ARENA Case Study

ARENA Case Study

- Problem statement
- Requirement elicitation
- Use case model
- OO Analysis Model

Initial Problem Statement

1. Problem

- The popularity of the Internet and the WWW has enabled the creation of a variety of virtual communities:
  - groups of people sharing common interests
  - who have never met each other in person
  - can be short lived (e.g., a group of people meeting in a chat room or playing a tournament) or long lived (e.g., subscribers to a mailing list).
- Many multi-player computer games now include support for the virtual communities of players:
  - receive news about game upgrades
  - new game maps and characters
  - announce and organize matches
  - compare scores and exchange tips
  - the game company takes advantage of this infrastructure to generate revenue or to advertise its products.

Initial Problem Statement (2)

Problem (cont.)

- Currently, however, each game company develops such community support in each individual game.
- Each company uses a different infrastructure, different concepts, and provides different levels of support.
- This redundancy and inconsistency results in disadvantages:
  - learning curve for players when joining each new community,
  - game companies need to develop the support from scratch
  - advertisers need to contact each individual community separately
  - this solution does not provide much opportunity for cross-fertilization among different communities.

Initial Problem Statement (3)

2. Objectives

- provide an infrastructure for operating an arena, including registering new games and players,
- organizing tournaments, and keeping track of the players scores
- provide a framework for league owners to customize the number and sequence of matches and the accumulation of expert rating points.
- provide a framework for game developers for developing new games, or for adapting existing games into the ARENA framework.
- provide an infrastructure for advertisers.

Initial Problem Statement (4)

3. Functional requirements

ARENA supports five types of users:

- The *operator* should be able to define new games, new tournament styles (e.g., knock-out tournaments, championships, best of series), define new expert rating formulas, and manage users.
- *League owners* should be able to define a new league, organize and announce new tournaments within a league, conduct a tournament, and declare a winner.
- *Players* should be able to register in an arena, apply for a league, play the matches that are assigned to the player, or drop out of the tournament.
- *Spectators* should be able to monitor any match in progress and check scores and statistics of past matches and players. Spectators do not need to register in an arena.
- The *advertiser* should be able to upload new advertisements, select an advertisement scheme (e.g., tournament sponsor, league sponsor), check balance due, and cancel advertisements.

Initial Problem Statement (5)

4. Non-functional requirements

- *Low operating cost.* The operator must be able to install and administer an arena without purchasing additional software components and without the help of a full-time system administrator.
- *Extensibility.* The operator must be able to add new games, new tournament styles, and new expert rating formulas. Such additions may require the system to be temporarily shut down and new modules (e.g., Java classes) to be added to the system. However, no modifications of the existing system should be required.
- *Scalability.* The system must support the kick-off of many parallel tournaments (e.g., 10), each involving up to 64 players and several hundreds of simultaneous spectators.
- *Low-bandwidth network*. Players should be able to play matches via a 56K analog modem or faster.

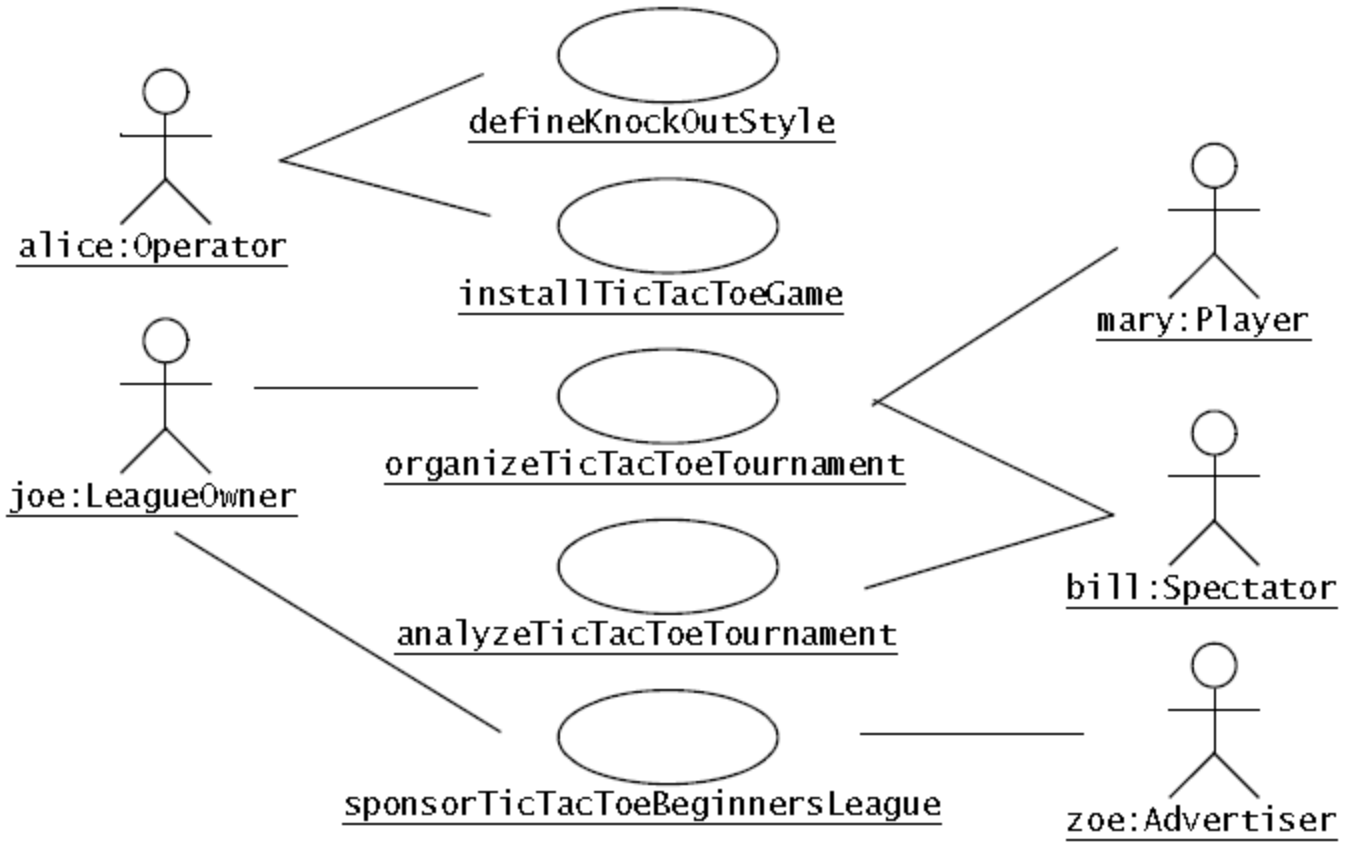
Initial Problem Statement (6)

