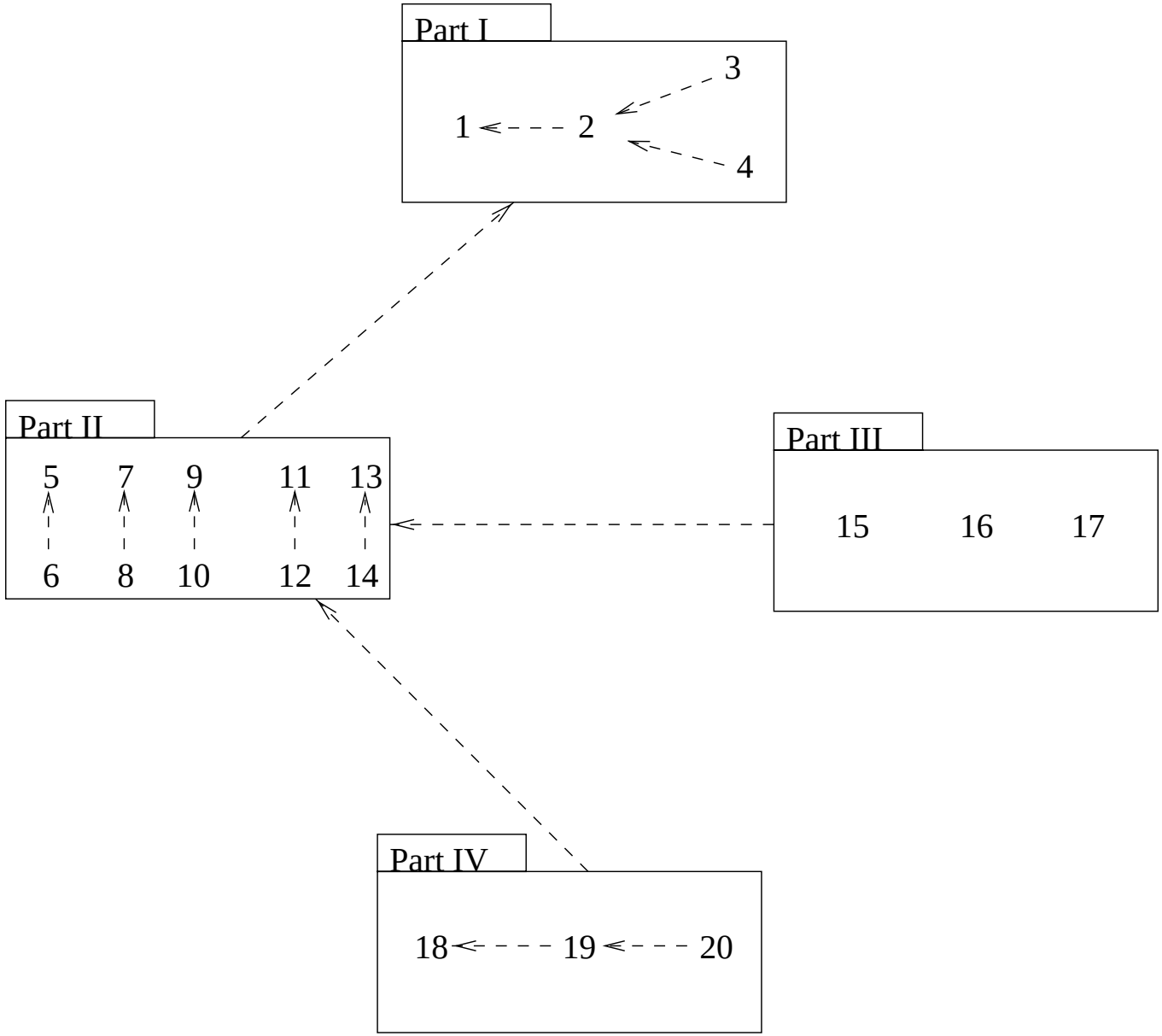


Figure Preface Chapter dependencies, not quite in UML!

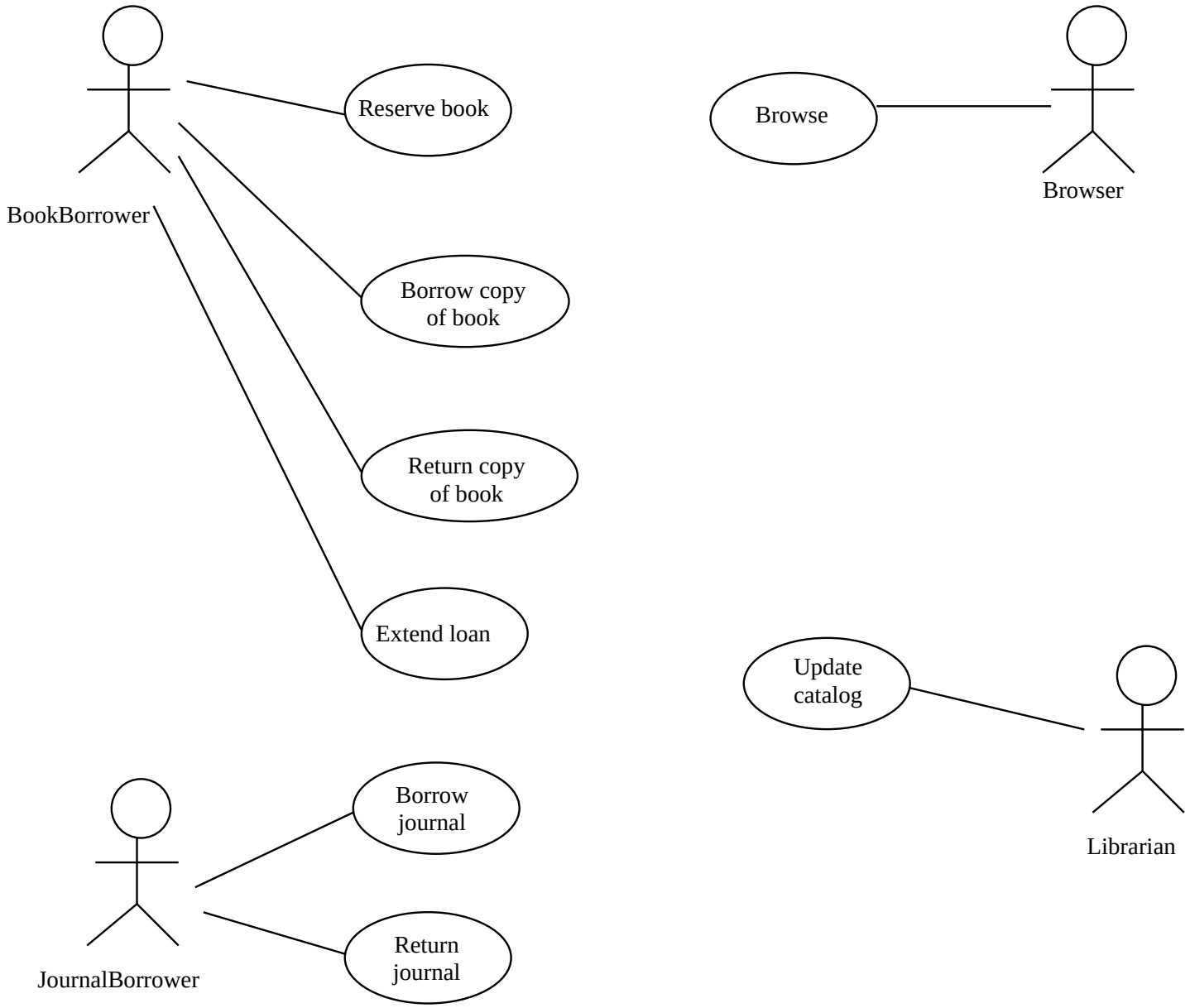


Figure 3.1 Use case diagram for the library.

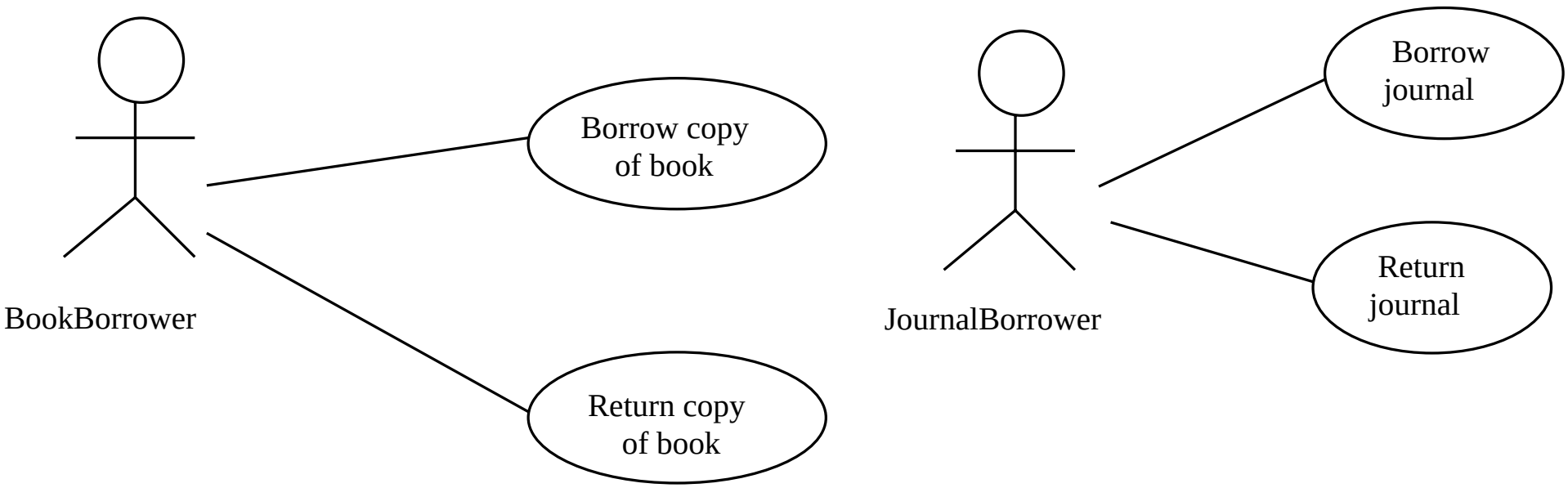


Figure 3.2 Use case diagram for the first iteration.

Books and journals The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals.

Borrowing The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.

Figure 3.3 Nouns and noun phrases in the library.

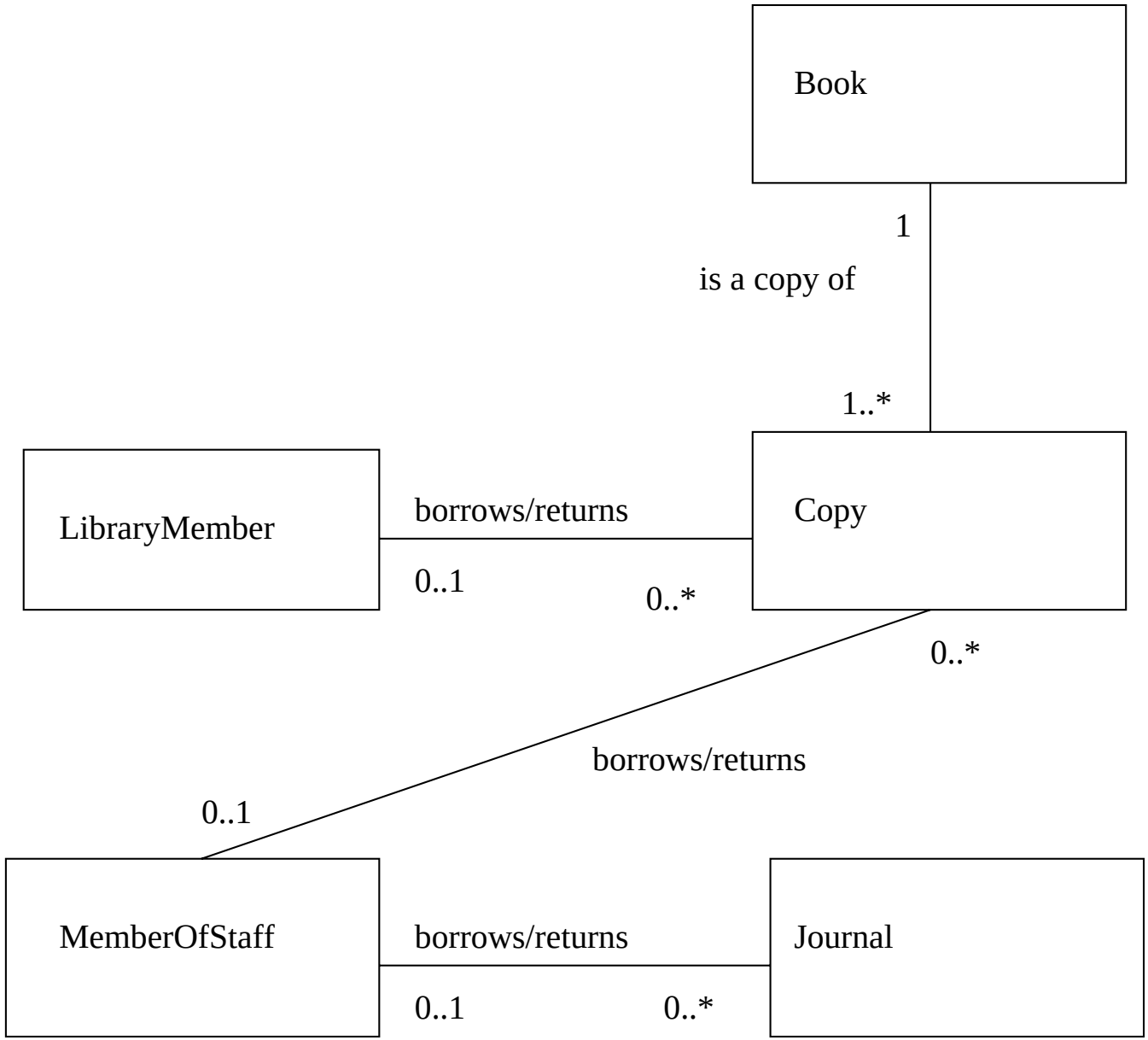


Figure 3.4 Initial class model of the library.

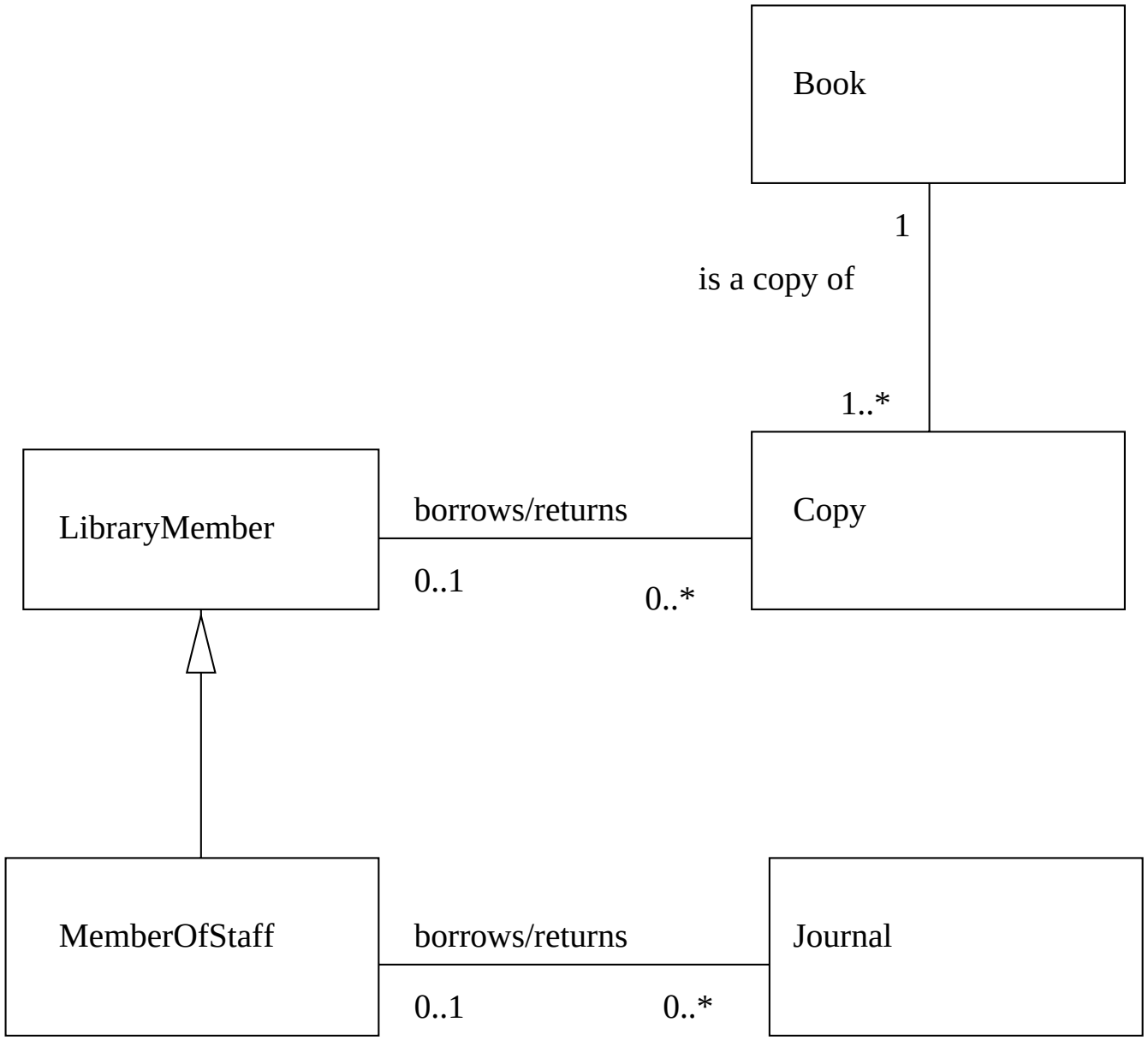


Figure 3.5 Revised library class model.

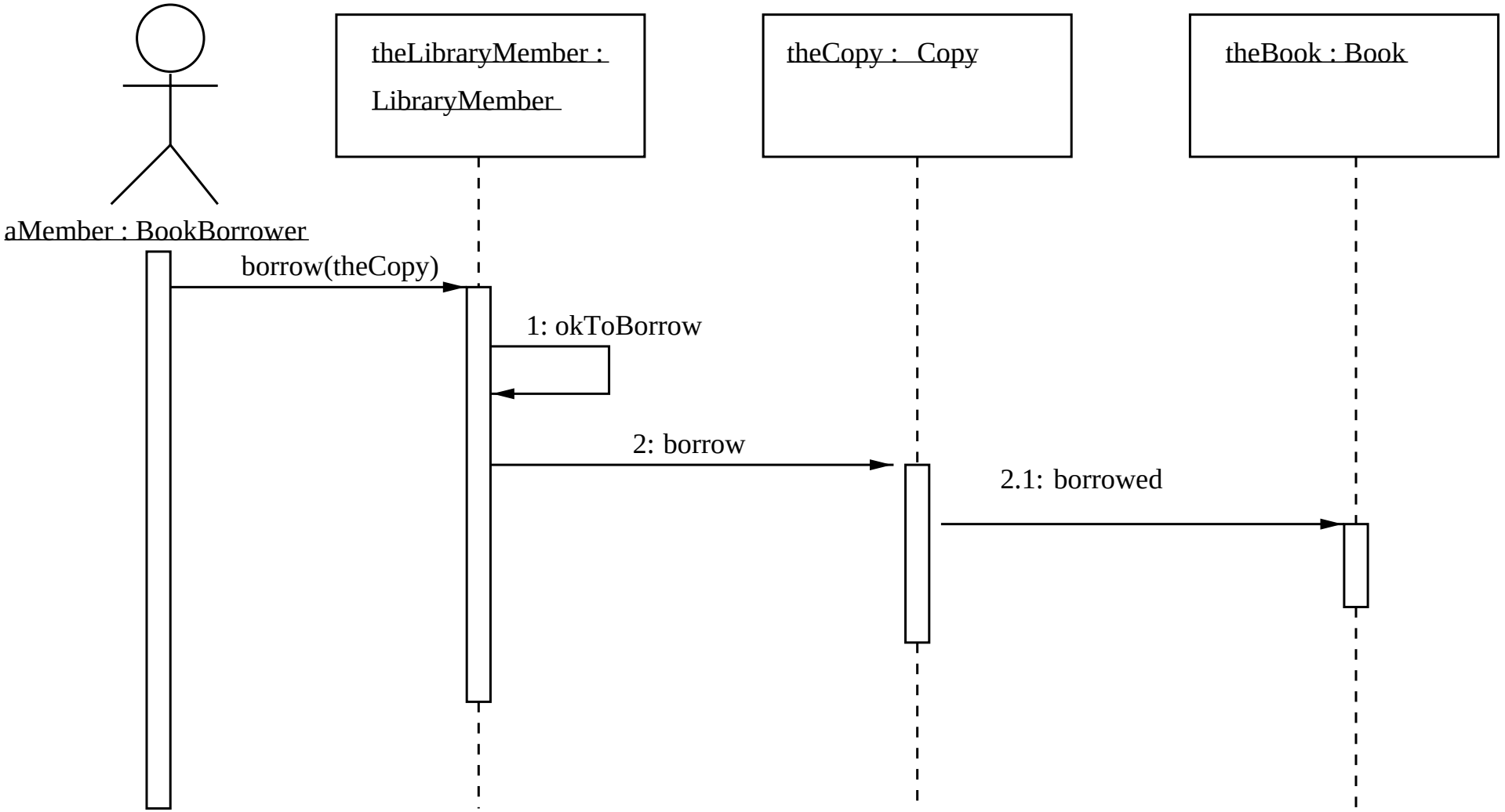


Figure 3.6 Interaction shown on a sequence diagram.

