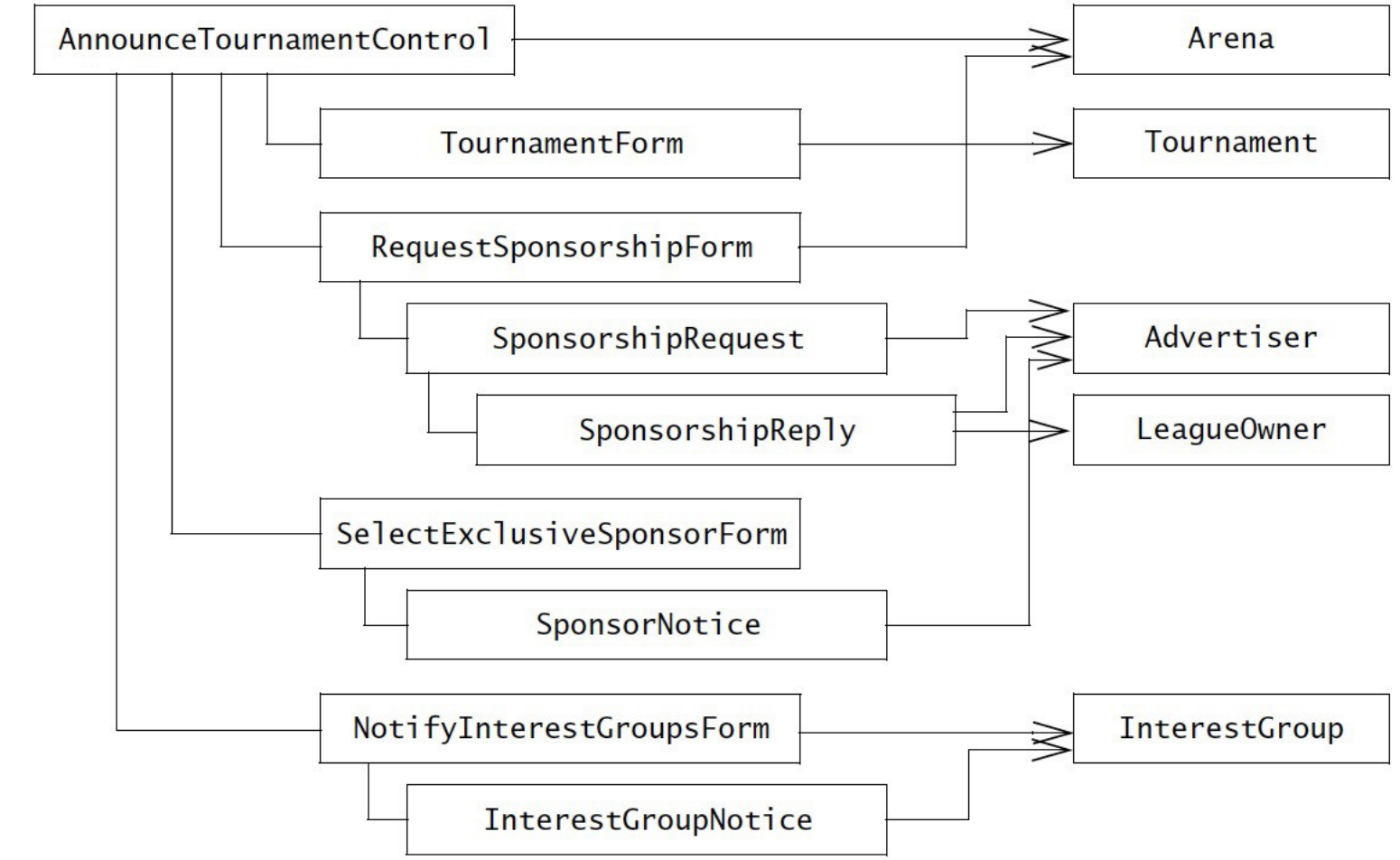


Figure 5-31 Associations among boundary, control, and selected entity objects participating in the AnnounceTournament use case.

Dorc Slayer Case Study

- Group exercise:
  - define the analysis object model
    - identify the entity, control, boundary objects
    - draw the corresponding class diagrams
  - define the dynamic model
    - draw the state machine diagrams for objects that store a state
    - draw the sequence diagrams for each use case

Analysis Recap

- What we learned:
  - identify high-level objects and categorize them
    - entity, control, and boundary objects
  - construct the corresponding *object model* with:
    - associations (inheritance, aggregation)
    - directionality, and multiplicity
    - UML class diagrams
  - construct the corresponding *dynamic model*
    - based on functional requirements
    - UML sequence diagrams, state machine diagrams, activity diagrams