5. Target environment

- All users should be able to access any arena with a web browser supporting cookies, Javascript, and Java applets.
- Administration functions (e.g., adding new games, tournament styles, and users) used by the operator should not be available through the web.
- ARENA should run on any Unix operating system (e.g., MacOS X, Linux, Solaris).

Identifying Actors and high-level Use Cases

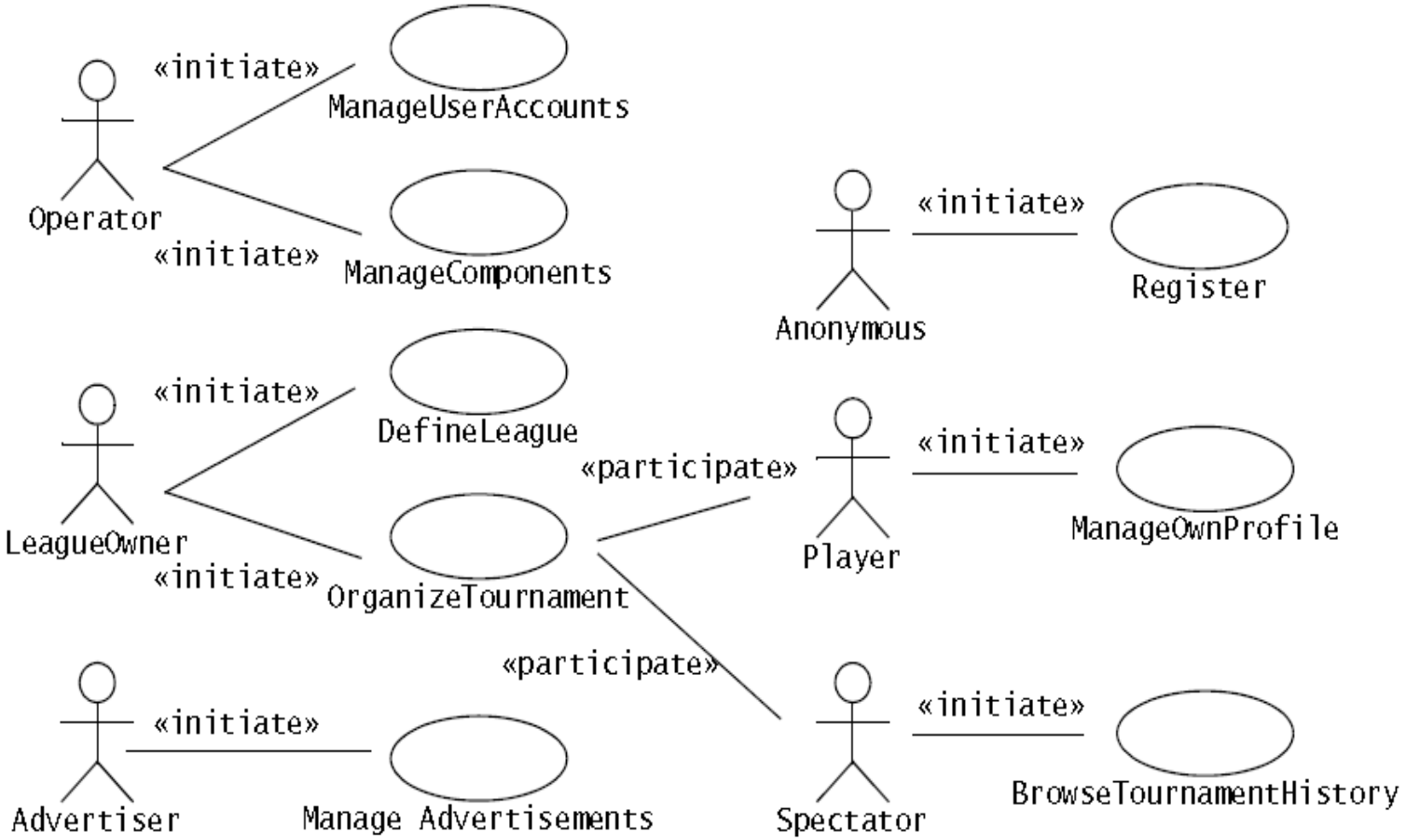
- Five actors are identified from the problem statement:
  - *Operator, LeagueOwner, Player, Spectator*, and *Advertiser*
- High level use cases identified by looking at a narrow vertical slice of the system: TicTacToeTournament (described in text and use-case diagram)



Glossary – define problem domain concepts

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. For example, tic tac toe and chess are Games.
Match	A Match is a contest between two or more Players following the rules 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 Games may disallow ties.
Tournament	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.
League	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 defined in the League. For example, a novice chess League has a different ExpertRating formula than an expert League.
TournamentStyle	The TournamentStyle defines the number of Matches and their sequence for a given set of Players. For example, Players face all other Players in the Tournament exactly once in a round robin TournamentStyle.