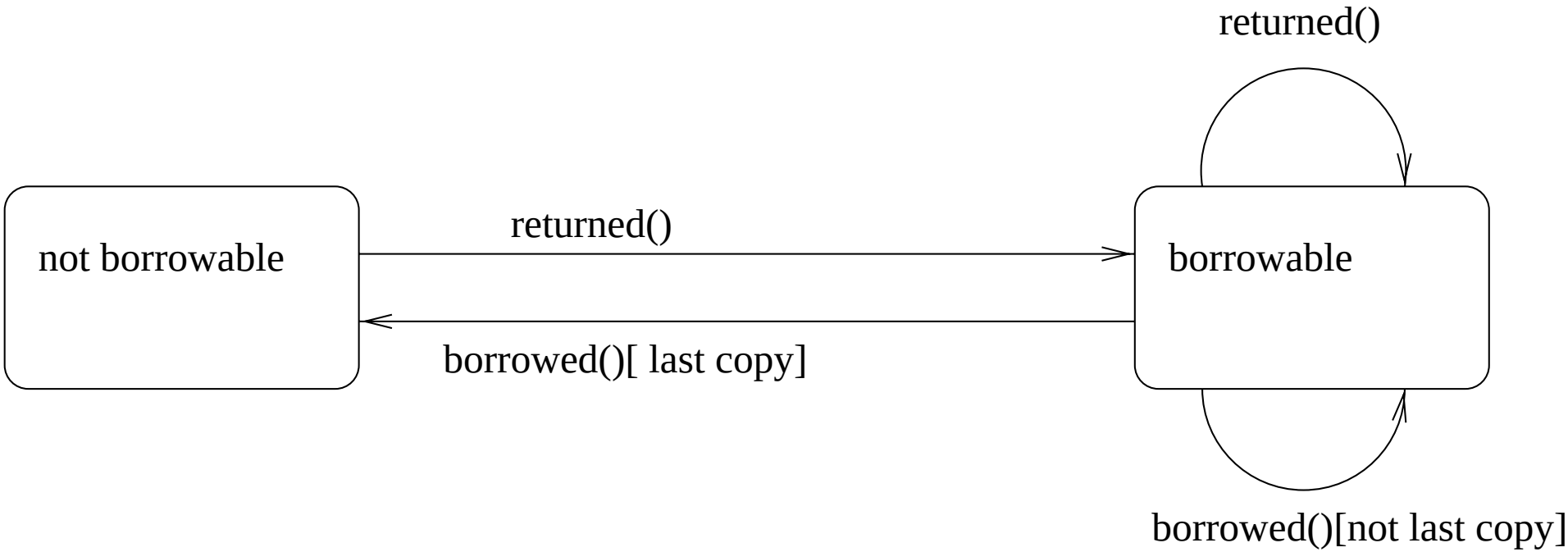


Figure 3.7 State diagram for class `Book`.

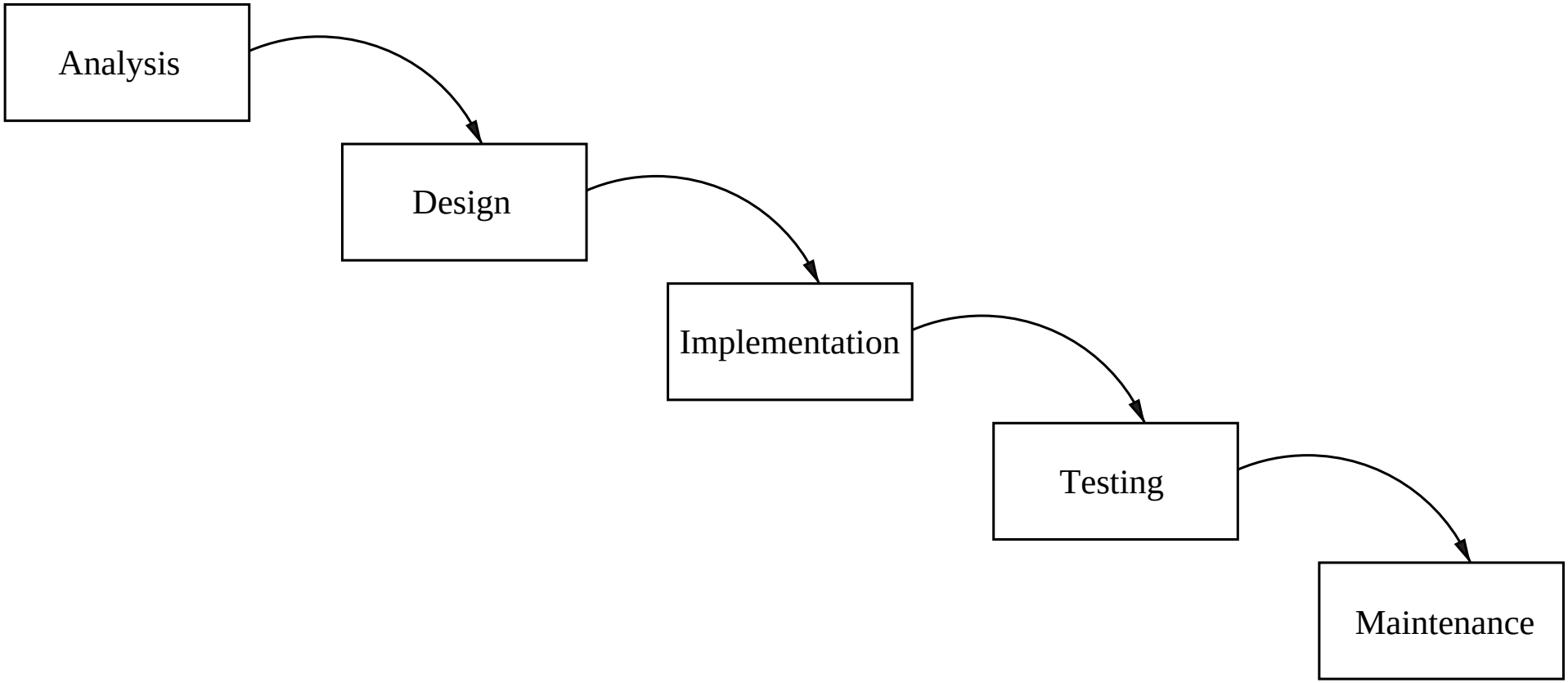


Figure 4.1 A simple waterfall process.

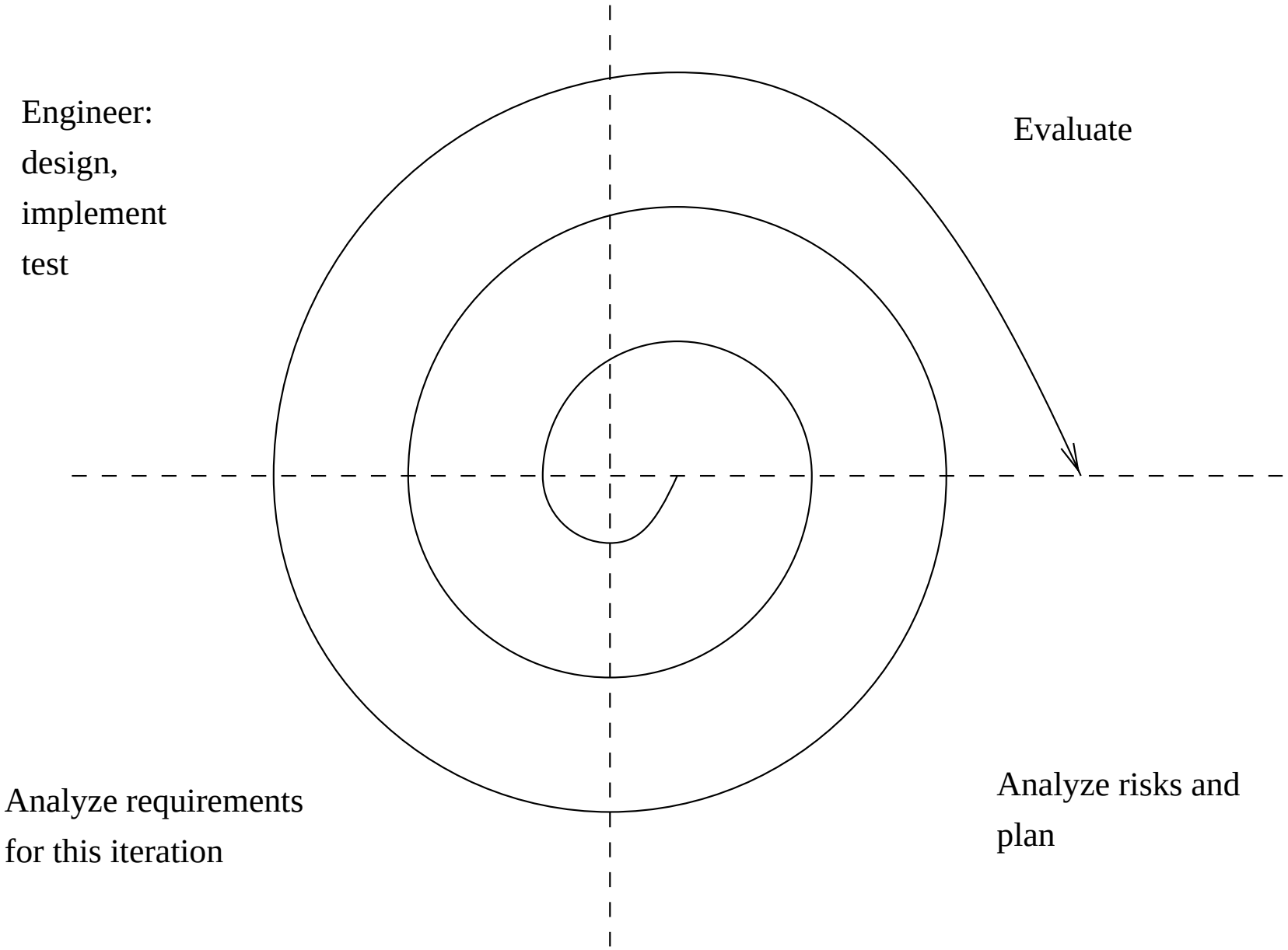


Figure 4.2 A simple spiral process.

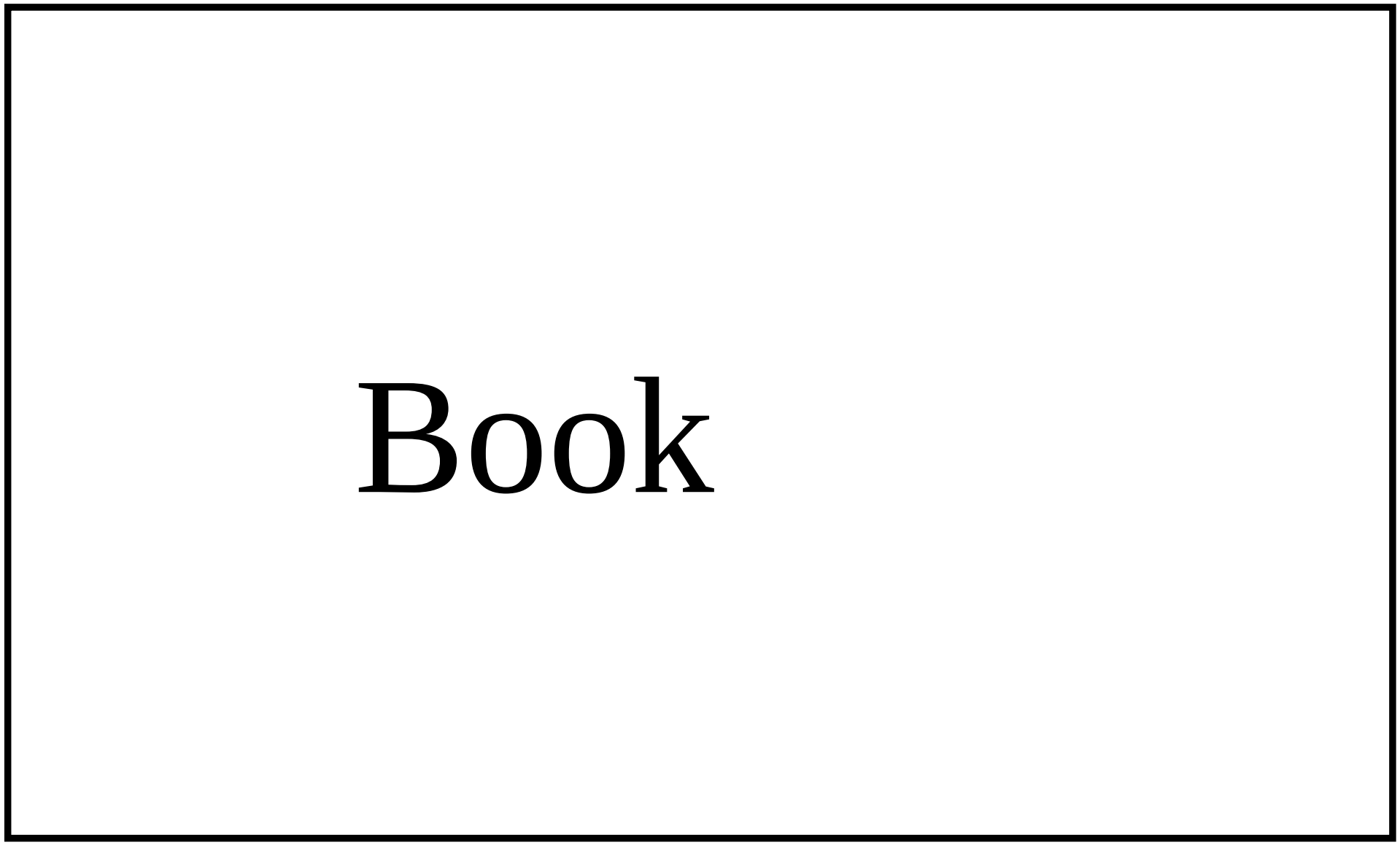


Figure 5.1 A very simple class model.

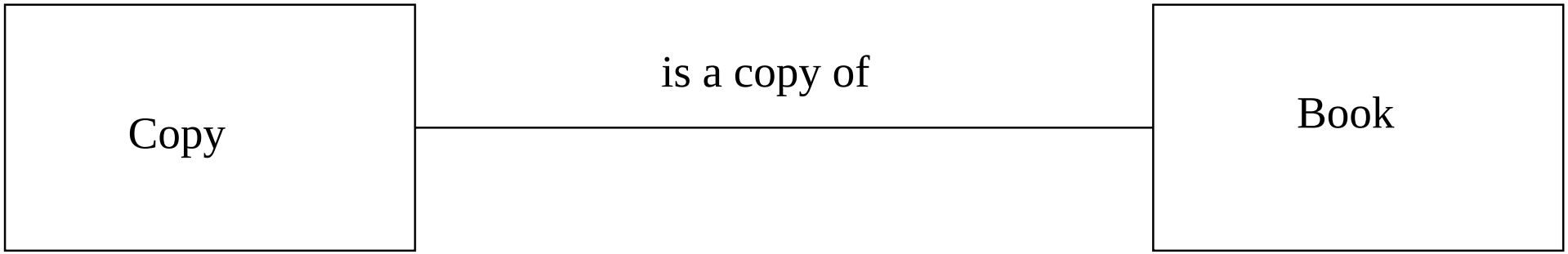


Figure 5.2 Simple association between classes.

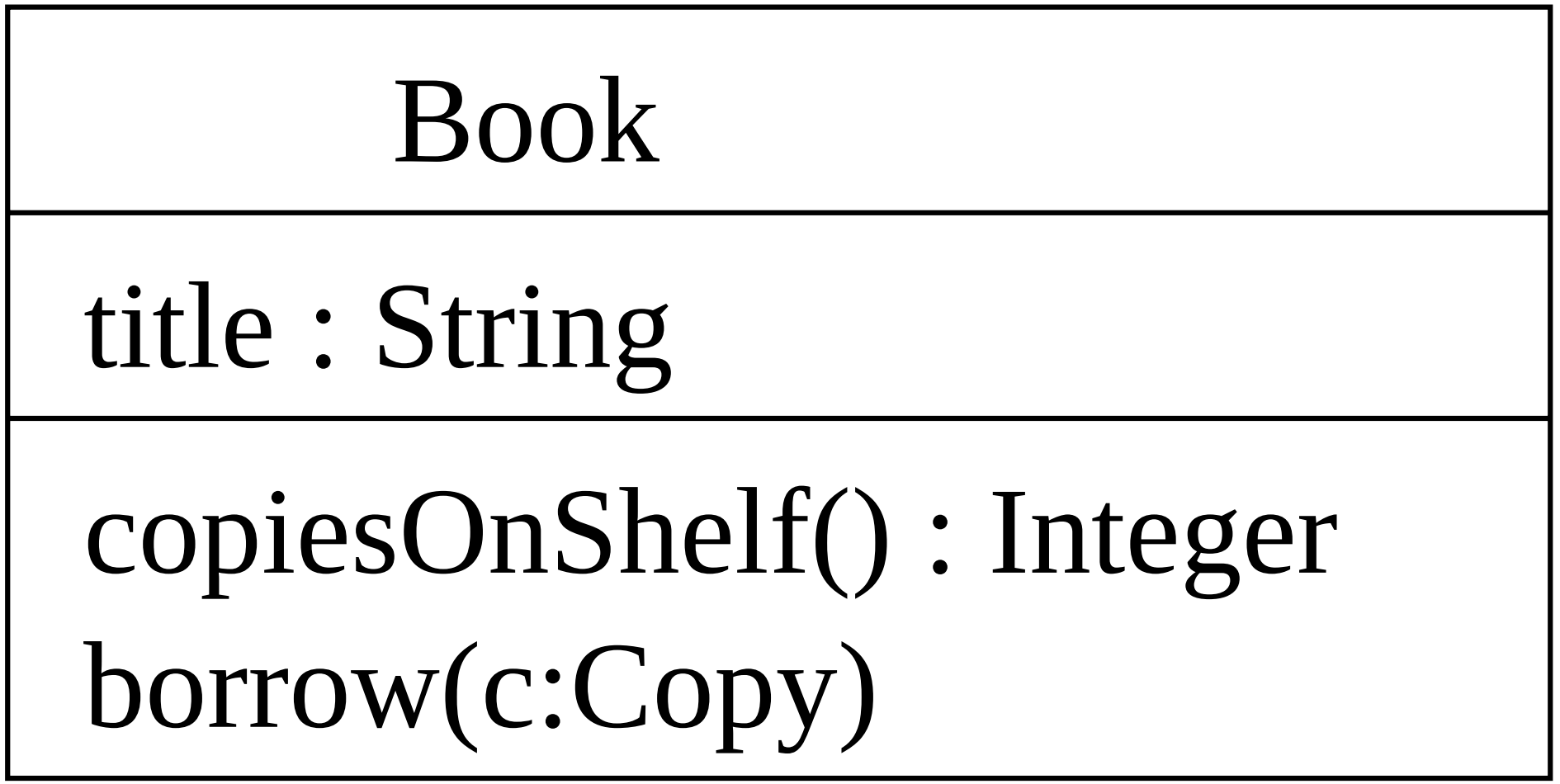


Figure 5.3 A simple class model, with attribute and operation.

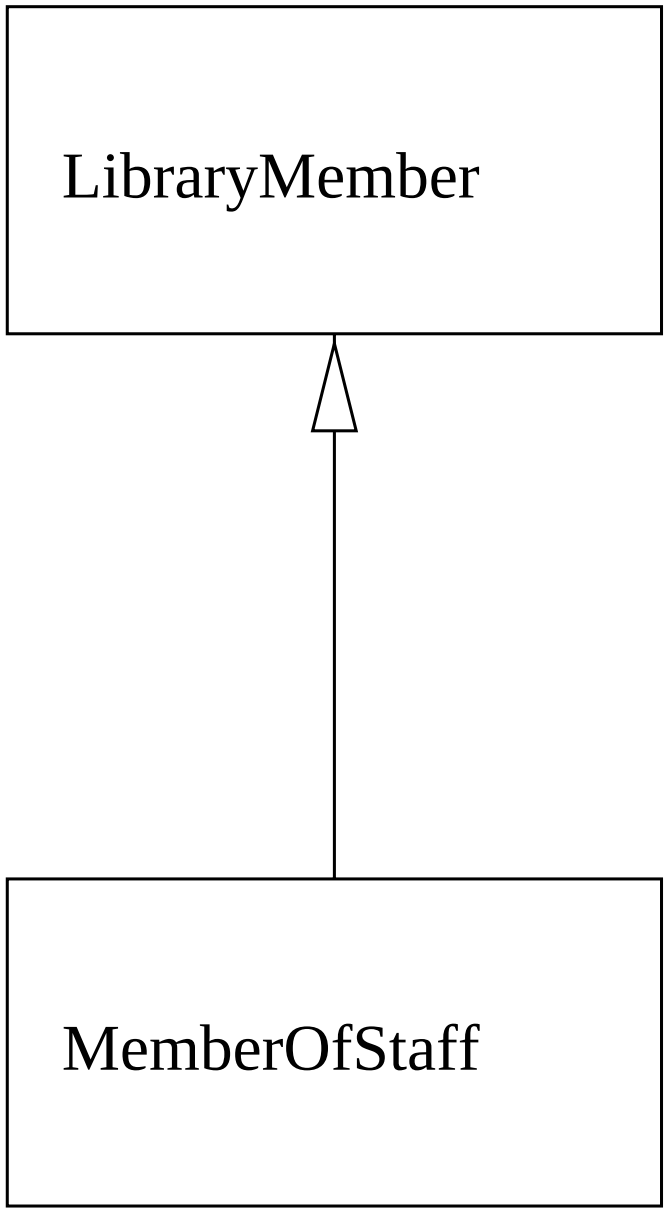


Figure 5.4 A simple generalization.

`LibraryMember`	
Responsibilities	**Collaborators**
Maintain data about copies currently borrowed Meet requests to borrow and return copies	 `Copy`

`Copy`	
Responsibilities	**Collaborators**
Maintain data about a particular copy of a book Inform corresponding `Book` when borrowed and returned	 `Book`

`Book`	
Responsibilities	**Collaborators**
Maintain data about one book Know whether there are borrowable copies	

Figure 5.5 Example of CRC cards for the library.

Figure 6.1 An aggregation.

Figure 6.2 A composition.

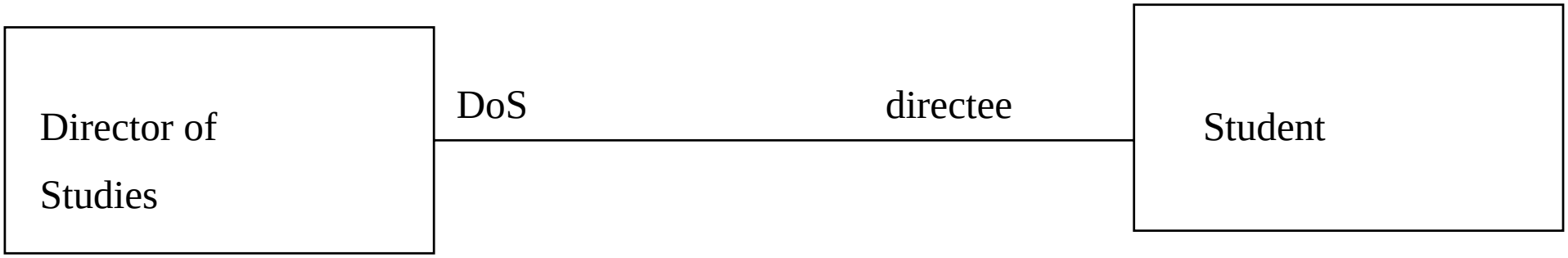


Figure 6.3 An association shown with role names.

Figure 6.4 Association with no navigability shown.

Figure 6.5 Association with one-way navigability shown.

Figure 6.6 Plain association between `Square` and `Board`.

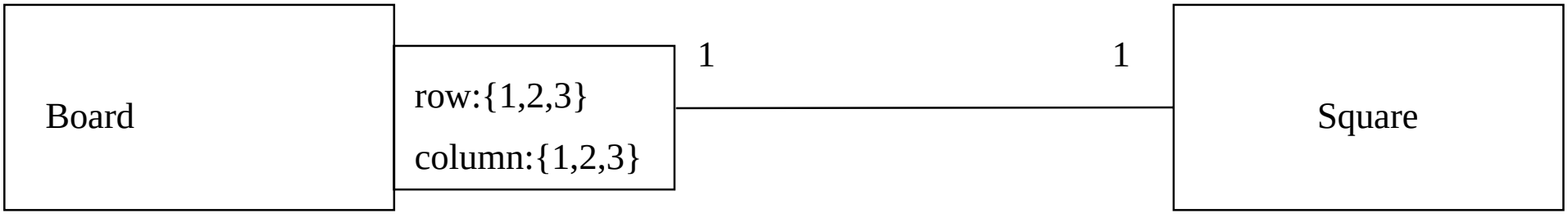


Figure 6.7 Qualified association.

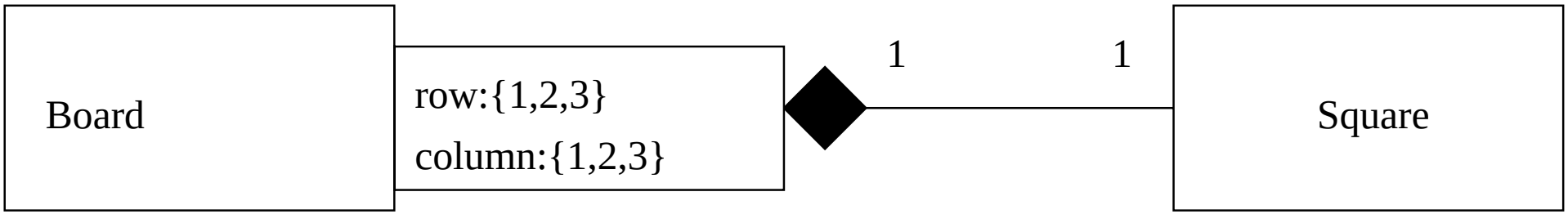


Figure 6.8 Qualified composition.

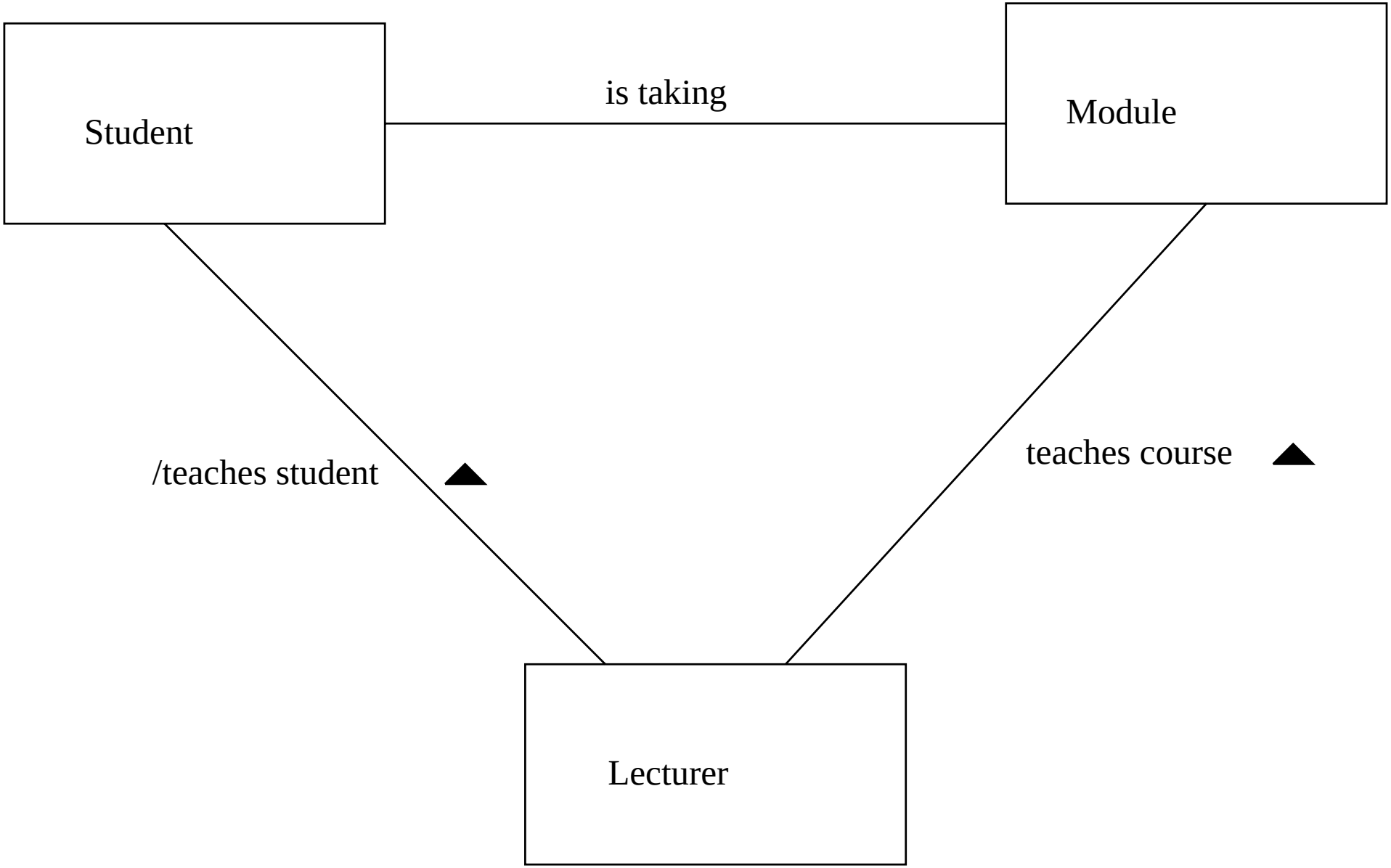


Figure 6.9 A derived association.

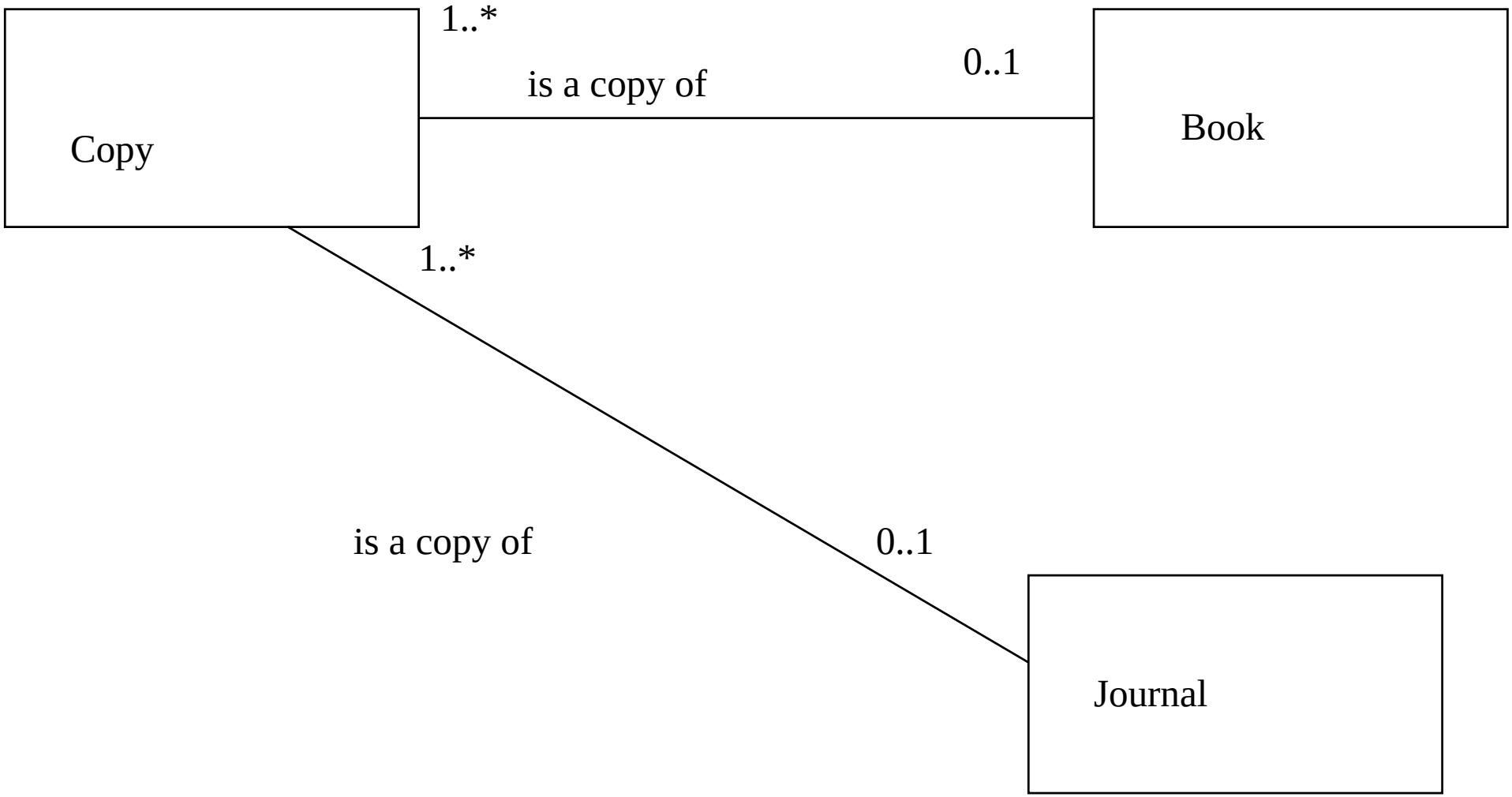


Figure 6.10 An under-constrained diagram.

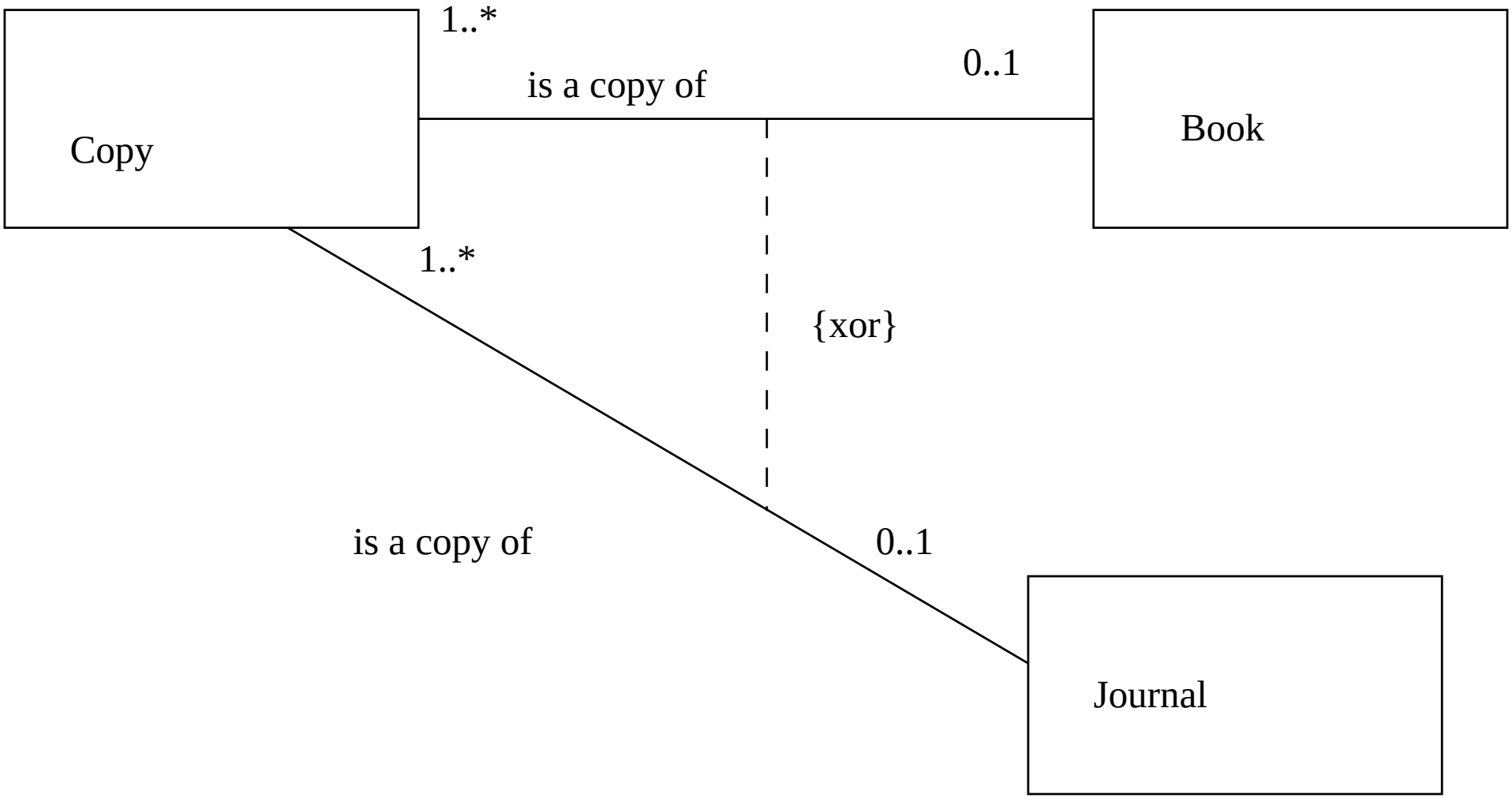


Figure 6.11 Using an `or`-constraint.

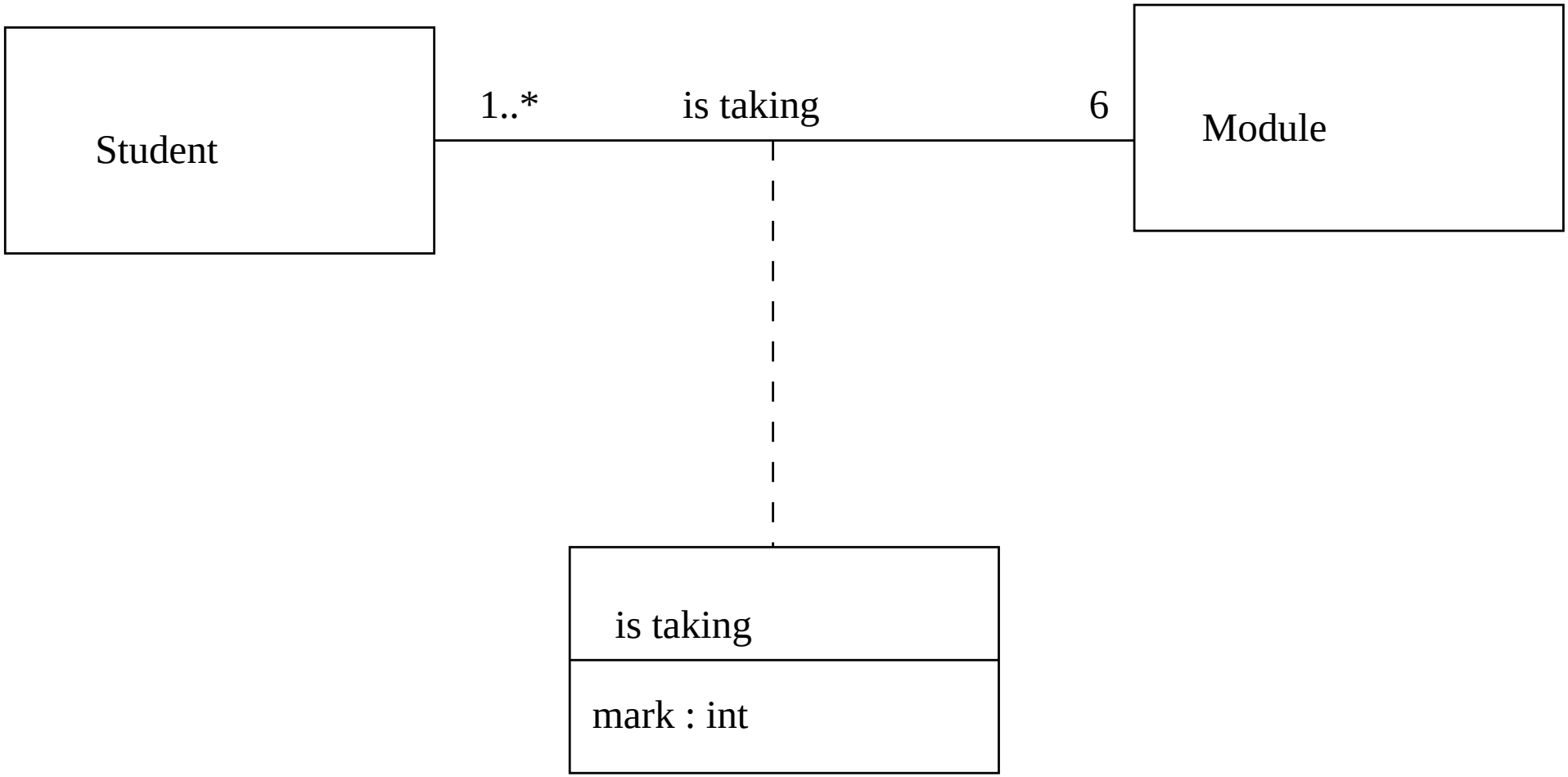


Figure 6.12 An association class.