Refine high-level Use Cases



Brief description of Use Cases (1)

Register	Anonymous users register with an Arena for a Player or a League-Owner account. User accounts are required before applying for a tournament or organizing a league. Spectators do not need accounts.
ManageUserAccounts	The Operator accepts registrations from LeagueOwners and for Players, cancels existing accounts, and interacts with users about extending their accounts.
ManageComponents	The Operator installs new games and defines new tournament styles (generalizes defineKnockOutStyle and installTicTacToeGame).
DefineLeague	The LeagueOwner defines a new league (generalizes the first steps of the scenario organizeTicTacToeTournament).

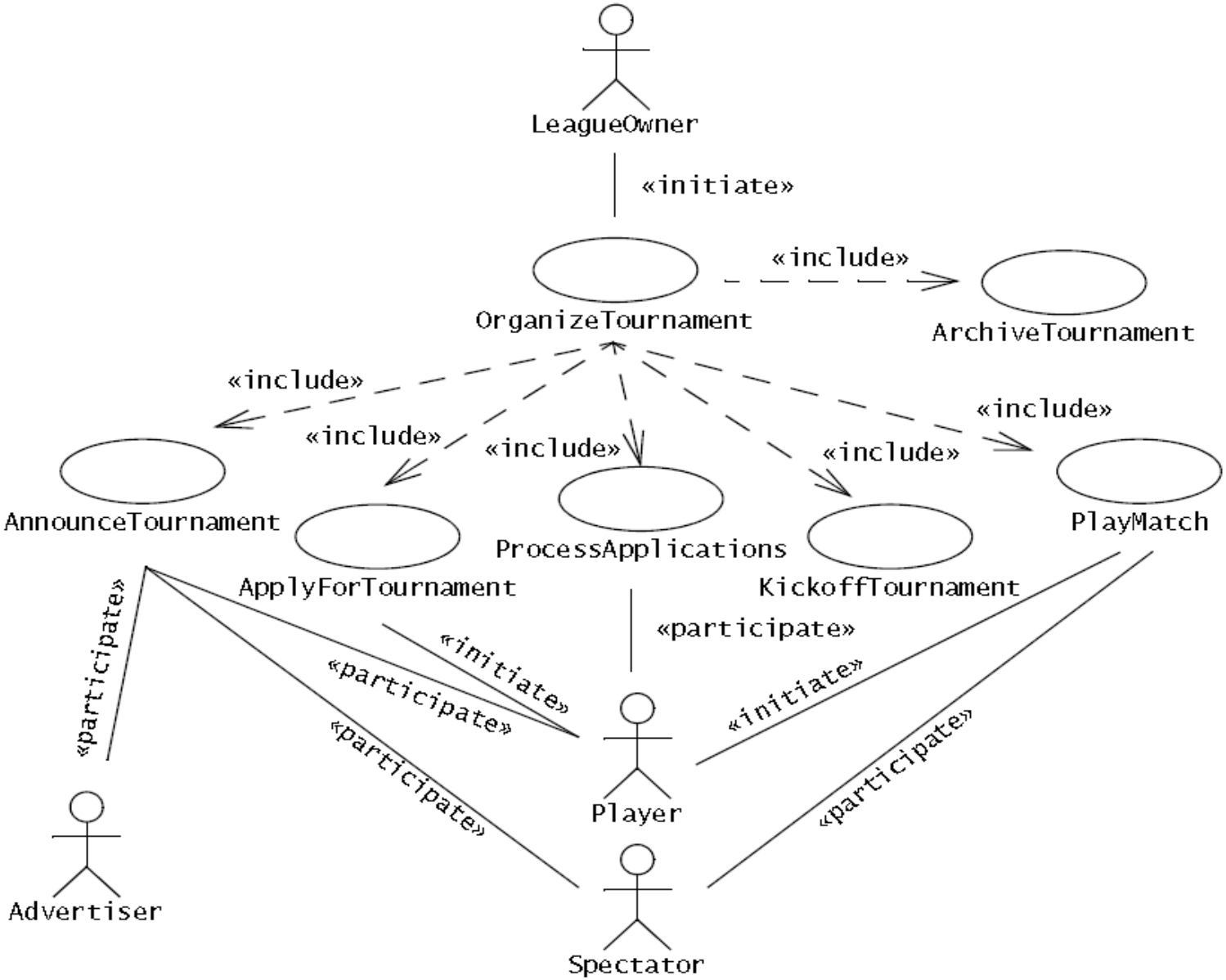
Brief description of Use Cases (2)

`OrganizeTournament`	The `LeagueOwner` creates and announces a new tournament, accepts player applications, schedules matches, and kicks off the tournament. During the tournament, players play matches and spectators follow matches. At the end of the tournament, players are credited with points (generalizes the scenario `organizeTicTacToeTournament`).
`ManageAdvertisements`	The `Advertiser` uploads banners and sponsors league or tournaments (generalizes `sponsorTicTacToeBeginnersLeague`).
`ManageOwnProfile`	The `Players` manage their subscriptions to mailing lists and answer a marketing survey.
`BrowseTournamentHistory`	`Spectators` examine tournament statistics and player statistics, and replay matches that have already been concluded (generalizes the scenario `analyzeTicTacToeTournament`).