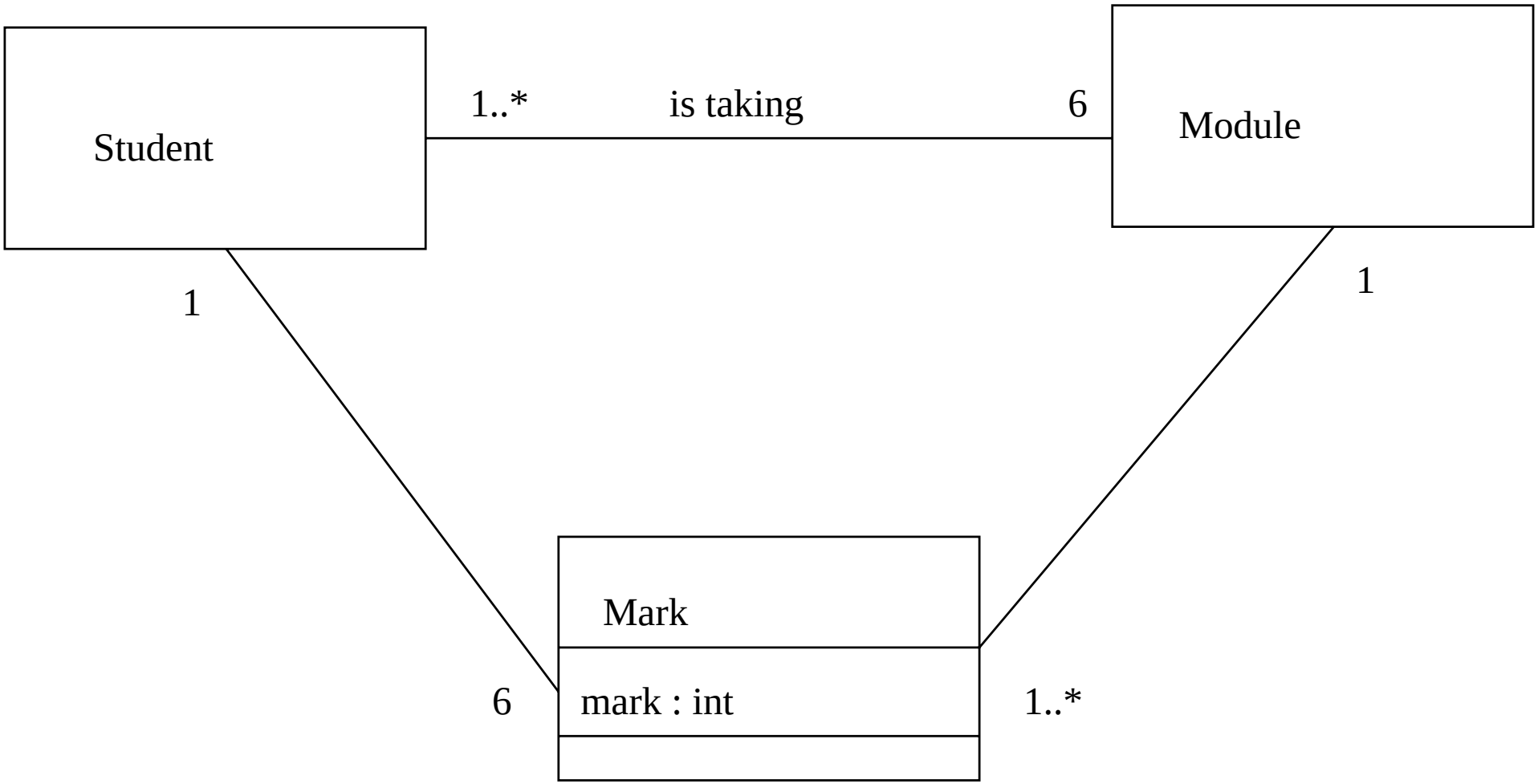


Figure 6.13 Avoiding an association class.

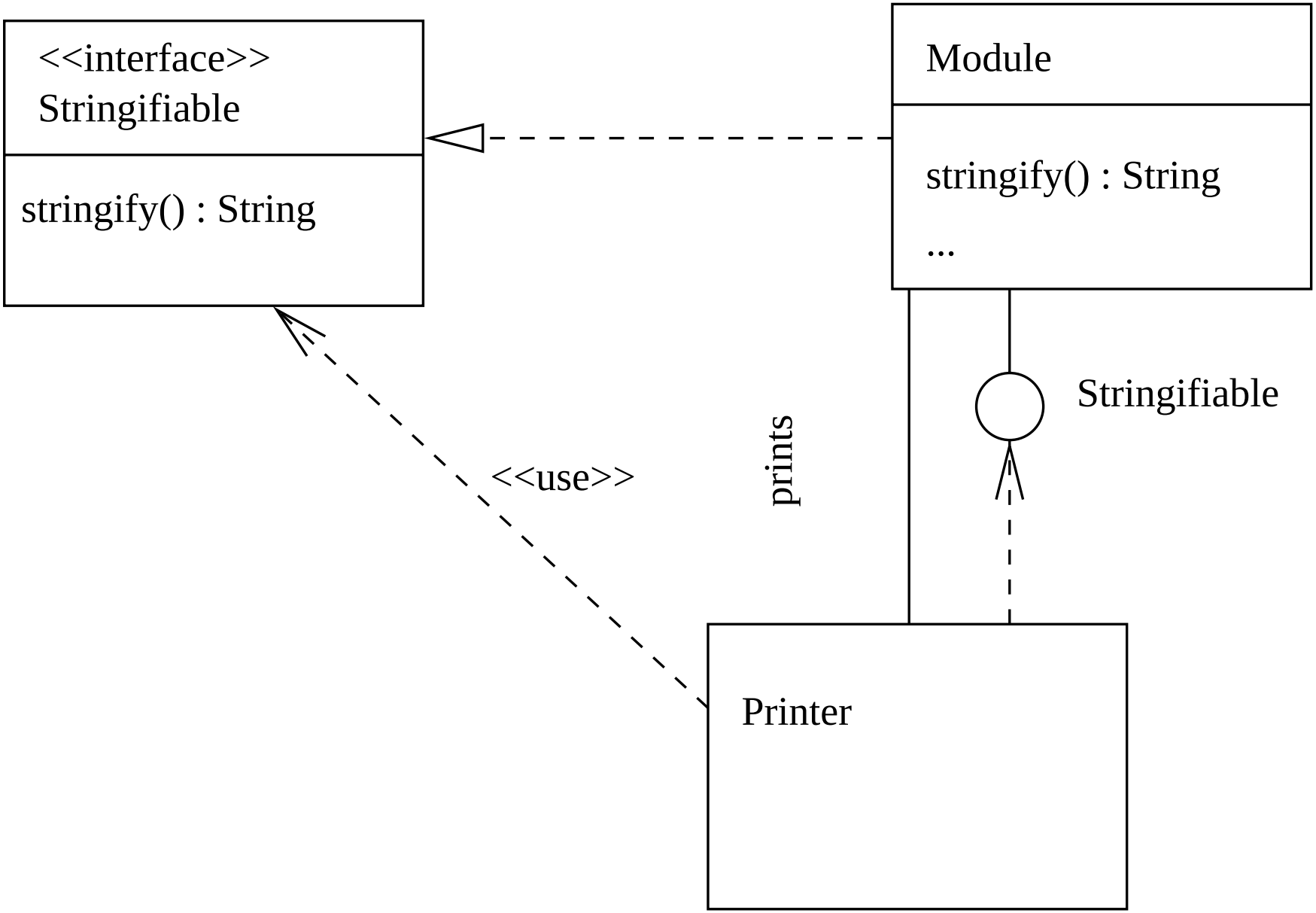


Figure 6.14 An interface and its use.

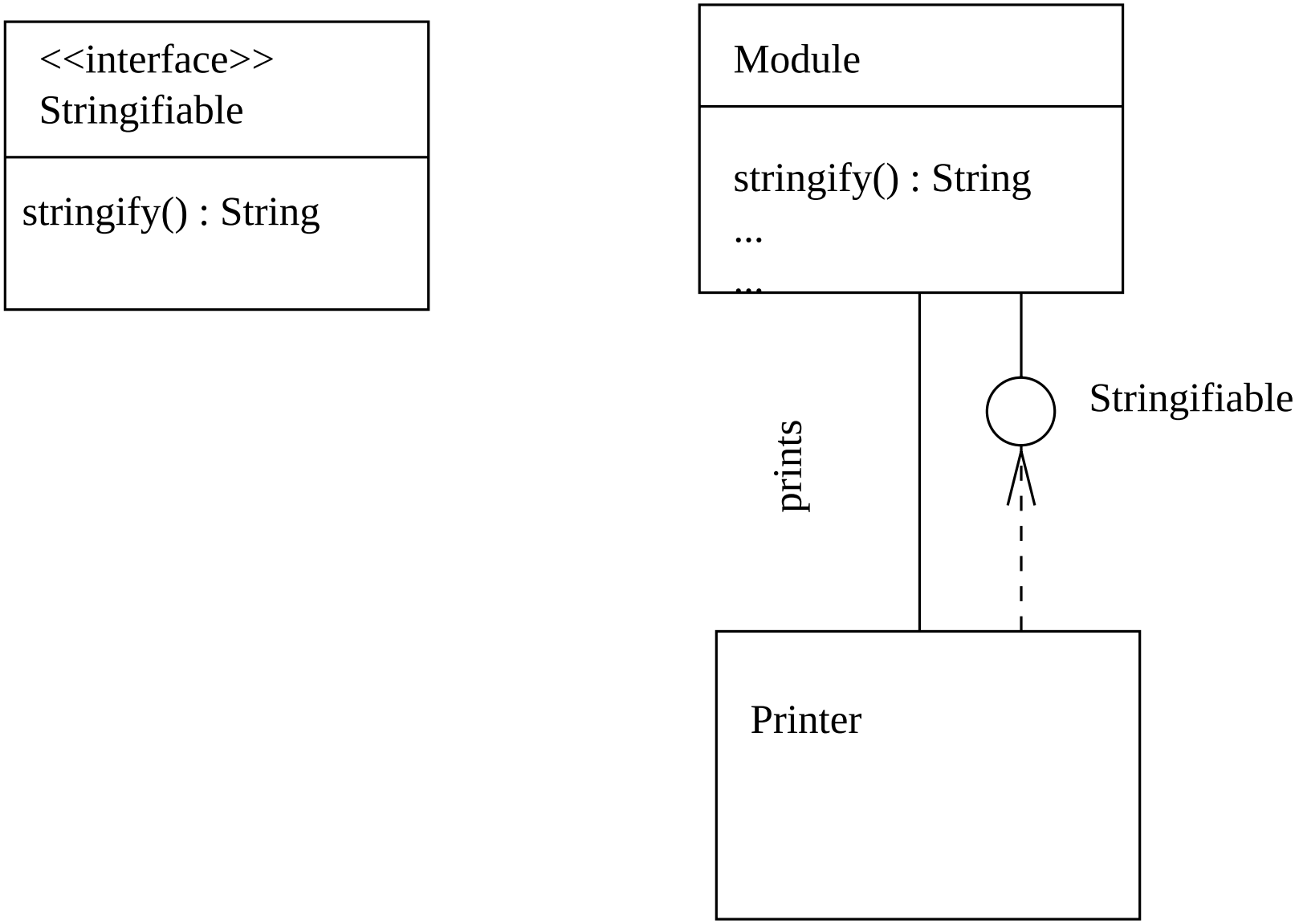


Figure 6.15 More parsimonious notation for interface dependency.

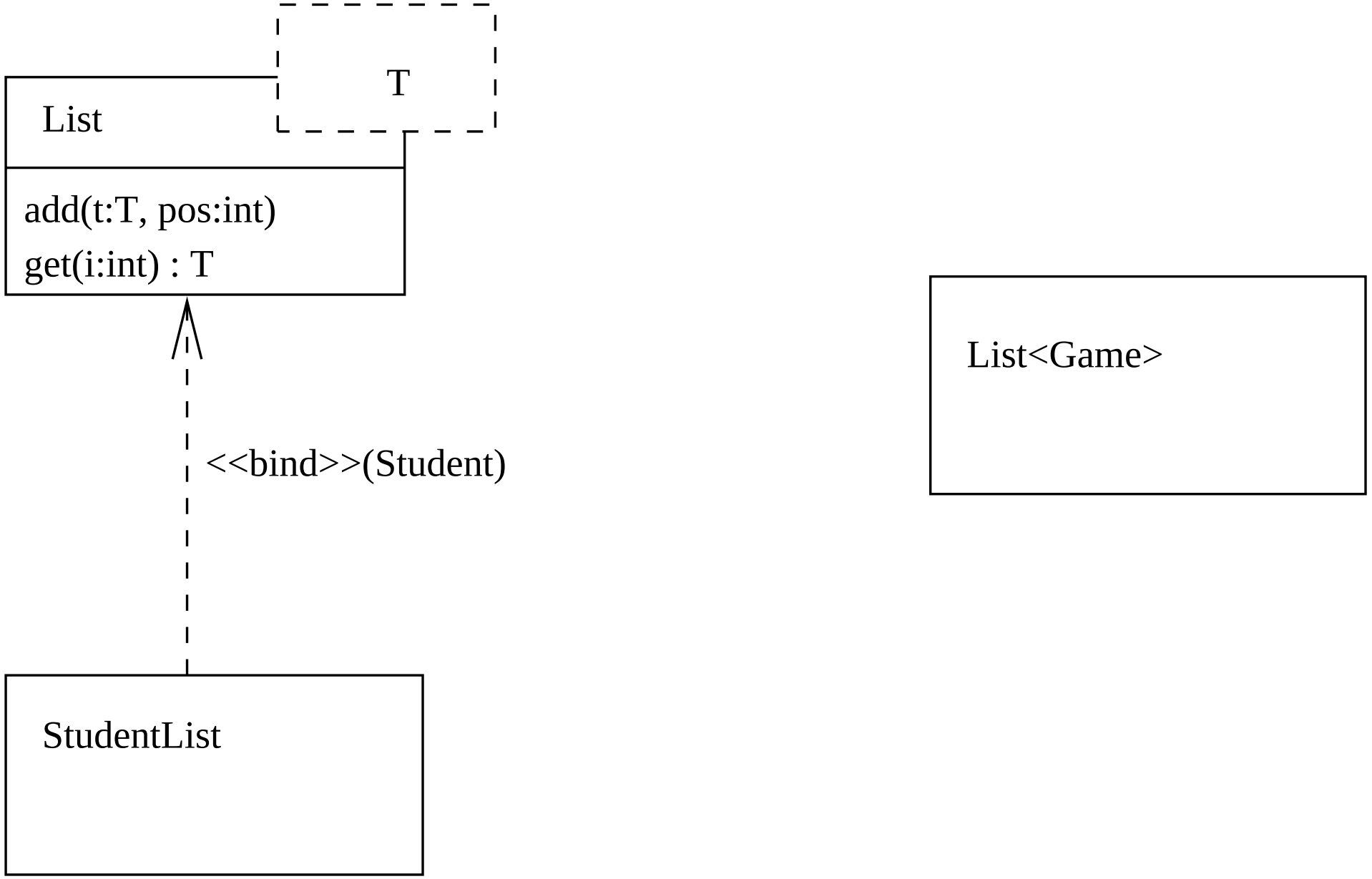


Figure 6.16 A parameterized class and its uses.

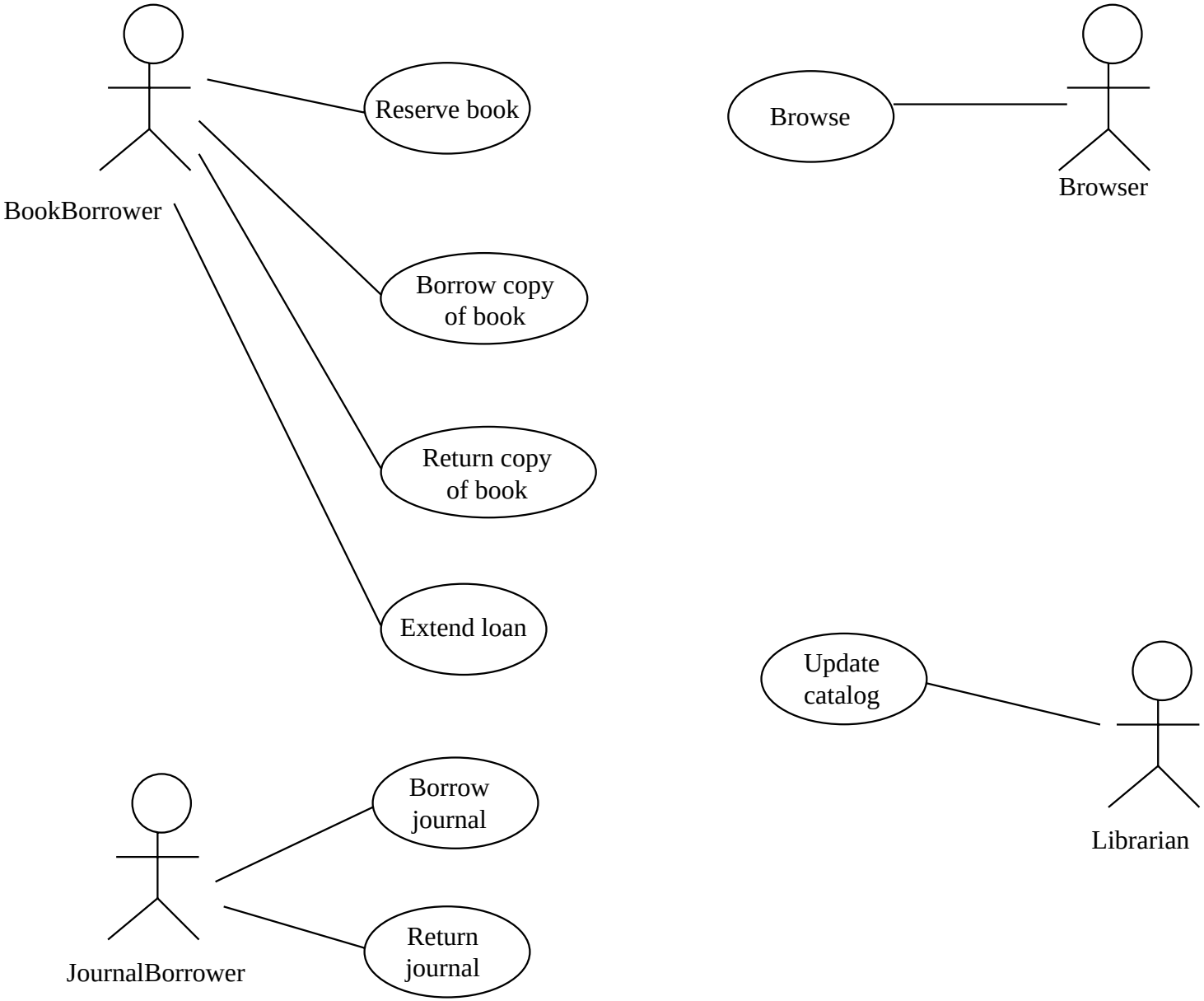


Figure 7.1 Use case diagram for the library.

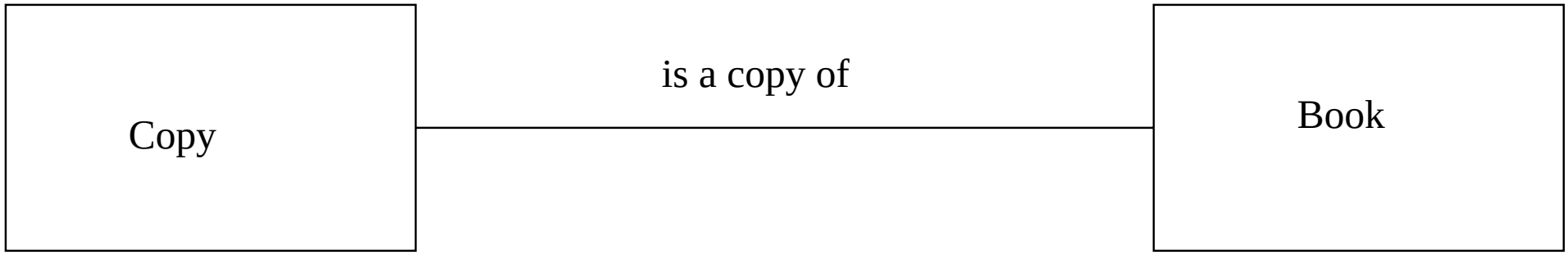


Figure 7.2 Simple association between classes.

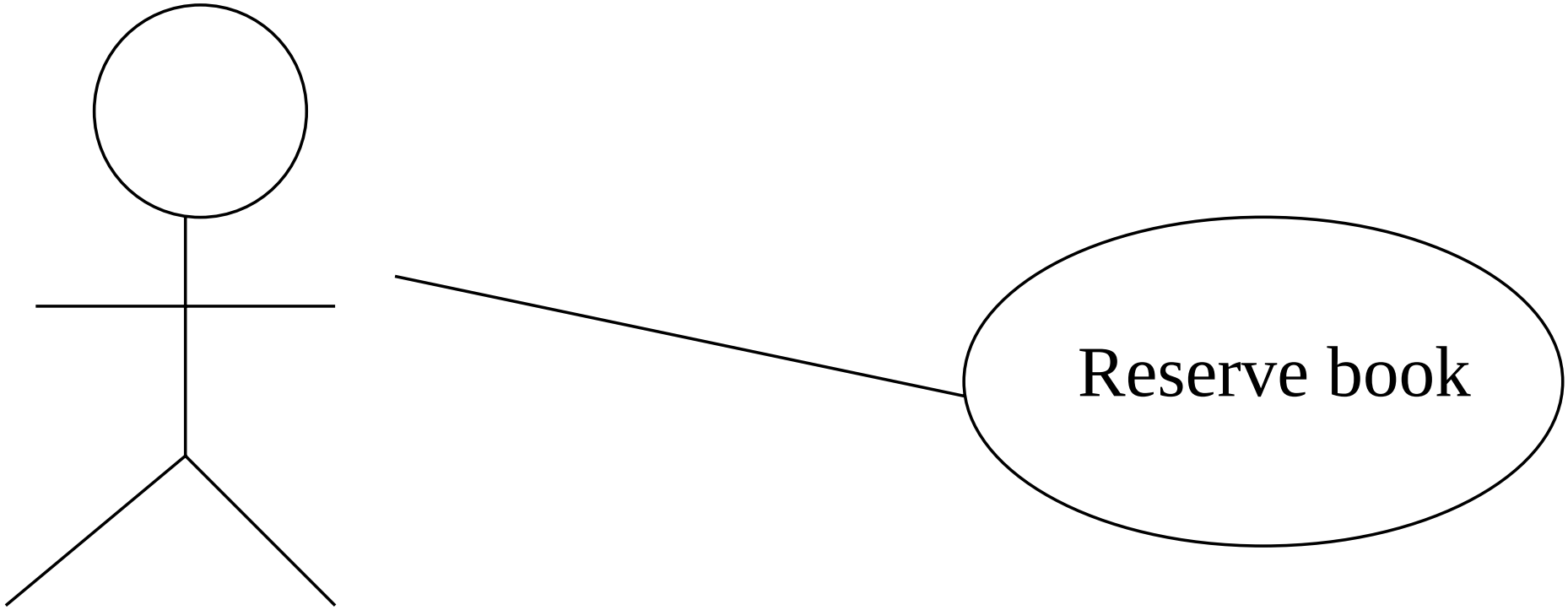


Figure 7.3 Simple communication between an actor and a use case.

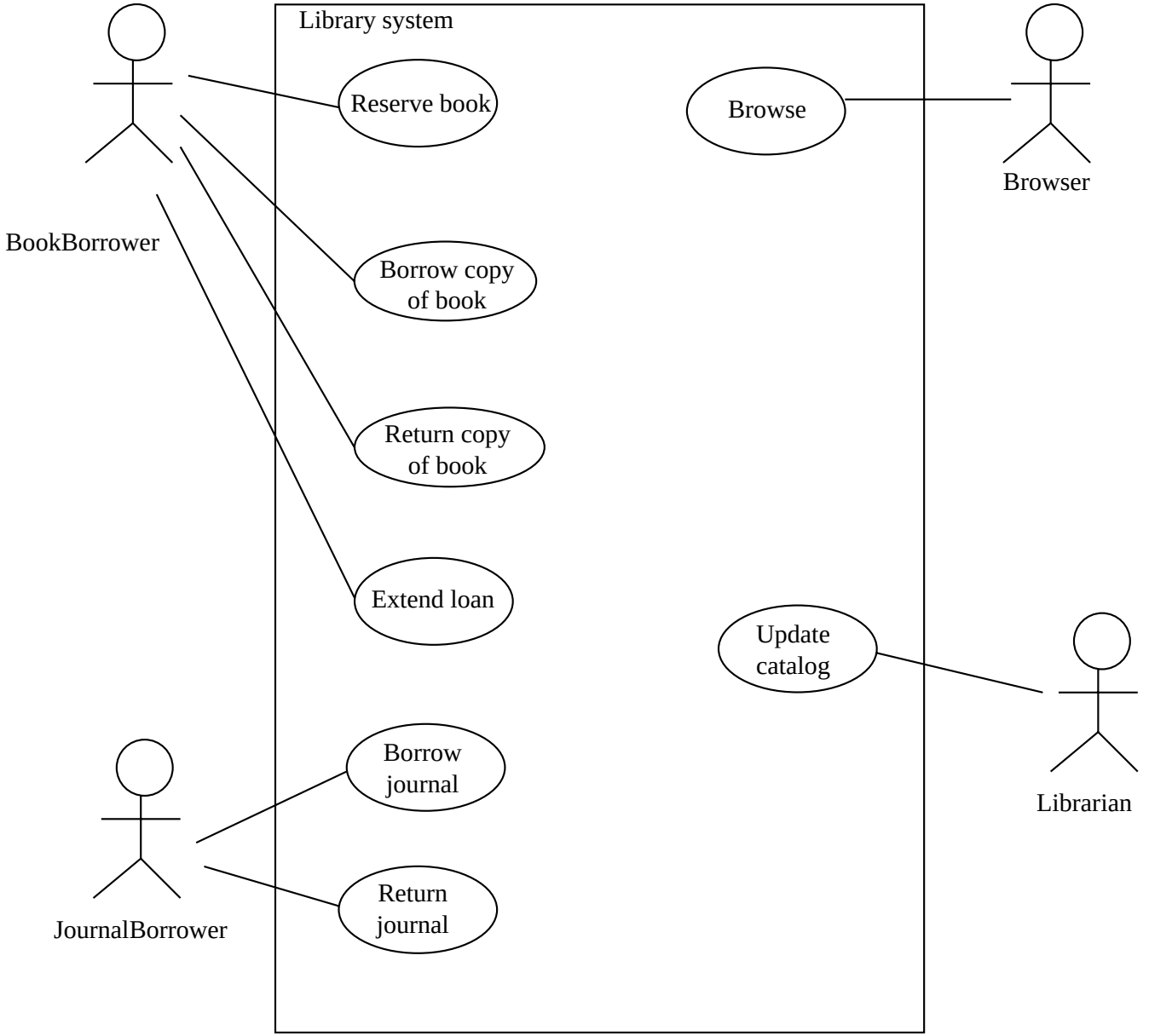


Figure 7.4 Use case diagram for the library.

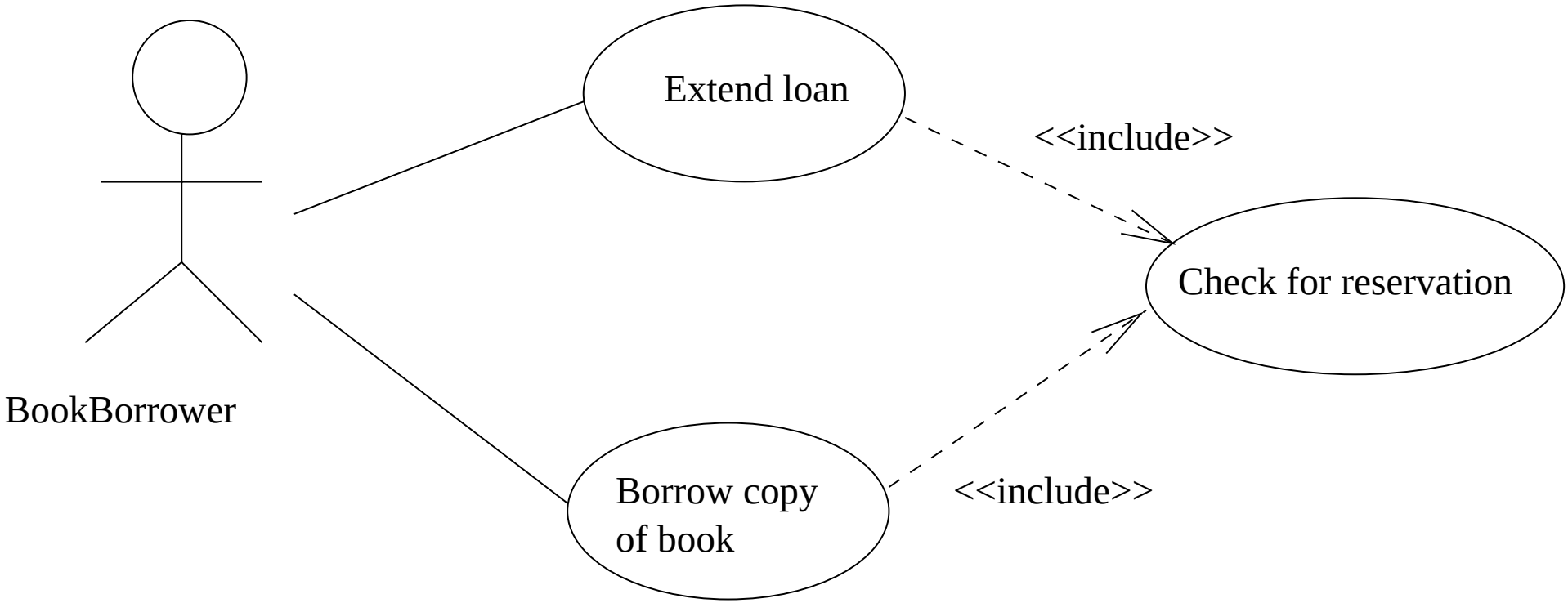


Figure 8.1 Use case reuse: ≪uses≫.

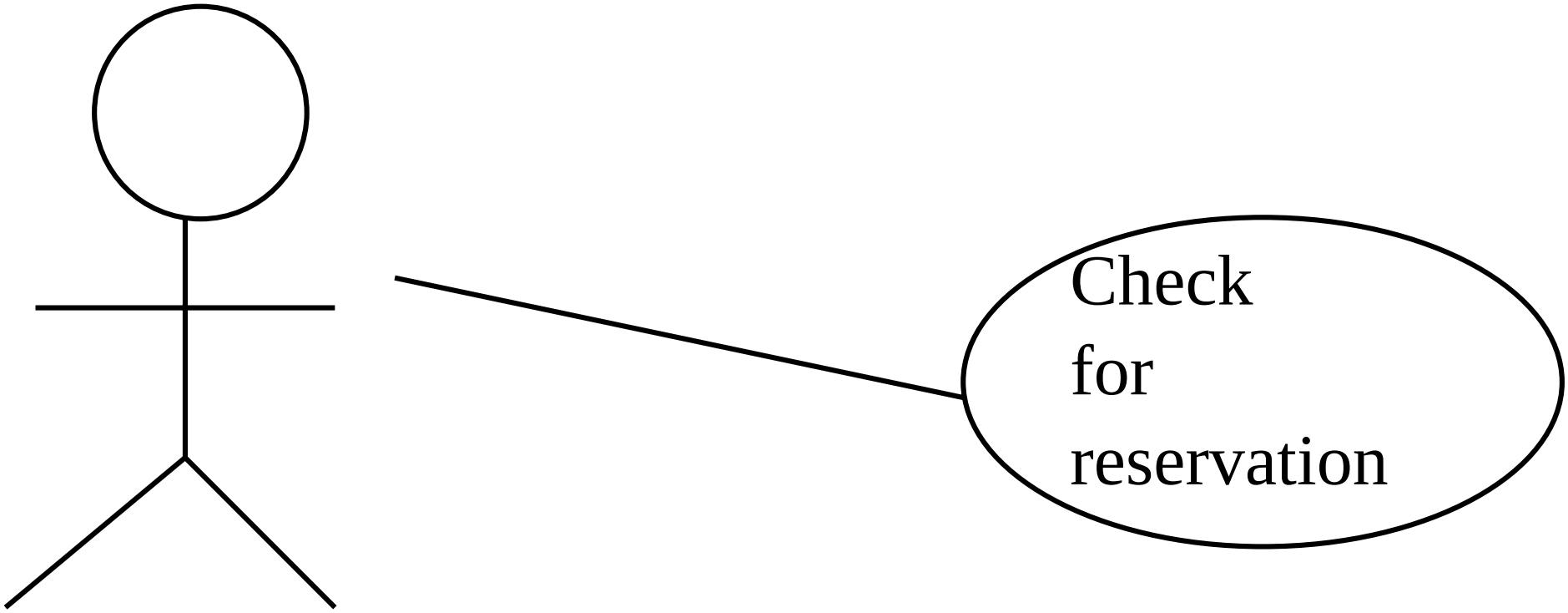


Figure 8.2 A use case diagram describing a component.

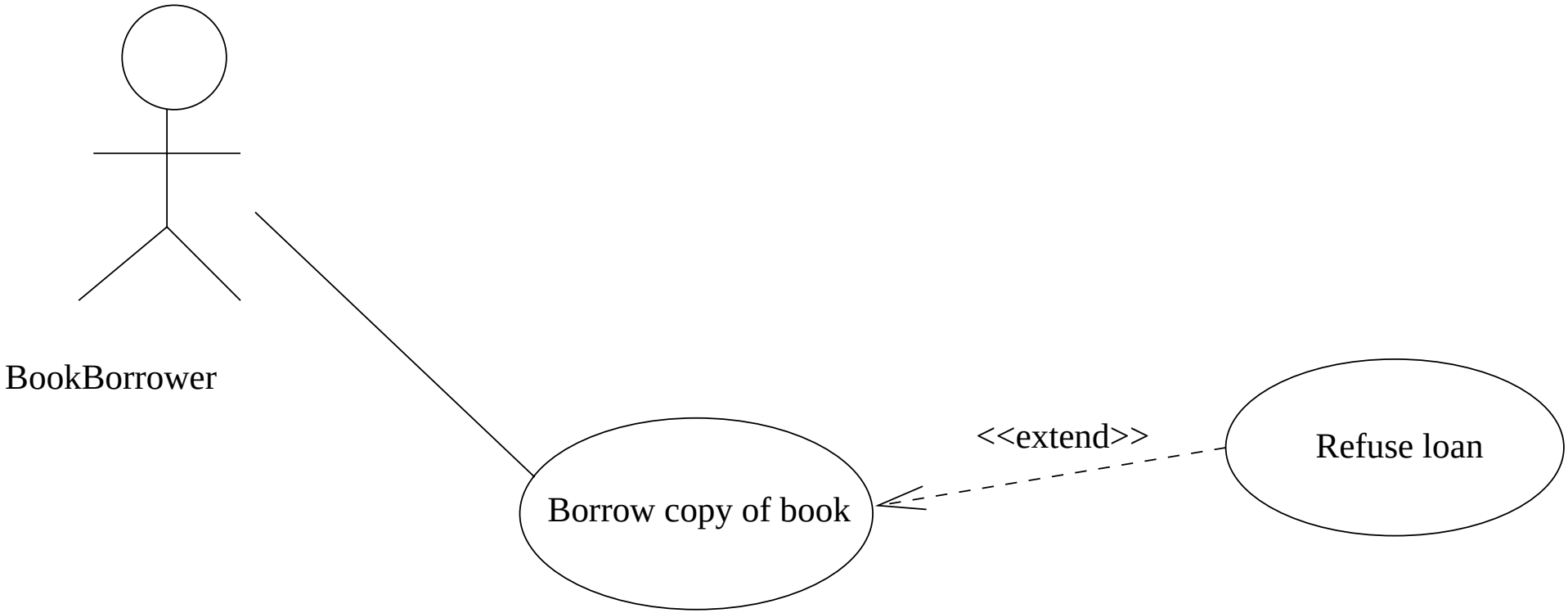


Figure 8.3 ≪extends≫.

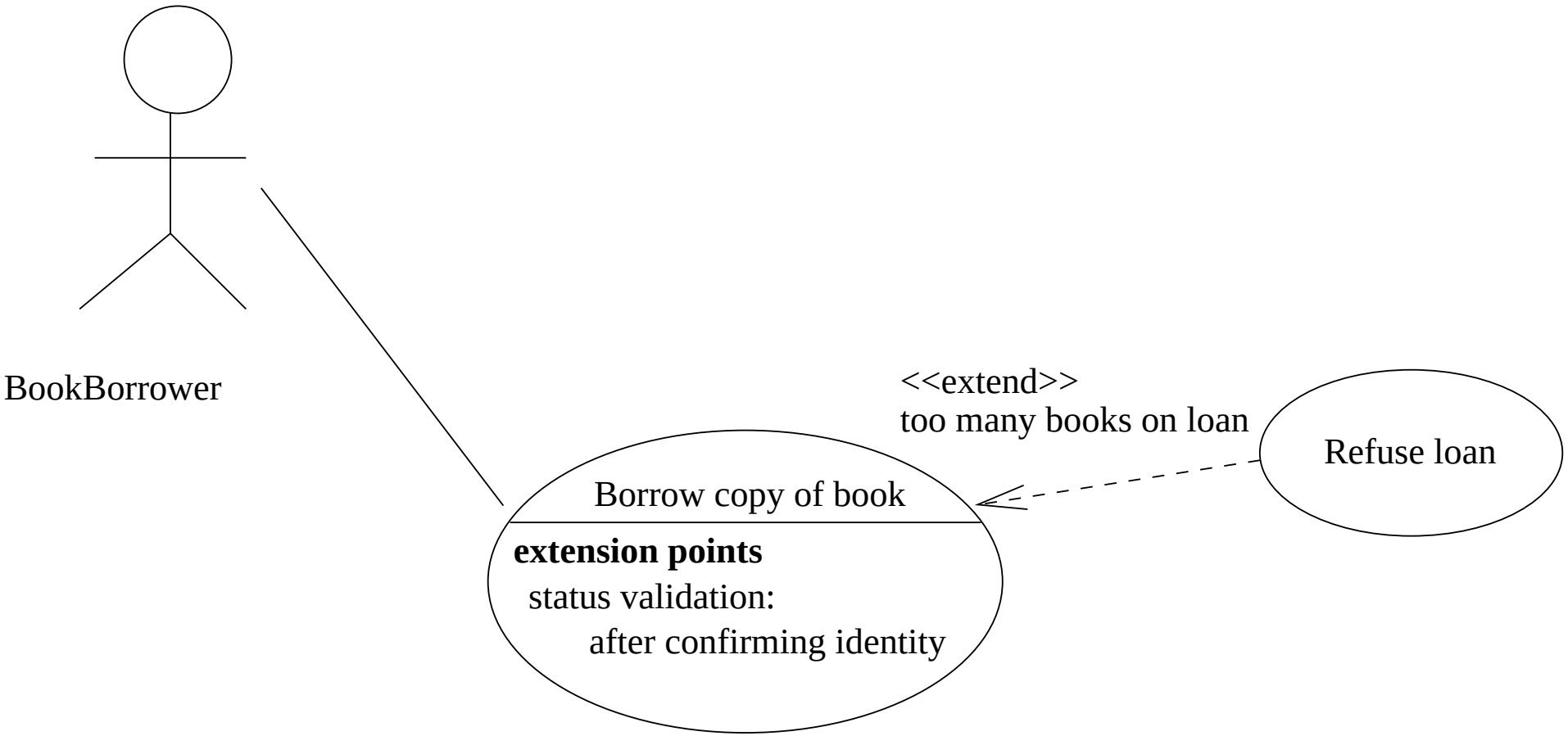


Figure 8.4 ≪extends≫ with extension point.

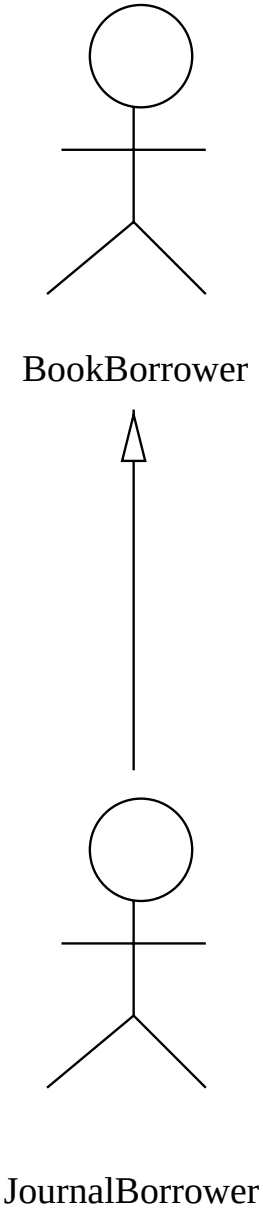
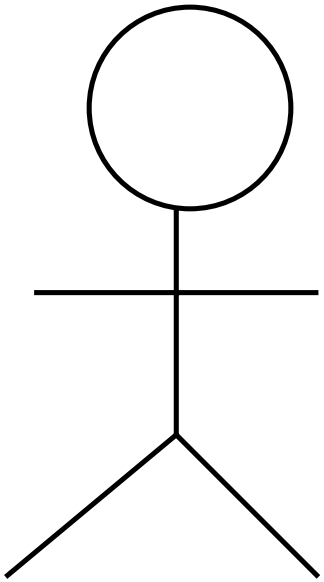


Figure 8.5 Generalization between actors.



BookBorrower

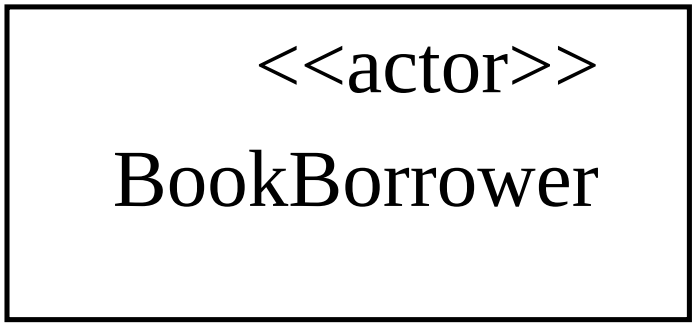


Figure 8.6 These two symbols mean the same.

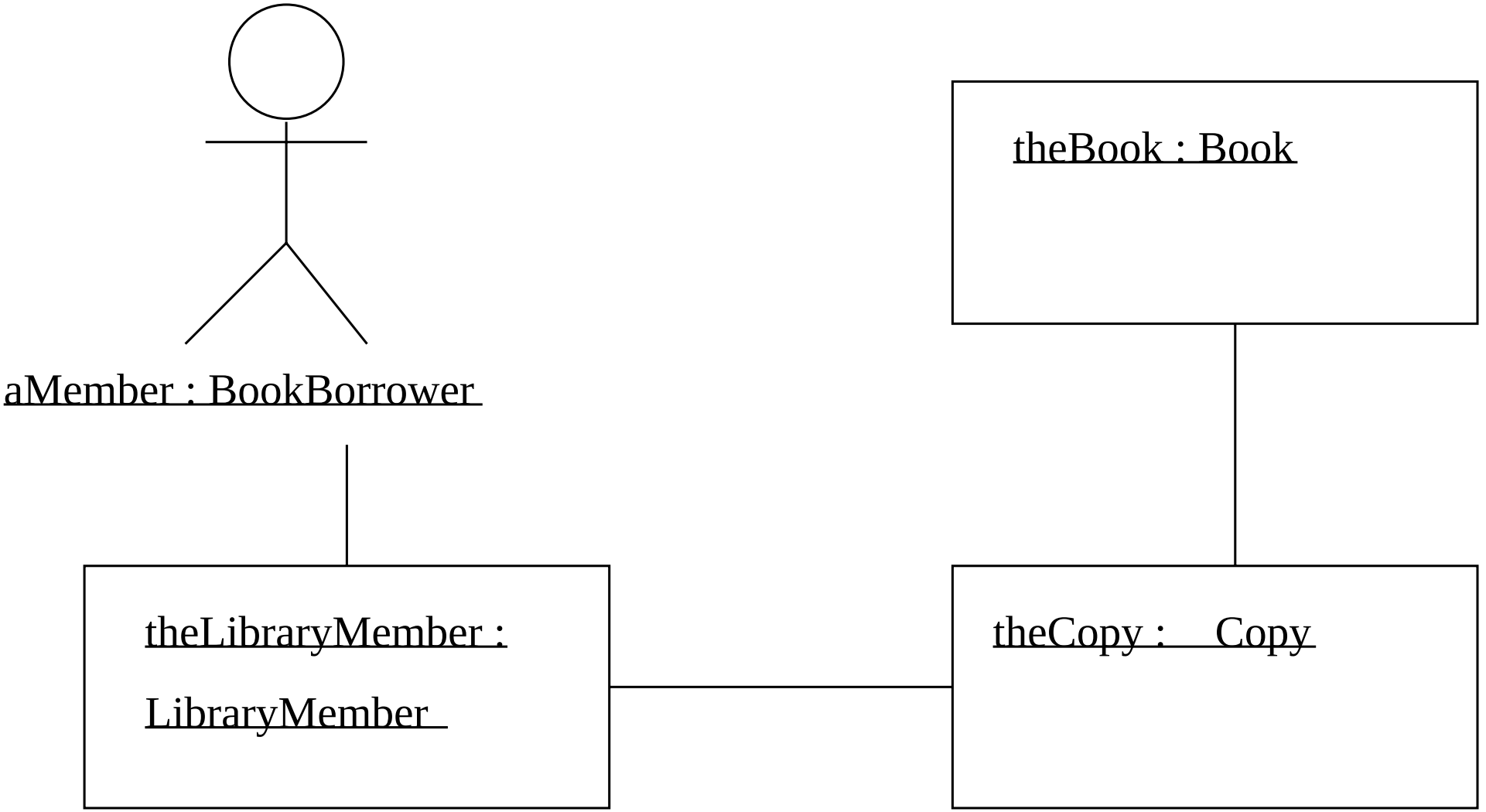


Figure 9.1 A simple collaboration, showing no interaction.