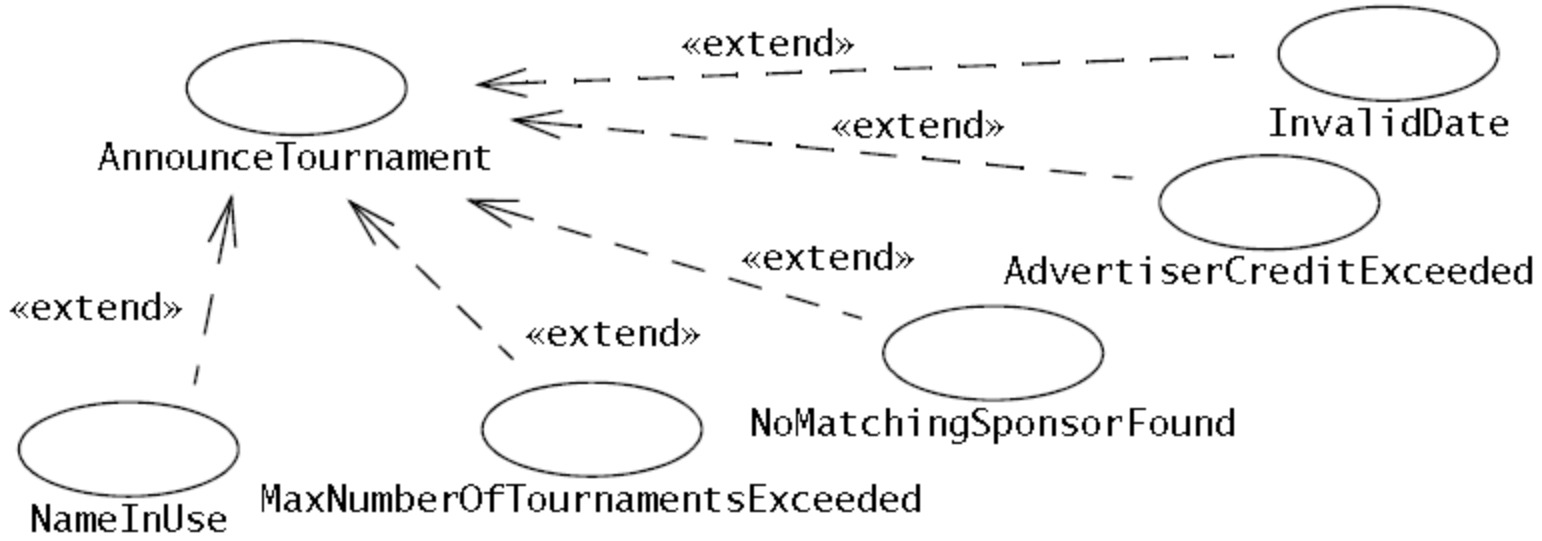
High-level use case OrganizeTournament

Use case name	`OrganizeTournament`
Participating actors	Initiated by `LeagueOwner` Communicates with `Advertiser`, `Player`, and `Spectator`
Flow of events	1. The `LeagueOwner` creates a `Tournament`, solicits sponsorships from `Advertisers`, and announces the `Tournament` (include use case `AnnounceTournament`). 2. The `Players` apply for the `Tournament` (include use case `ApplyForTournament`). 3. The `LeagueOwner` processes the `Player` applications and assigns them to matches (include use case `ProcessApplications`). 4. The `LeagueOwner` kicks off the `Tournament` (include use case `KickoffTournament`). 5. The `Players` compete in the matches as scheduled and `Spectators` view the matches (include use case `PlayMatch`). 6. The `LeagueOwner` declares the winner and archives the `Tournament` (include use case `ArchiveTournament`).
Entry condition	• The `LeagueOwner` is logged into `ARENA`.
Exit conditions	• The `LeagueOwner` archived a new tournament in the `ARENA` archive and the winner has accumulated new points in the league, OR • The `LeagueOwner` cancelled the tournament and the players' standing in the league is unchanged.

Refine OrganizeTournament



Exceptions occurring in AnnounceTournament



AdvertiserCreditExceeded	The system removes the Advertiser from the list of potential sponsors.
InvalidDate	The system informs the LeagueOwner and prompts for a new date.
MaxNumberOfTournaments Exceeded	The AnnounceTournament use case is terminated.
NameInUse	The system informs the LeagueOwner and prompts for a new name.
NoMatchingSponsorFound	The system skips the exclusive sponsor steps and chooses random advertisements from the advertisement pool.

Non-functional requirements (1)

Usability	• Spectators must be able to access games in progress without prior registration and without prior knowledge of the Game.
Reliability	• Crashes due to software bugs in game components should interrupt at most one Tournament using the Game. The other Tournaments in progress should proceed normally. • When a Tournament is interrupted because of a crash, its LeagueOwner should be able to restart the Tournament. At most, only the last move of each interrupted Match can be lost.
Performance	• The system must support the kick-off of many parallel Tournaments (e.g., 10), each involving up to 64 Players and several hundreds of simultaneous Spectators. • Players should be able to play matches via an analog modem.
Supportability	• The Operator must be able to add new Games and new TournamentStyles. Such additions may require the system to be temporarily shut down and new modules (e.g., Java classes) to be added to the system. However, no modifications of the existing system should be required.

Non-functional requirements (2)

Implementation	• All users should be able to access an `Arena` with a web browser supporting cookies, Javascript, and Java applets. Administration functions used by the operator are not available through the web. • `ARENA` should run on any Unix operating system (e.g., MacOS X, Linux, Solaris).
Operation	• An `Advertiser` should not be able to spend more advertisement money than a fixed limit agreed beforehand with the `Operator` during the registration.
Legal	• Offers to and replies from `Advertisers` require secure authentication, so that agreements can be built solely on their replies. • `Advertisers` should be able to cancel sponsorship agreements within a fixed period, as required by local laws.

Lessons Learned during Requirements Elicitation

- Requirements elicitation involves constant switching between perspectives
  - high-level vs. detailed,
  - client vs. developer,
  - activity vs. entity.
- Requirements elicitation requires a substantial involvement from the client.
- Developers should not assume that they know what the client wants.
- Eliciting non-functional requirements forces stakeholders to make and document trade-offs.

OO Analysis Phase

Develop the part of the analysis object model relevant to the AnnounceTournament use case of ARENA.

- Start by identifying entity objects using Abbott's heuristics
- Identify boundary and control objects
- Use sequence diagrams to find:
  - additional associations
  - objects
  - attributes.
- Finally, consolidate the object model and represented it in a series of class diagrams.

Entity objects (using Abbott's heuristics)

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.