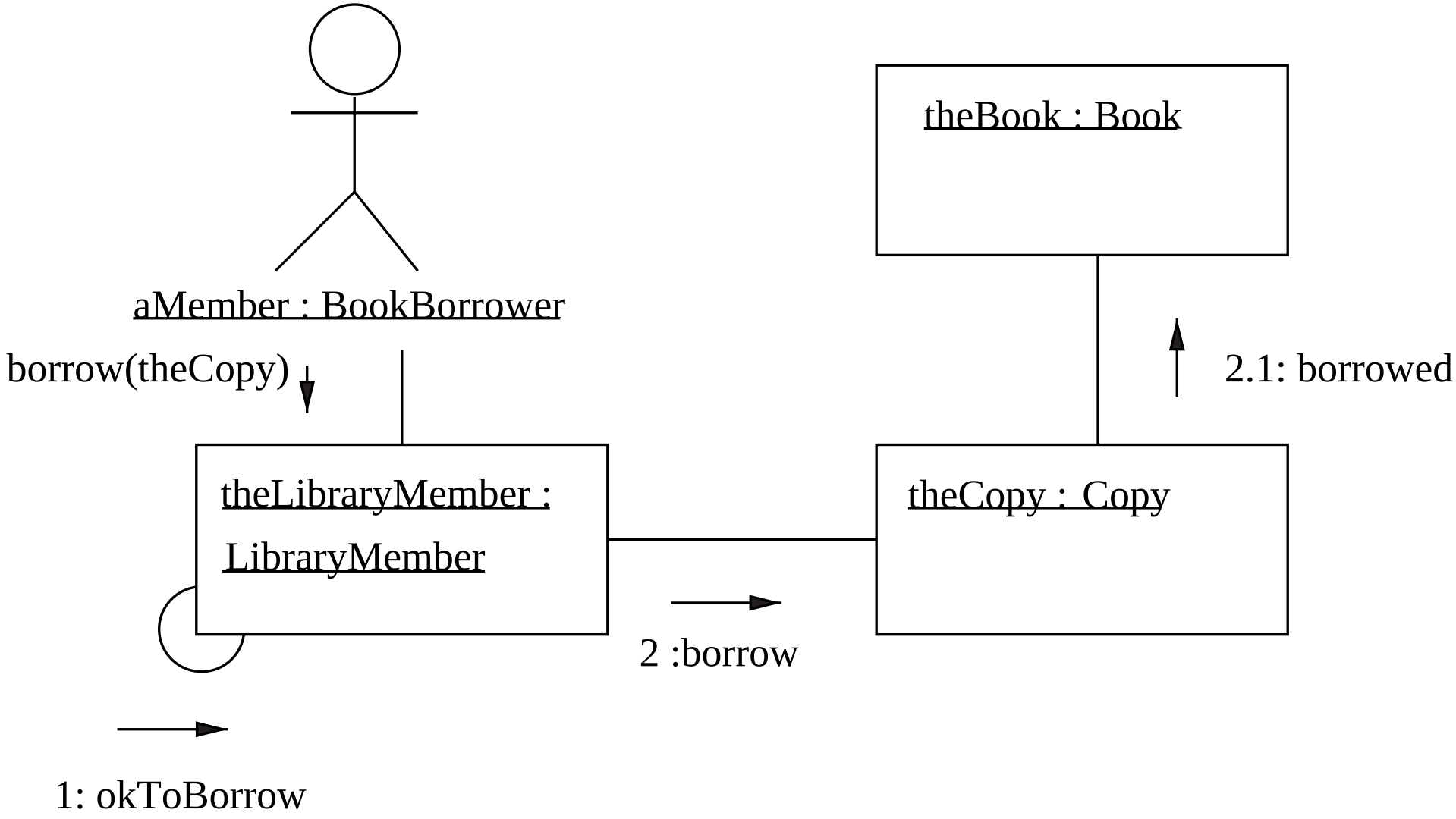


Figure 9.2 Interaction shown on a collaboration diagram.

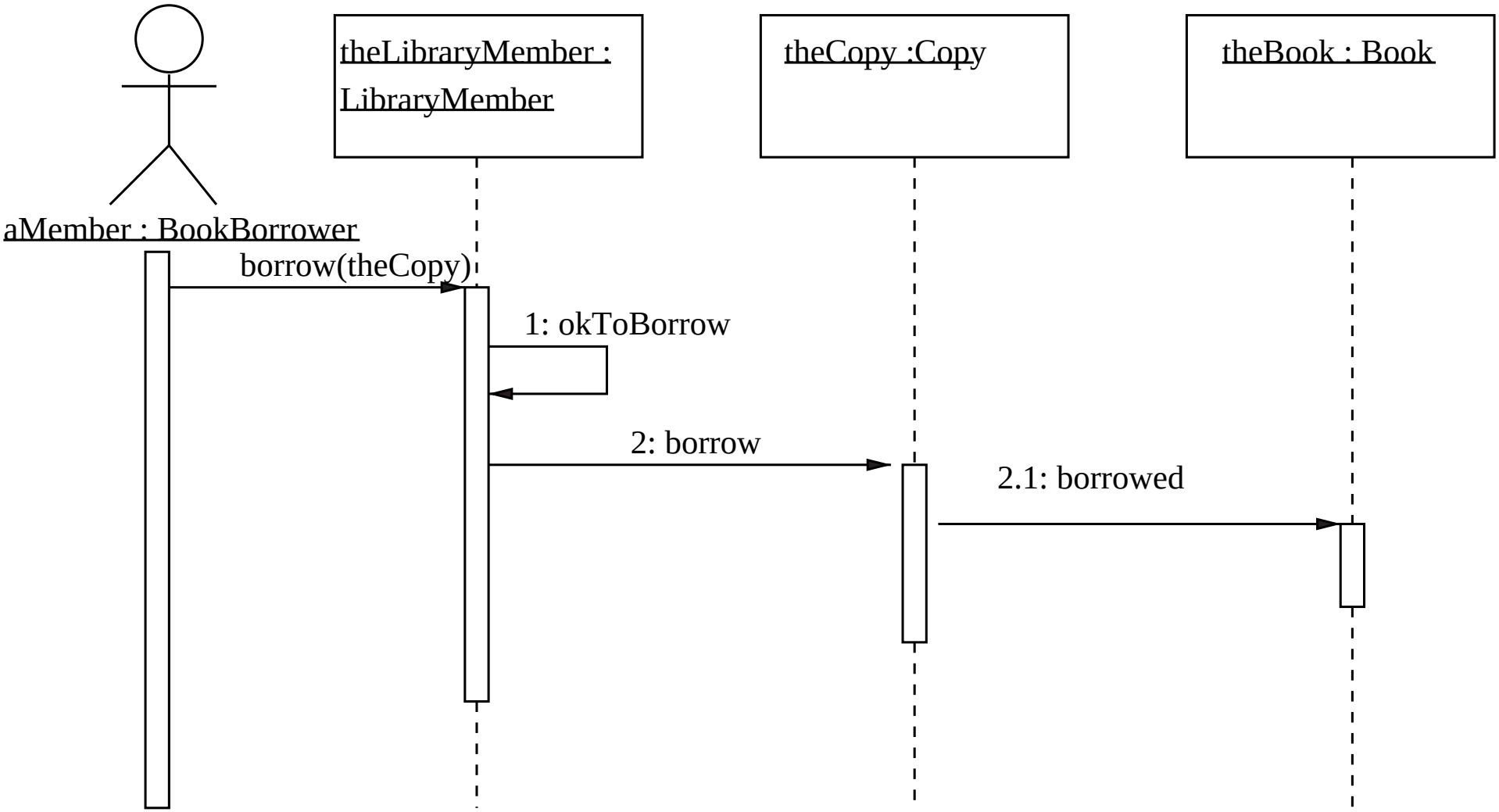


Figure 9.3 Interaction shown on a sequence diagram.

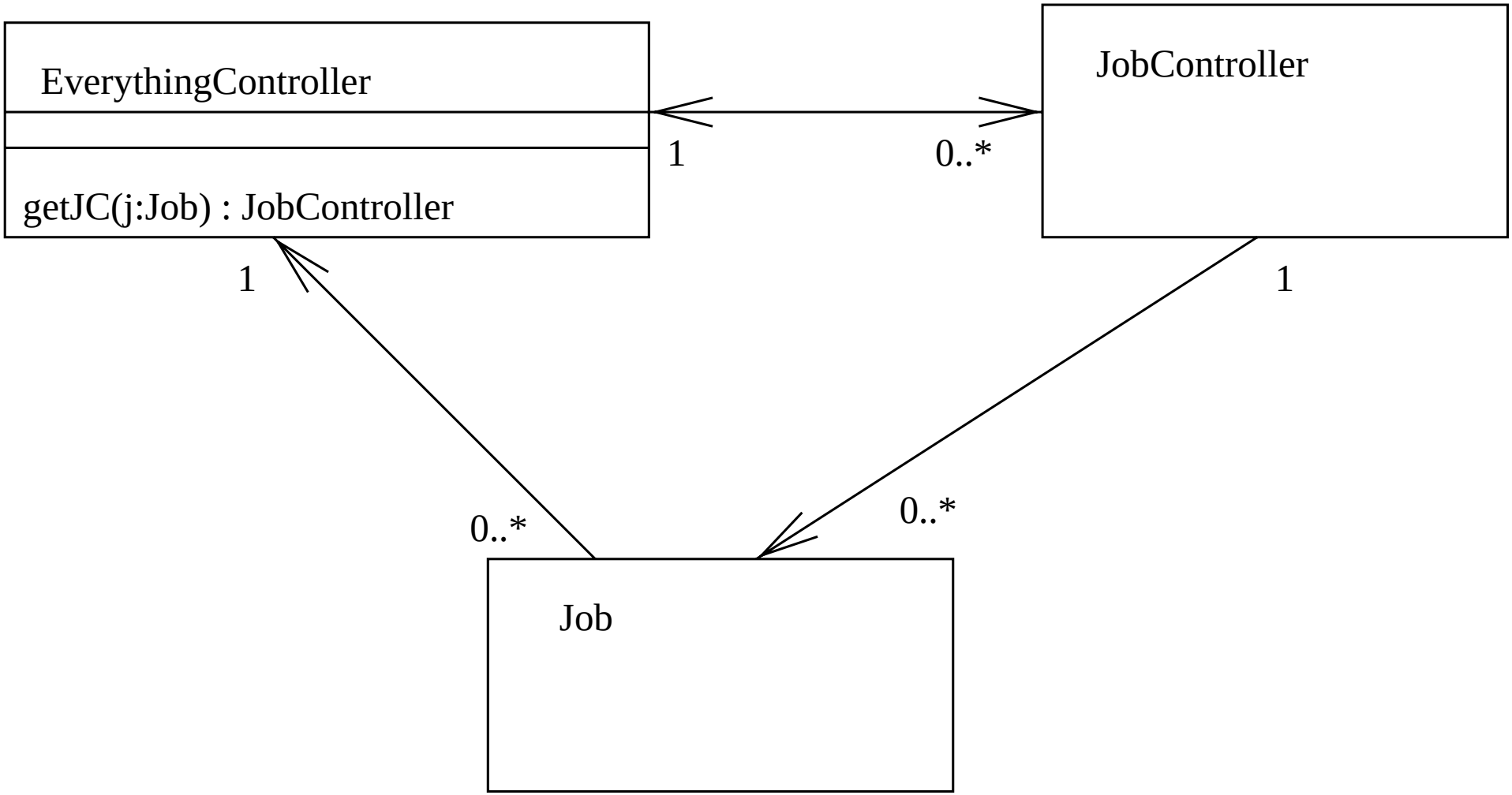


Figure 9.4 Bad design, breaking the Law of Demeter.

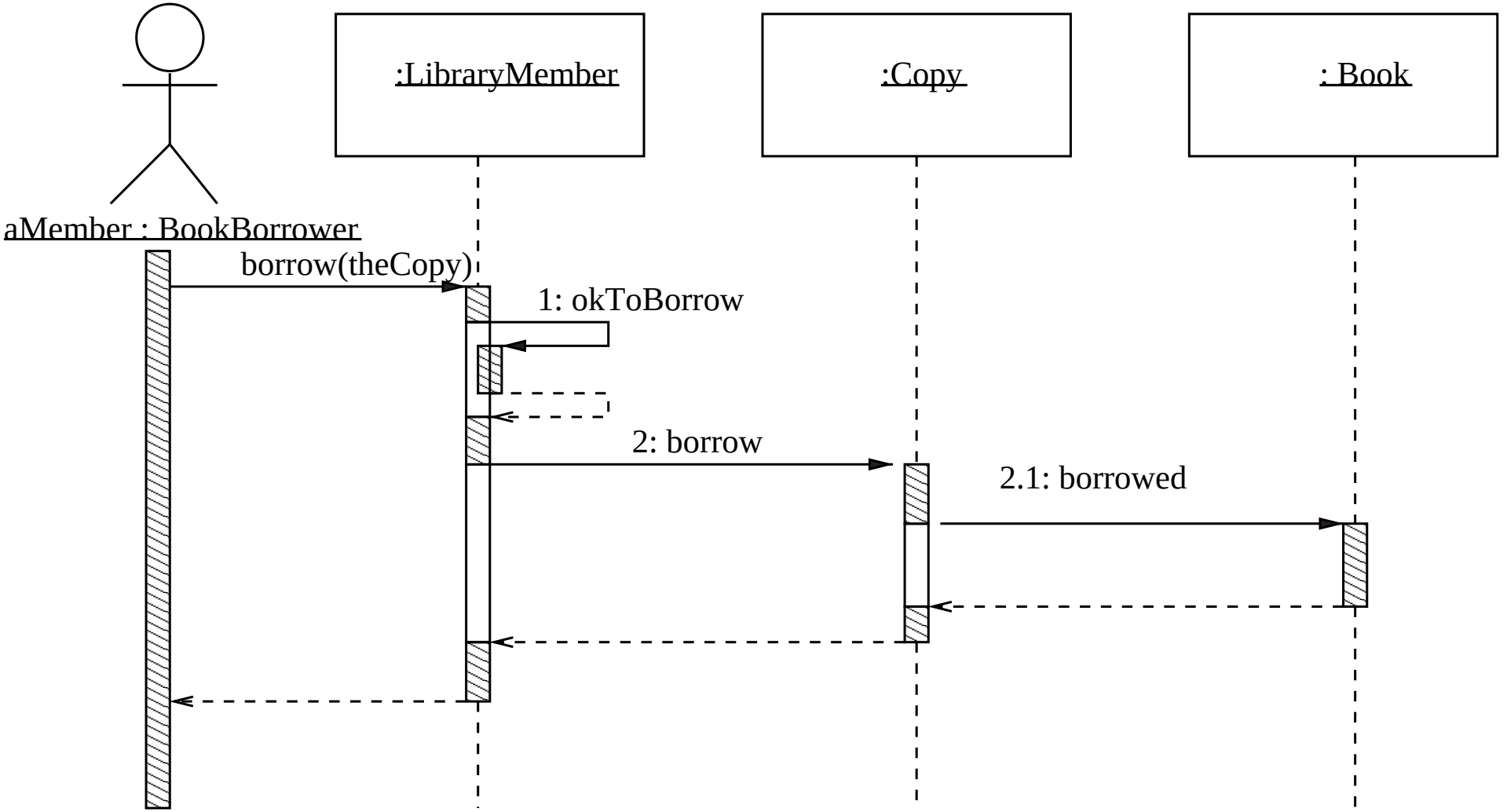


Figure 9.5 Interaction shown on a sequence diagram, with optional features.

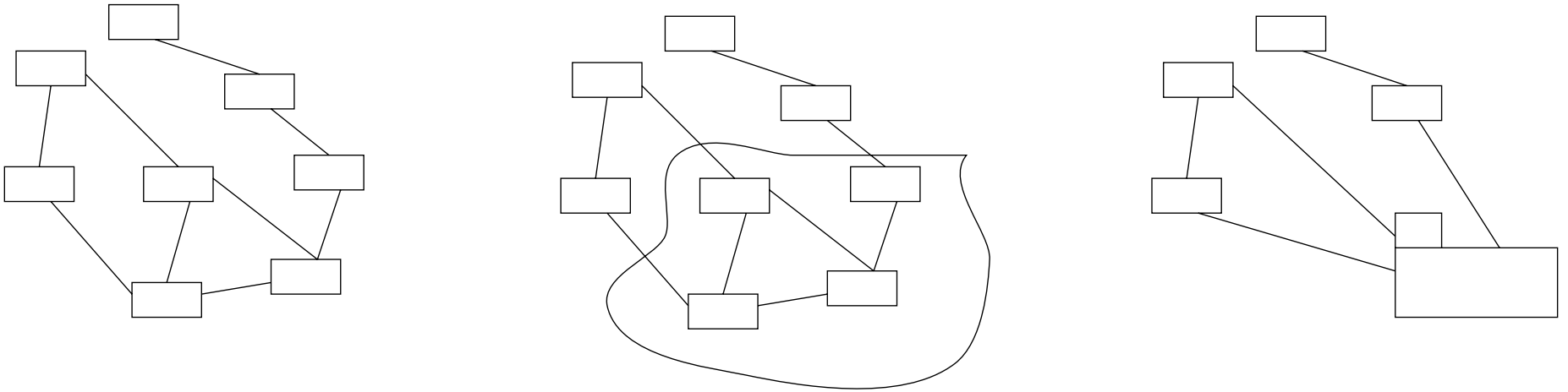


Figure 9.6 Using a package to simplify a collaboration.

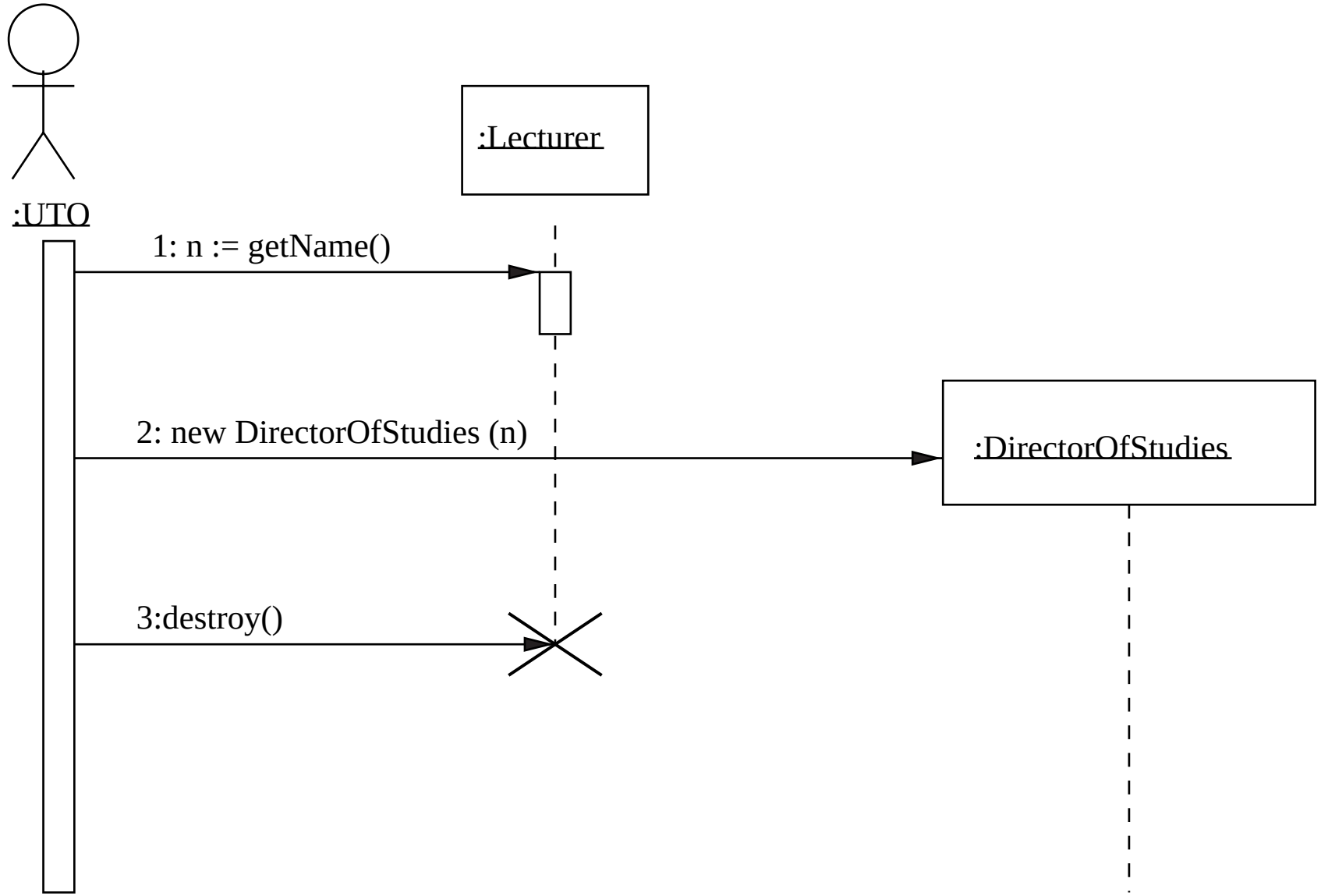


Figure 9.7 Sequence diagram: creation and deletion of objects, and use of return value.