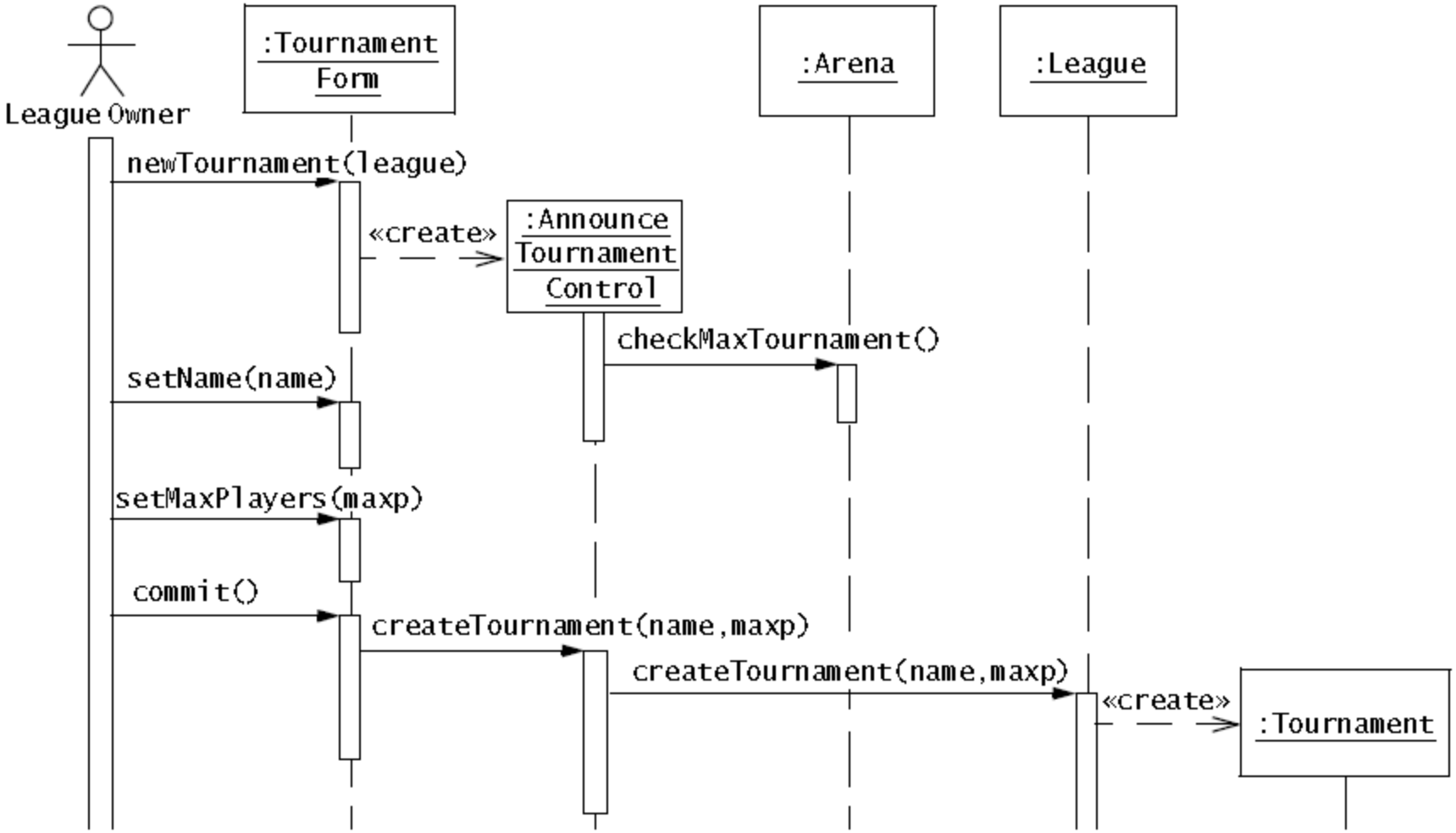
continue…

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.
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.