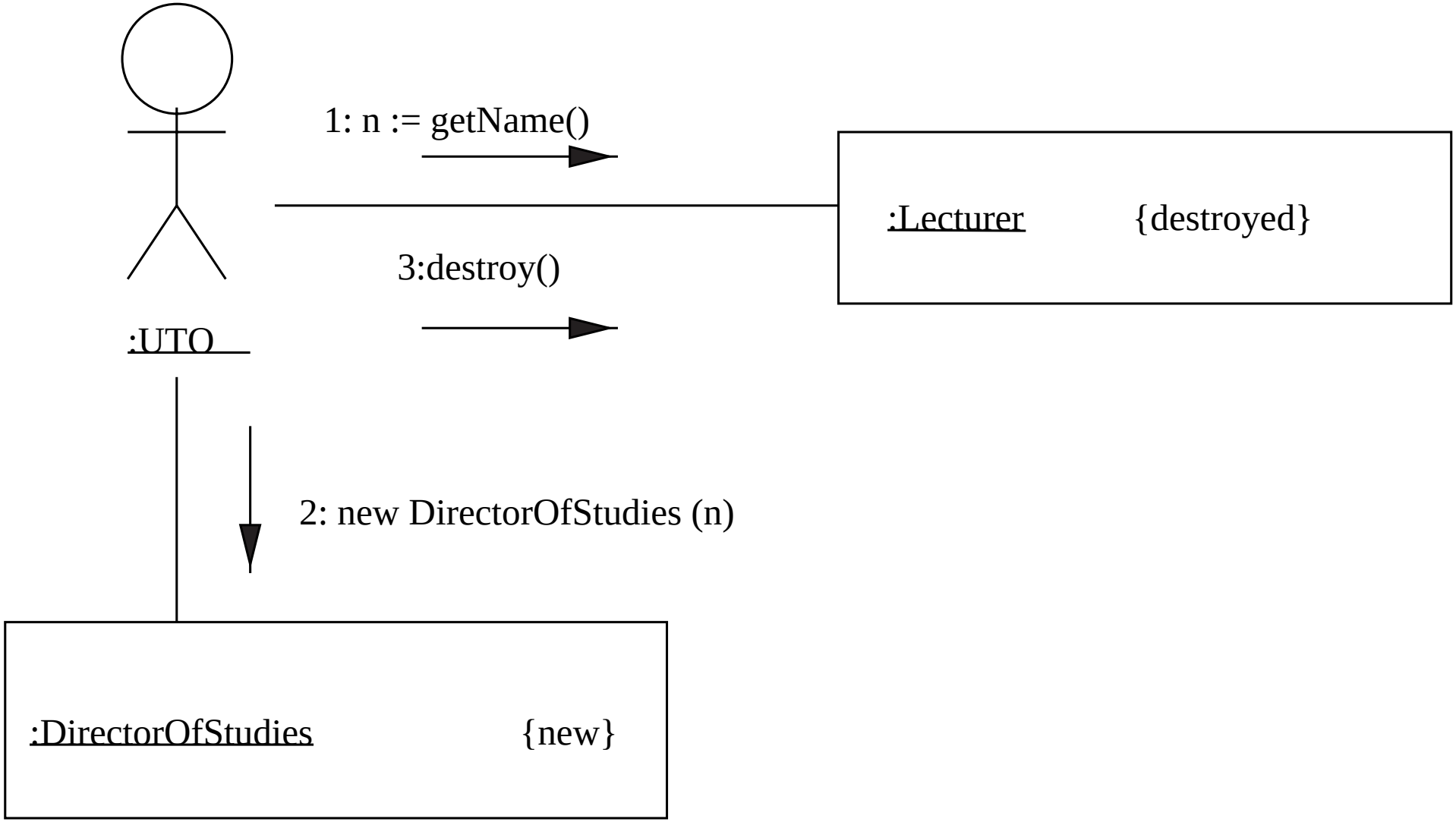


Figure 9.8 Collaboration diagram: creation and deletion of objects, and use of return value.

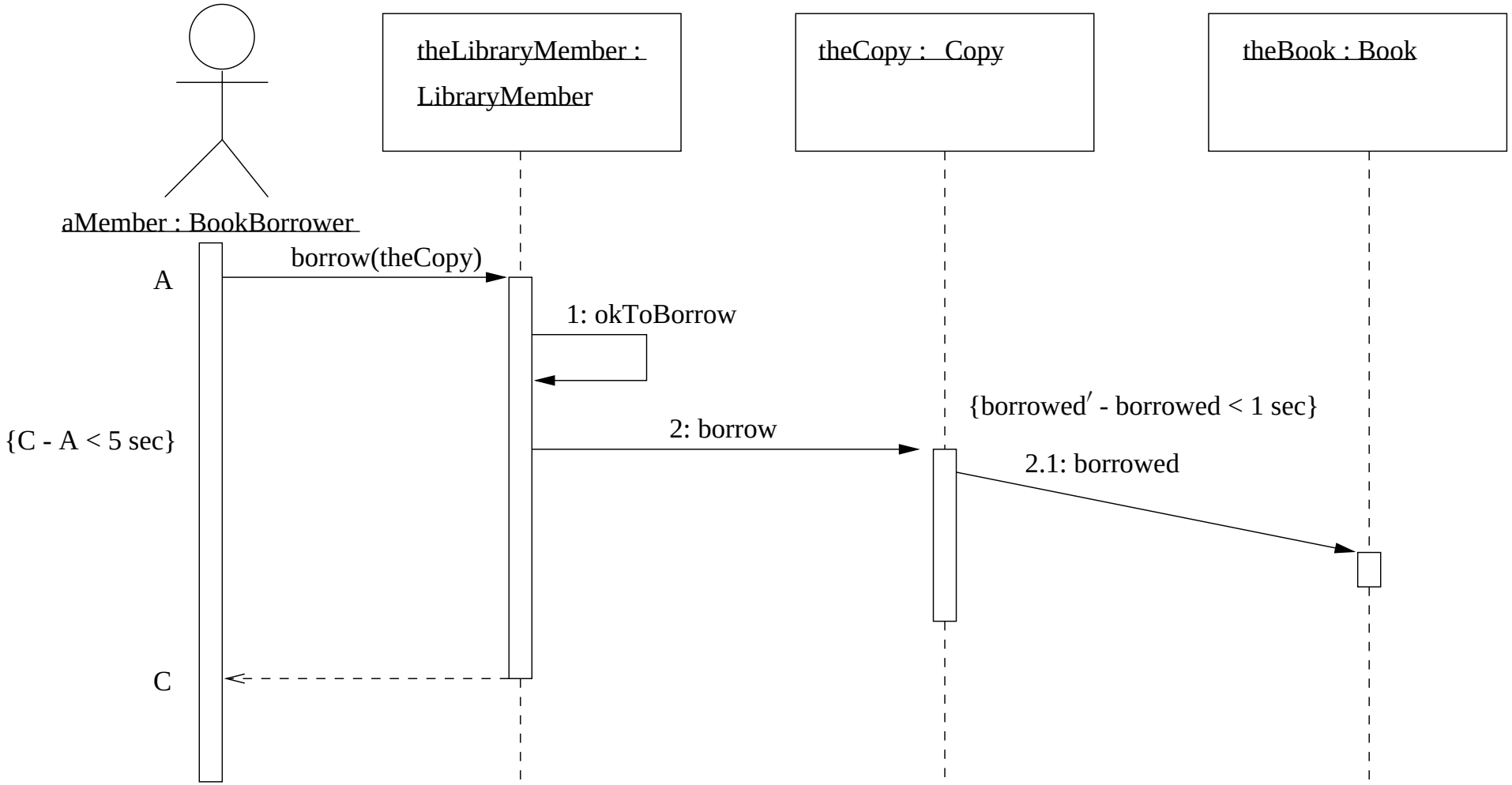


Figure 9.9 Showing timing constraints on a sequence diagram.

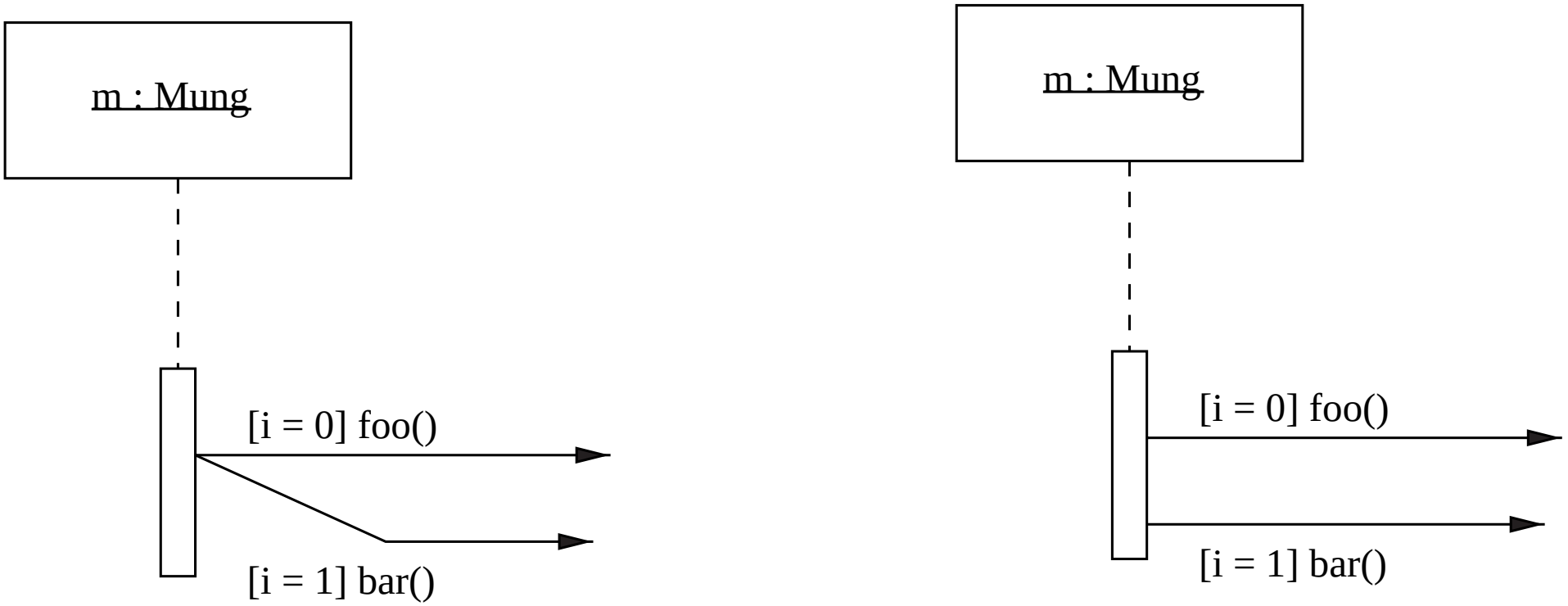


Figure 10.1 Two sequence diagram fragments.

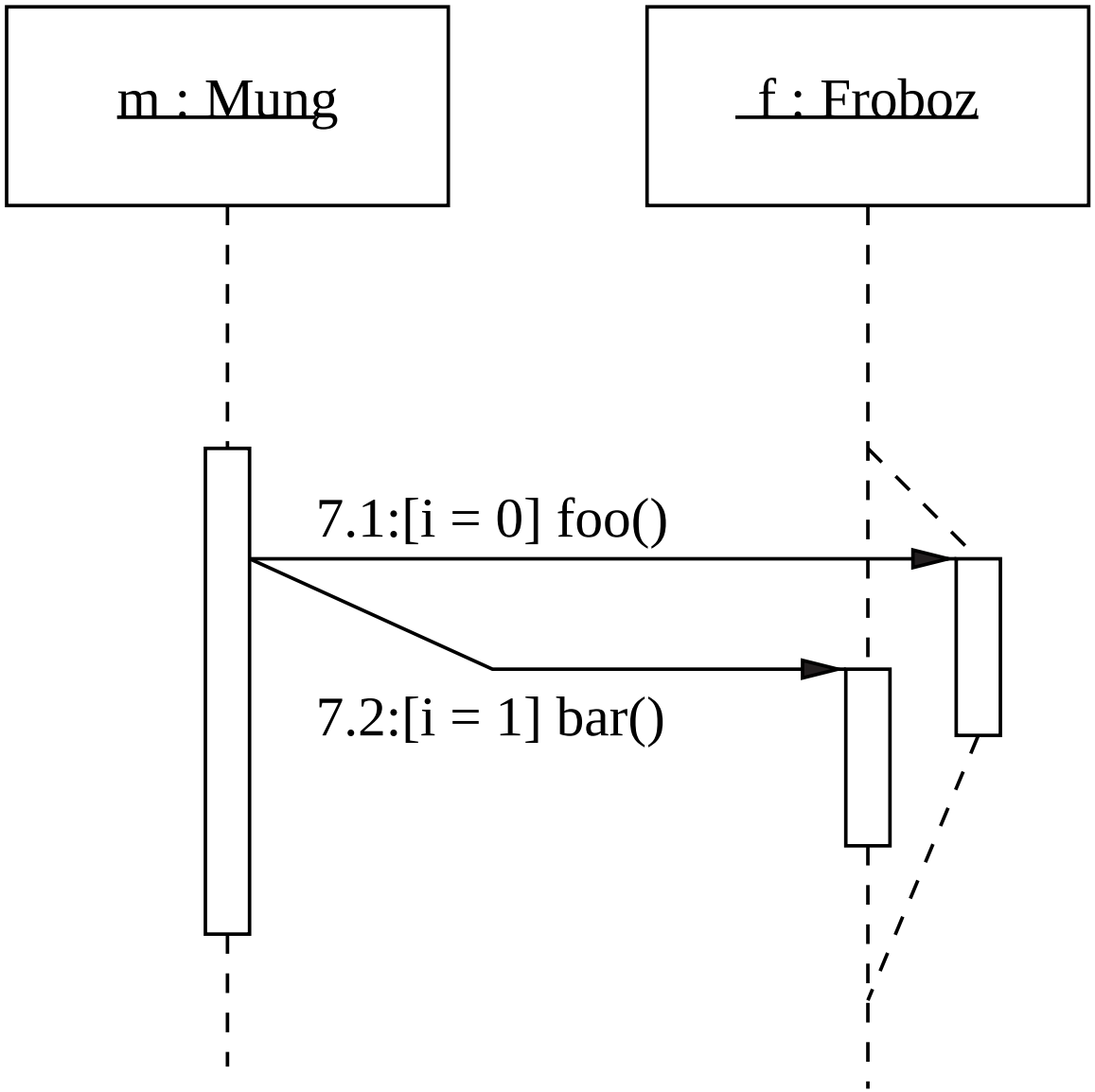


Figure 10.2 Fragment of sequence diagram with branching lifeline.

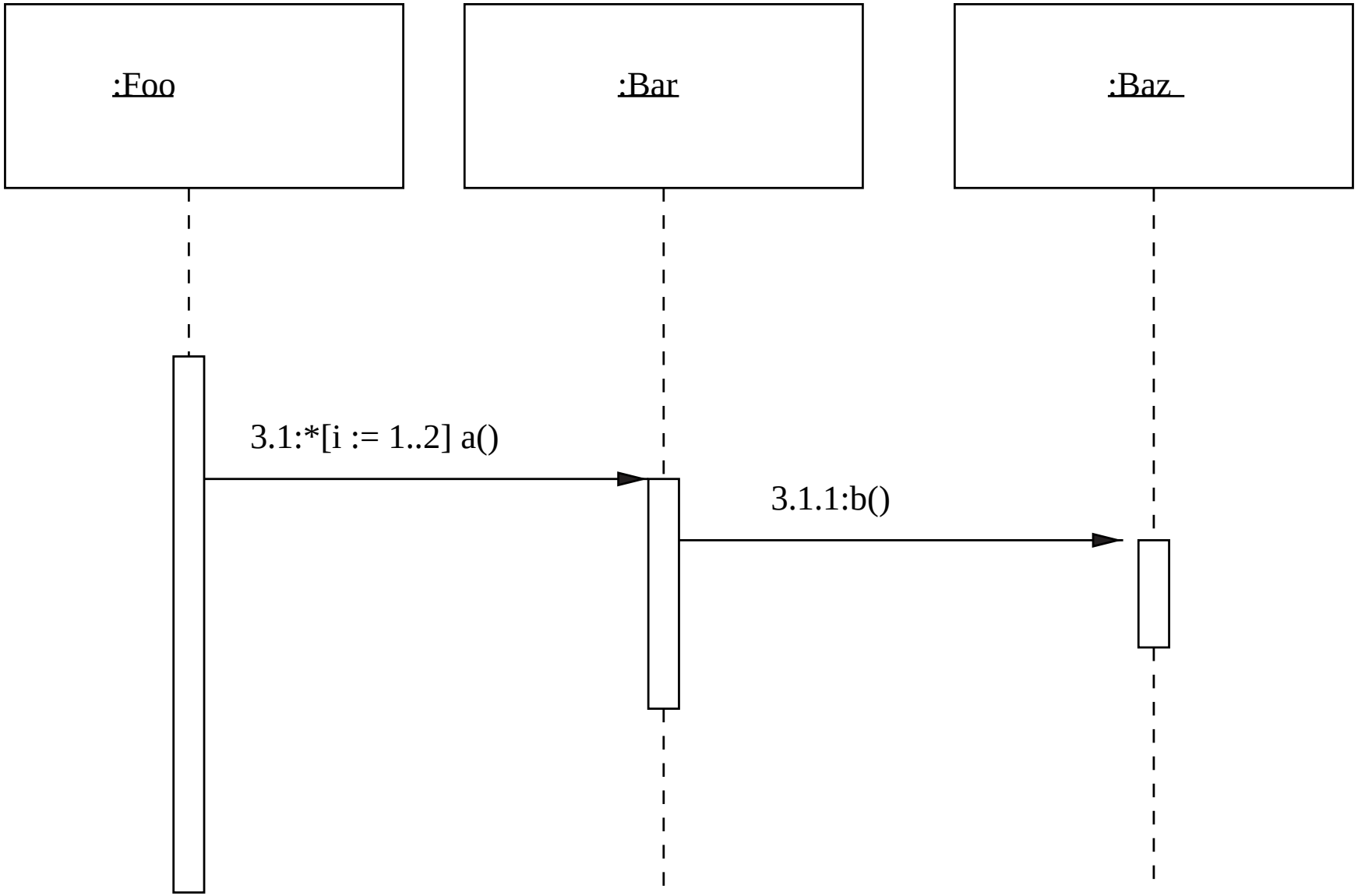


Figure 10.3 Sequence diagram fragment: iteration showing messages `abab`.