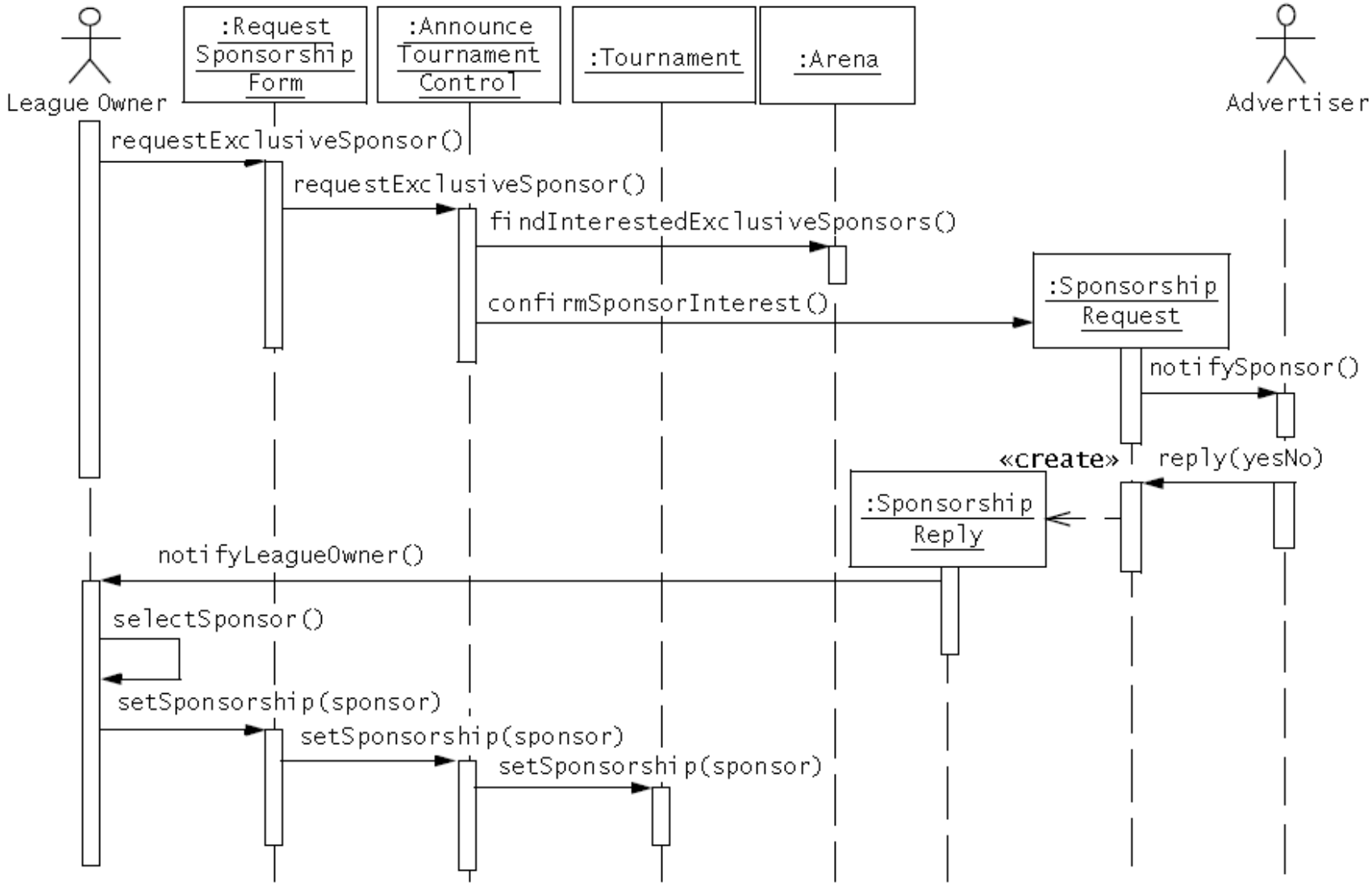
Boundary Objects in Announce Tournament

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.

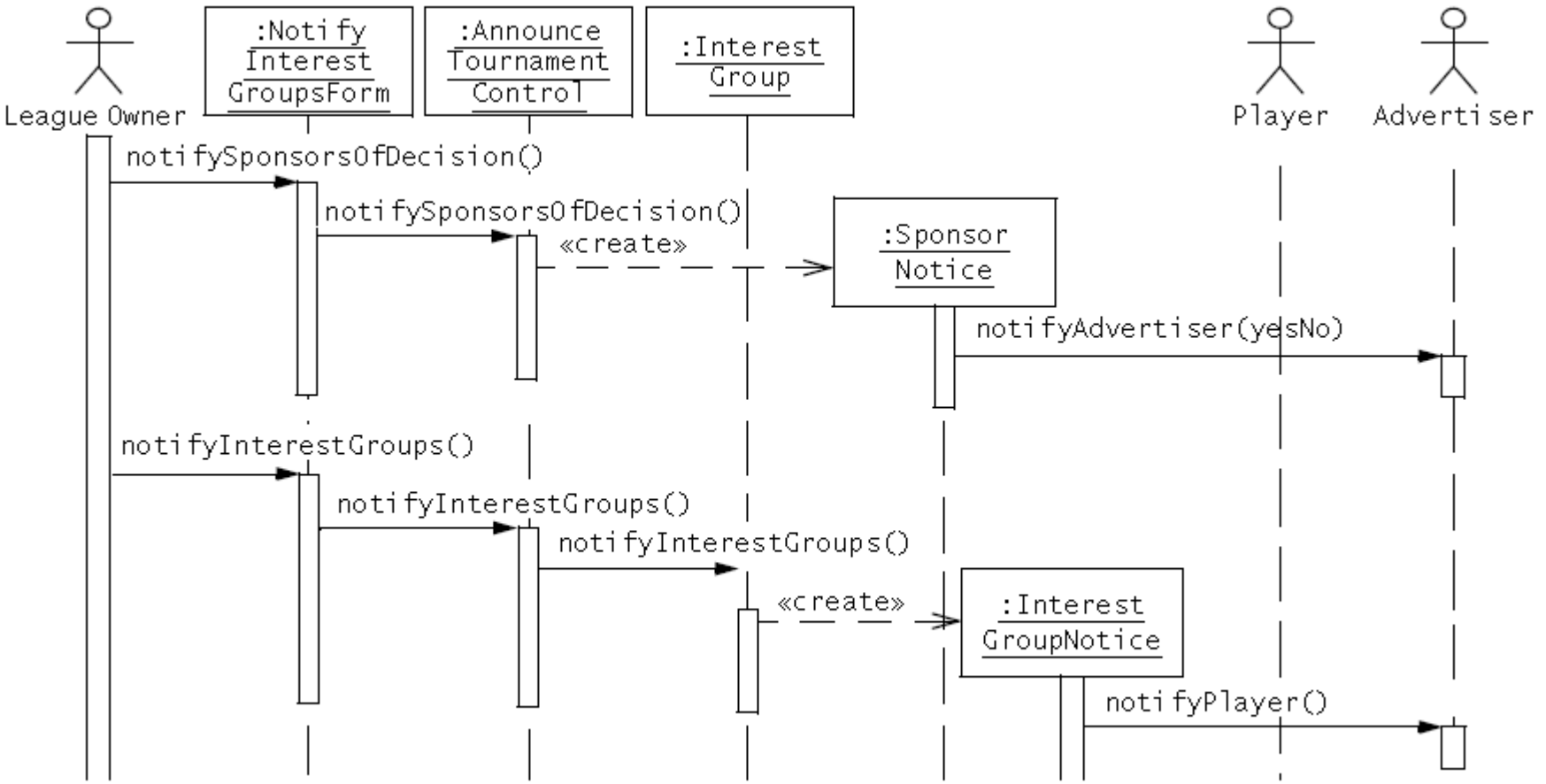
Object interaction: tournament creation workflow



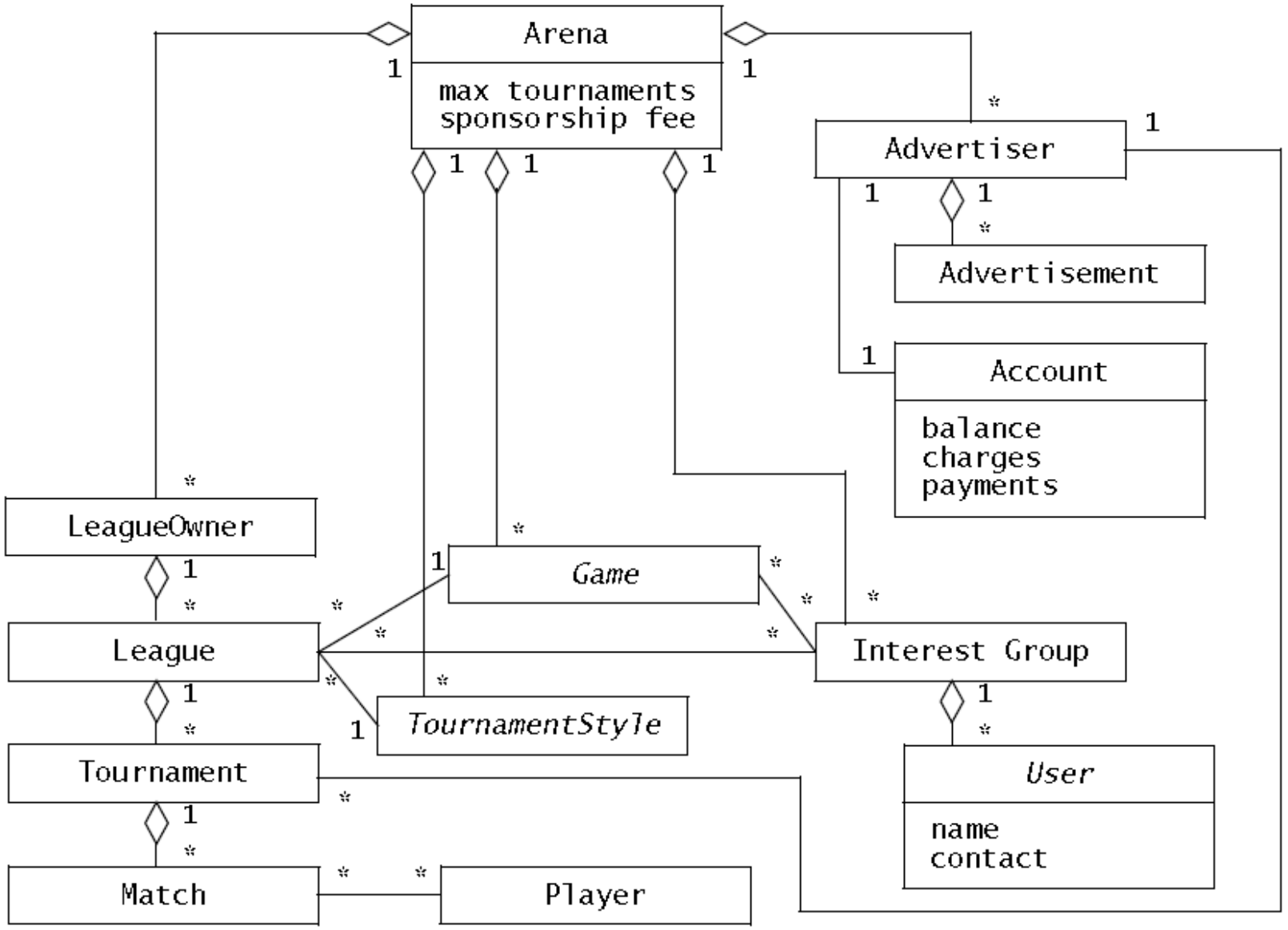
Object interaction: sponsorship workflow



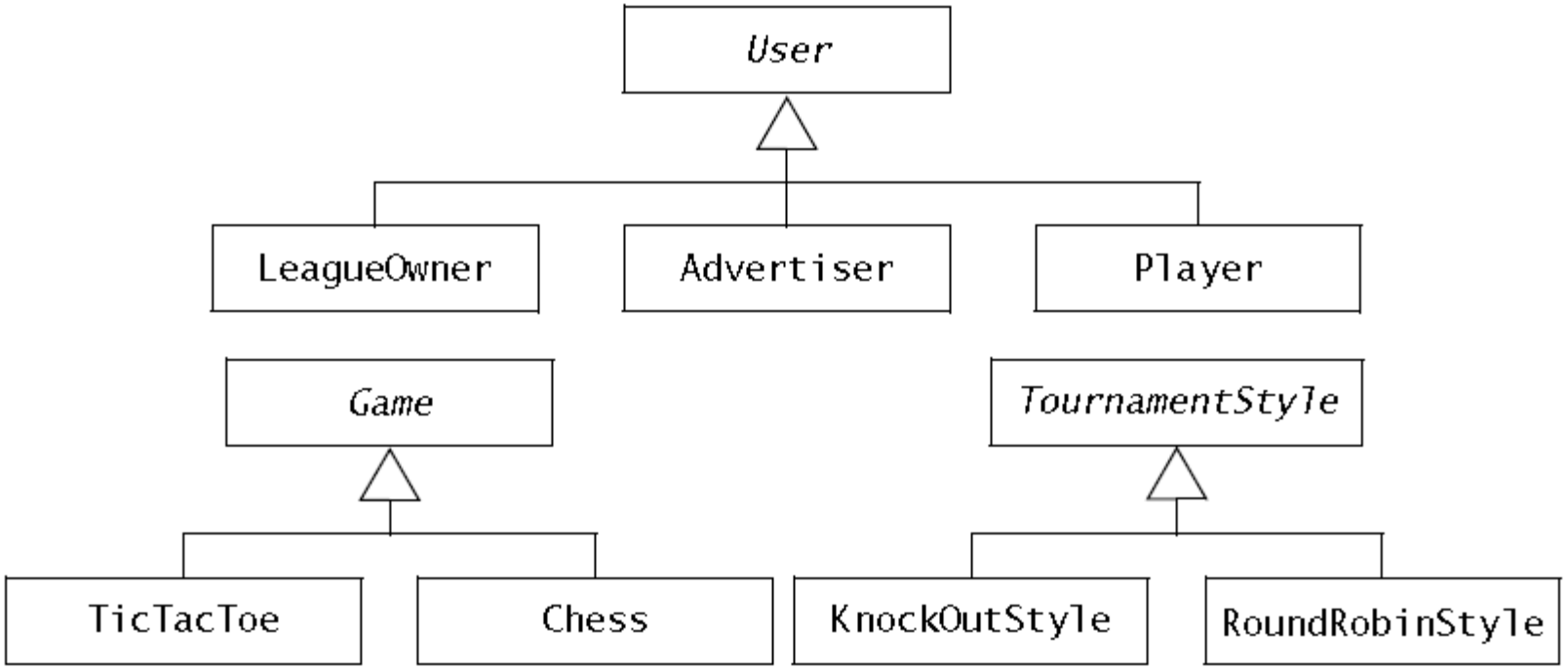
Object interaction: interest group workflow



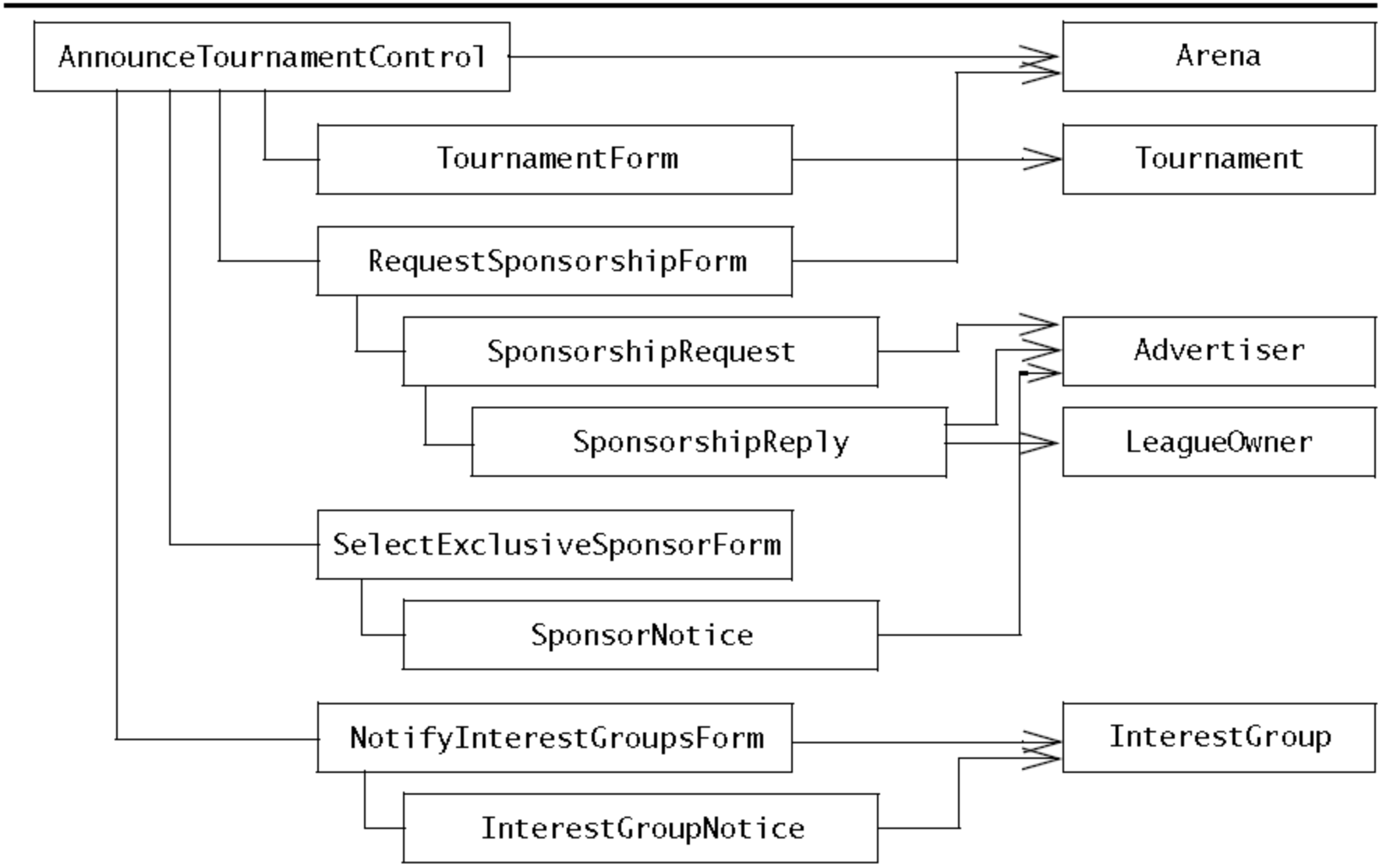
Consolidated class diagram: entity objects



Inheritance hierarchy among entity objects



Associations among boundary, control and selected entity objects



Lessons learned during OO Analysis

- Identifying objects, their attributes and associations, takes many iterations, often with the client.
- Object identification uses many sources, including the problem statement, use case model, the glossary, and the event flows of the use cases.
- A nontrivial use case can require many sequence diagrams and several class diagrams.
- It is unrealistic to represent all discovered objects in a single diagram. Instead, each diagram serves a specific purpose:
  - depicting associations among entity objects
  - depicting associations among participating objects in one use case.
- Key deliverables, such as the glossary, should be kept up to date as the analysis model is refined.
  - others, such as sequence diagrams, can be redone later if necessary.
  - maintaining consistency at all times, however, is unrealistic.
- There are many different ways to model the same application domain or the same system, based on the personal style and experience of the analyst.
  - This calls for developing style guides and conventions within a project.