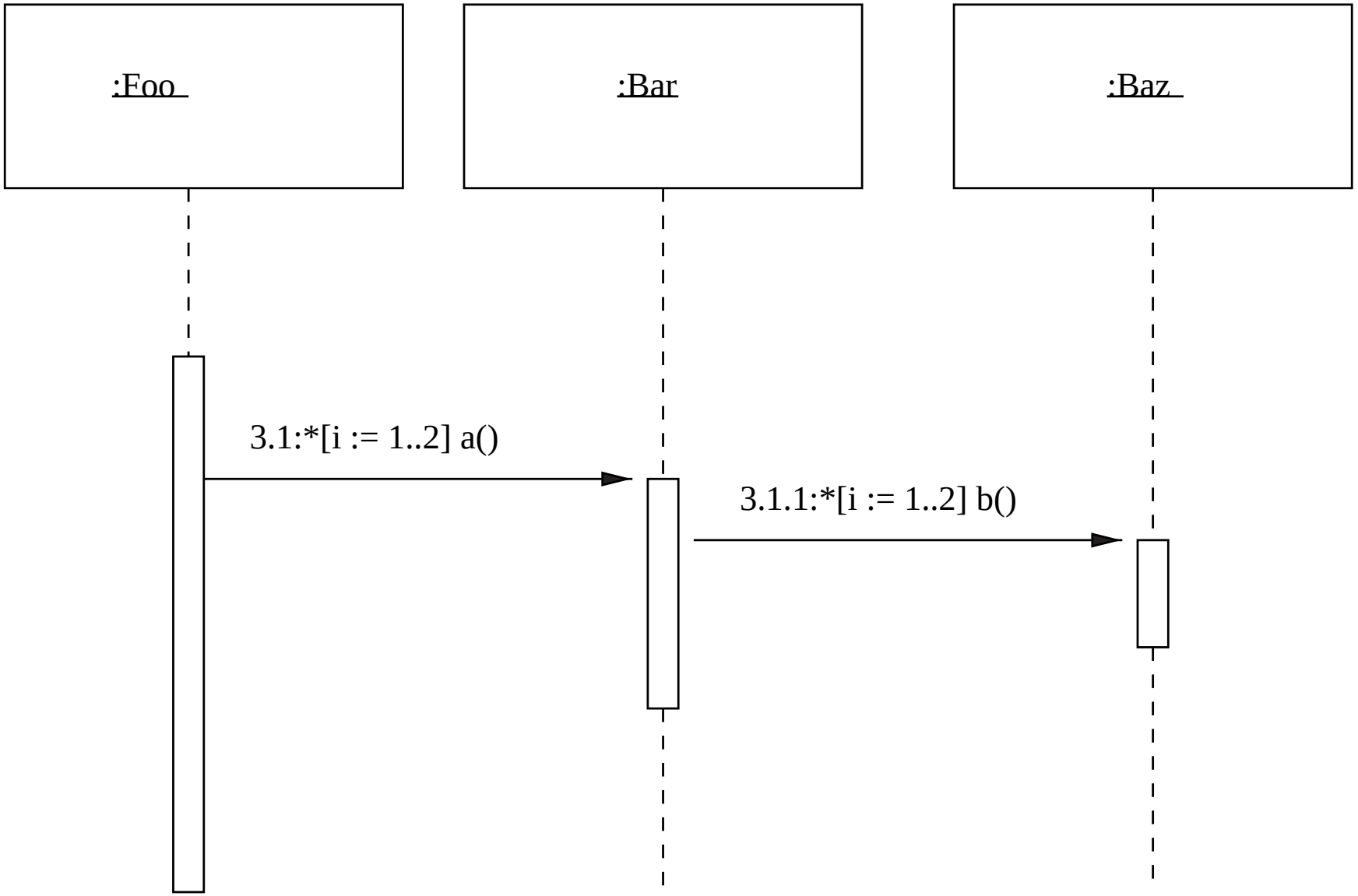


Figure 10.4 Sequence diagram fragment: iteration showing messages abbabb.

Interaction type	Symbol	Meaning
Synchronous or call	–▶	The 'normal' procedural situation. The sender loses control until the receiver finishes handling the message, then gets control back, which can optionally be shown as a return arrow.
Return	<– –	Not a message, but a return from an earlier message. Unblocks a synchronous send.
Flat	→	The message doesn't expect a reply; control passes from the sender to the receiver, so the next message (in this thread) will be sent by the receiver of this message.
Asynchronous	–›	The message doesn't expect a reply, but unlike the flat case, the sender stays active and may send further messages.

Figure 10.5 Variants of message sending in sequence diagrams.

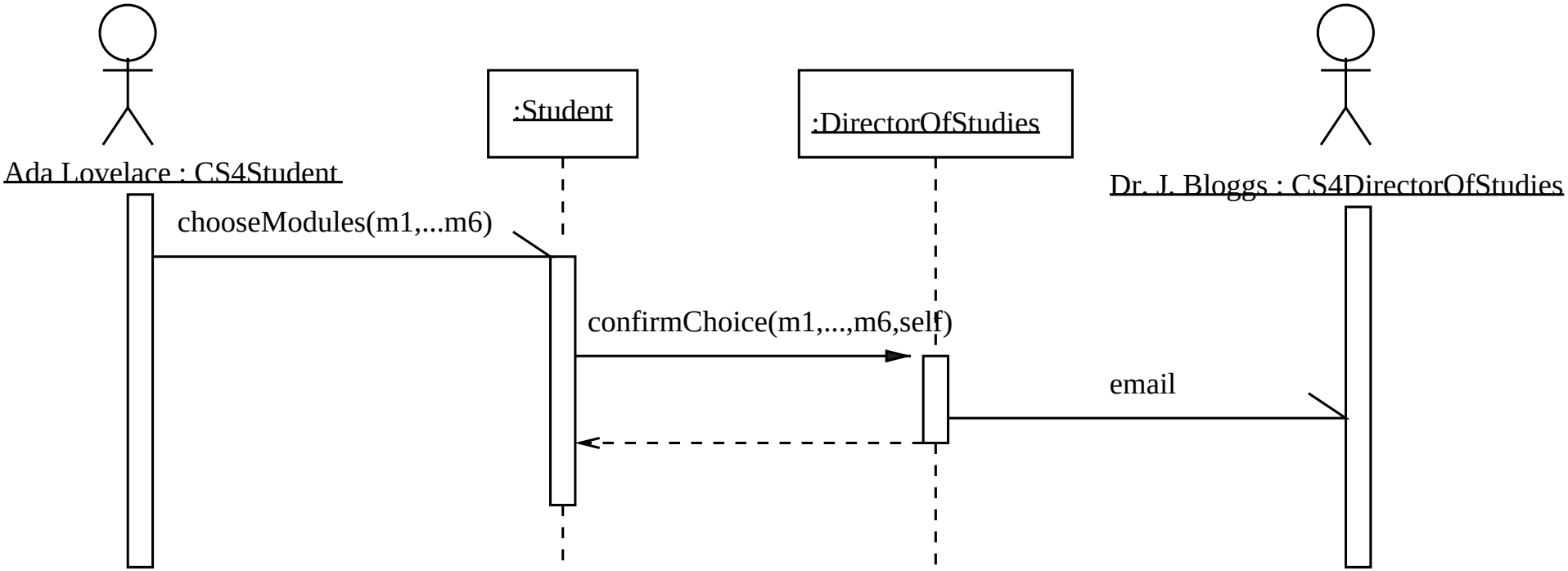


Figure 10.6 Asynchronous message-passing.

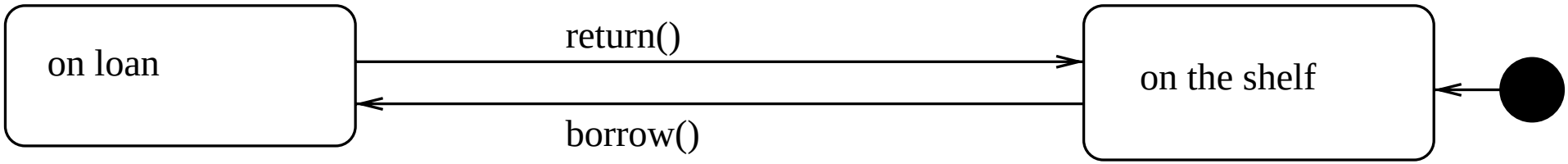


Figure 11.1 State diagram of class `Copy`.

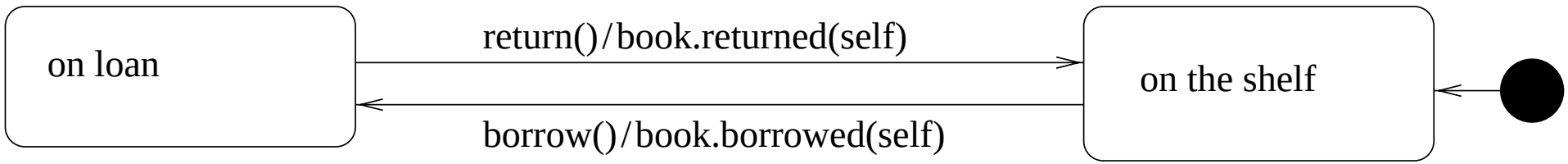


Figure 11.2 State diagram of class `Copy`, with actions.

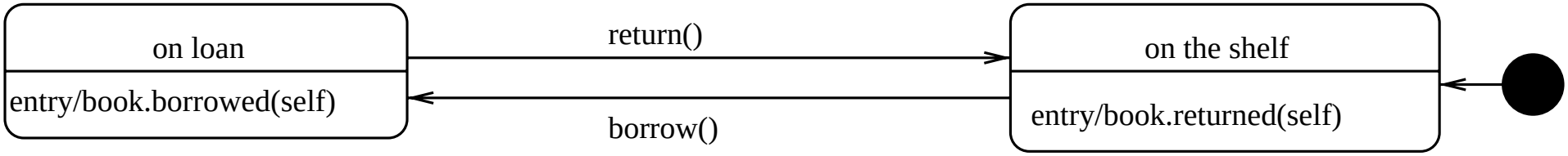


Figure 11.3 State diagram of class `Copy`, with `entry` actions.

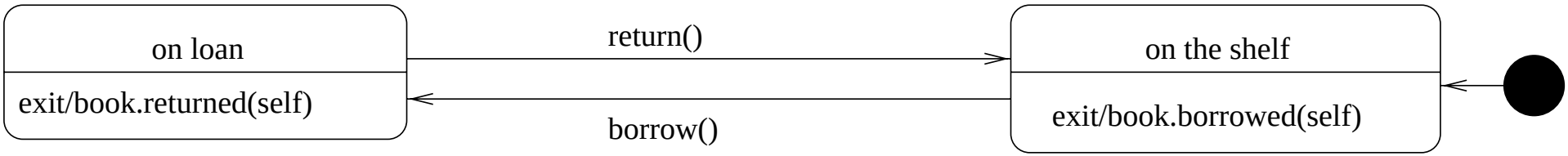


Figure 11.4 State diagram of class `Copy`, with `exit` actions.

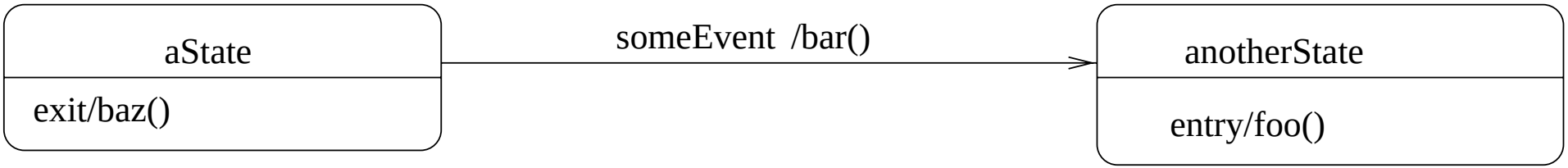


Figure 11.5 Several actions in one diagram.

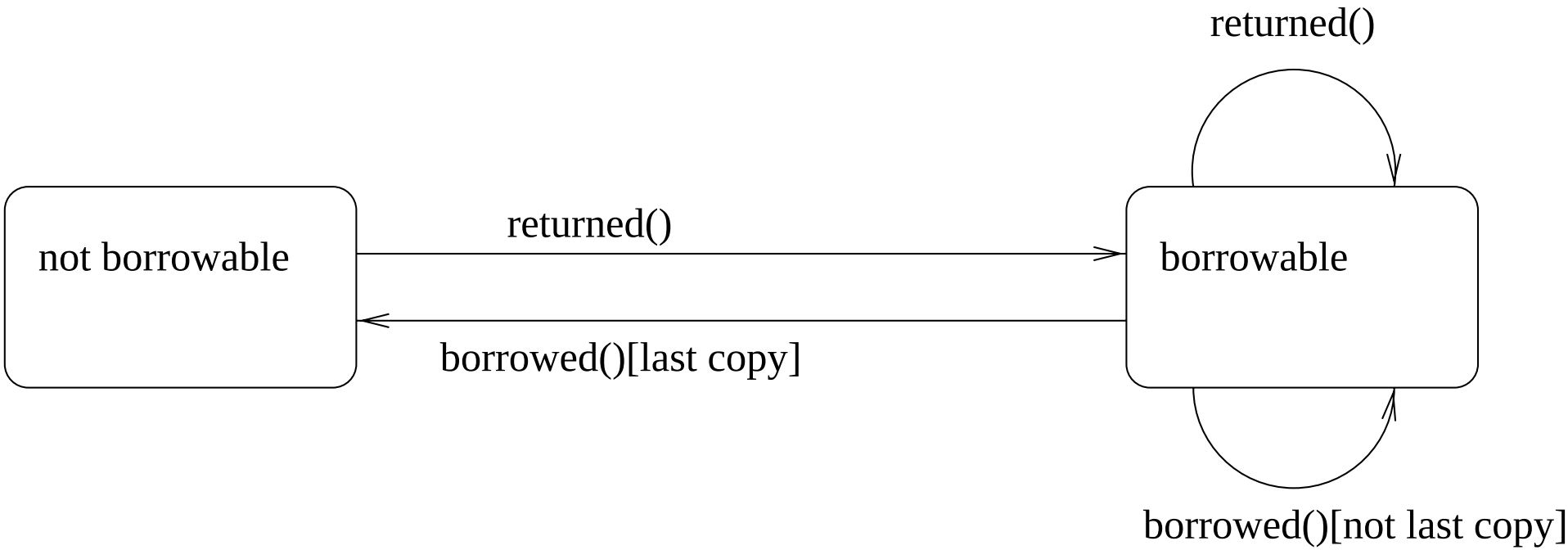


Figure 11.6 State diagram for class `Book`.

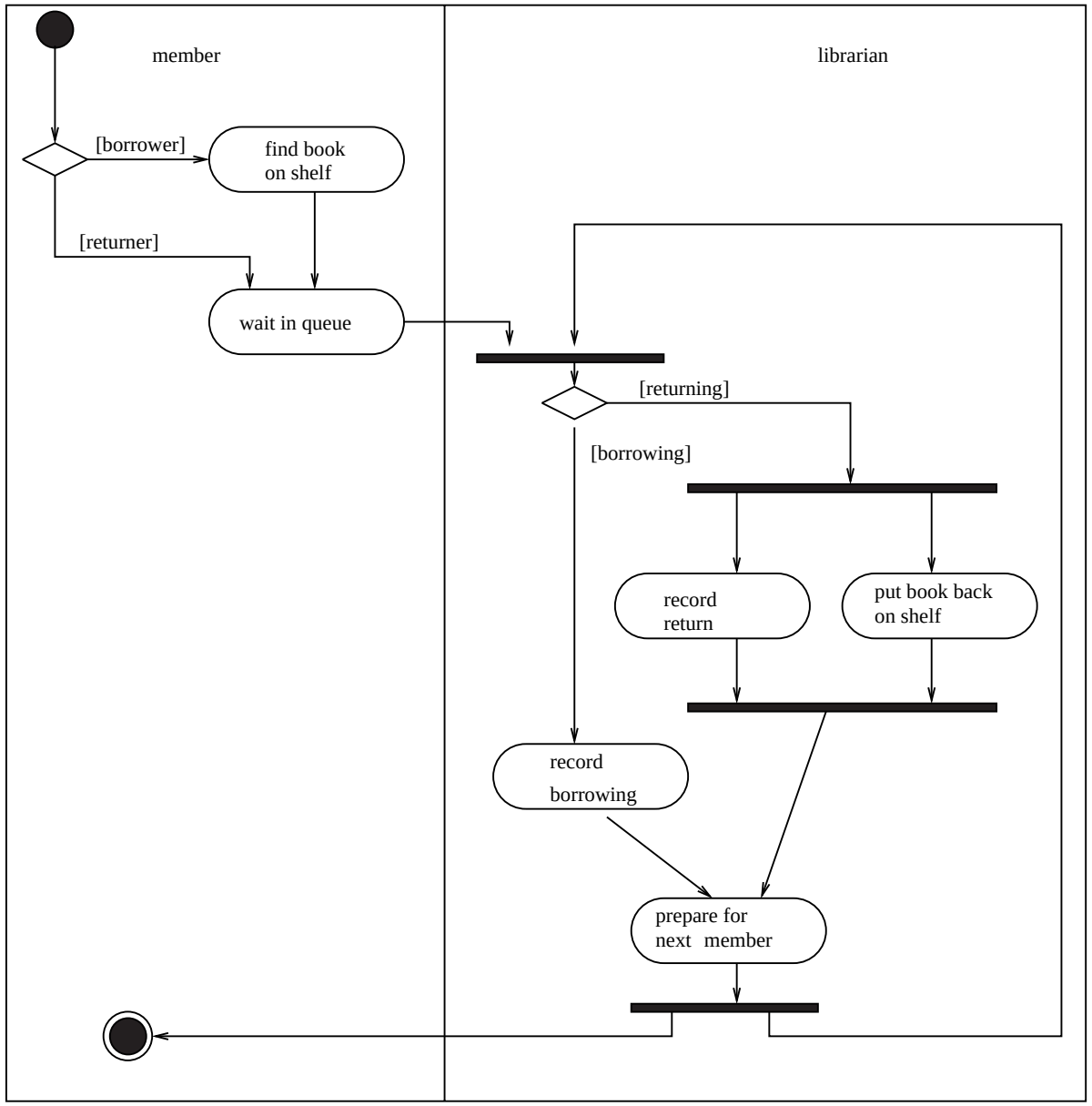


Figure 11.7 Business level activity diagram of the library.

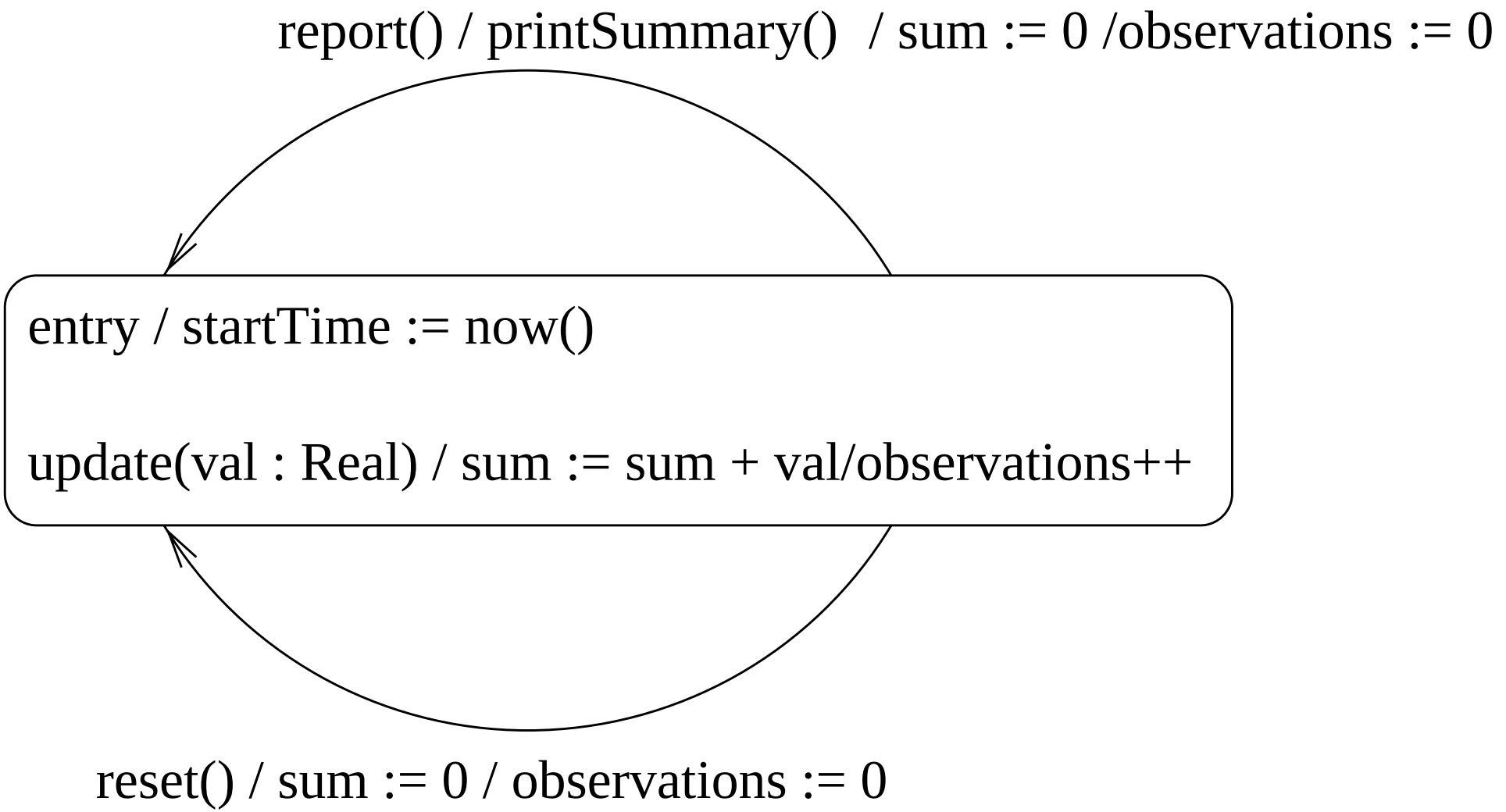


Figure 12.1 State diagram for class `Average`: not good style!

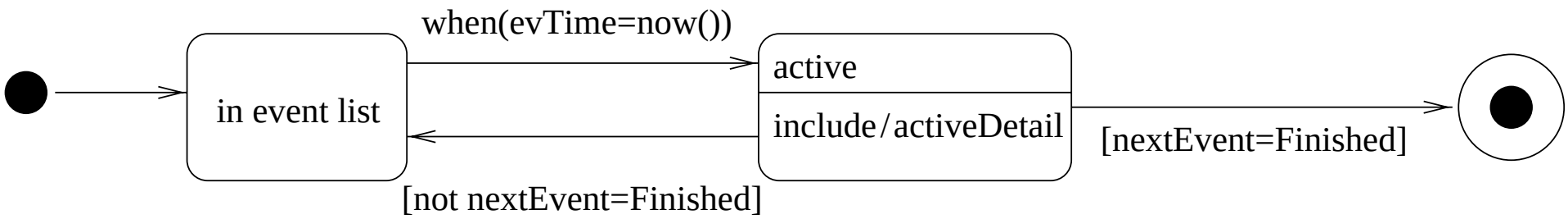


Figure 12.2 State diagram for class `Customer`.

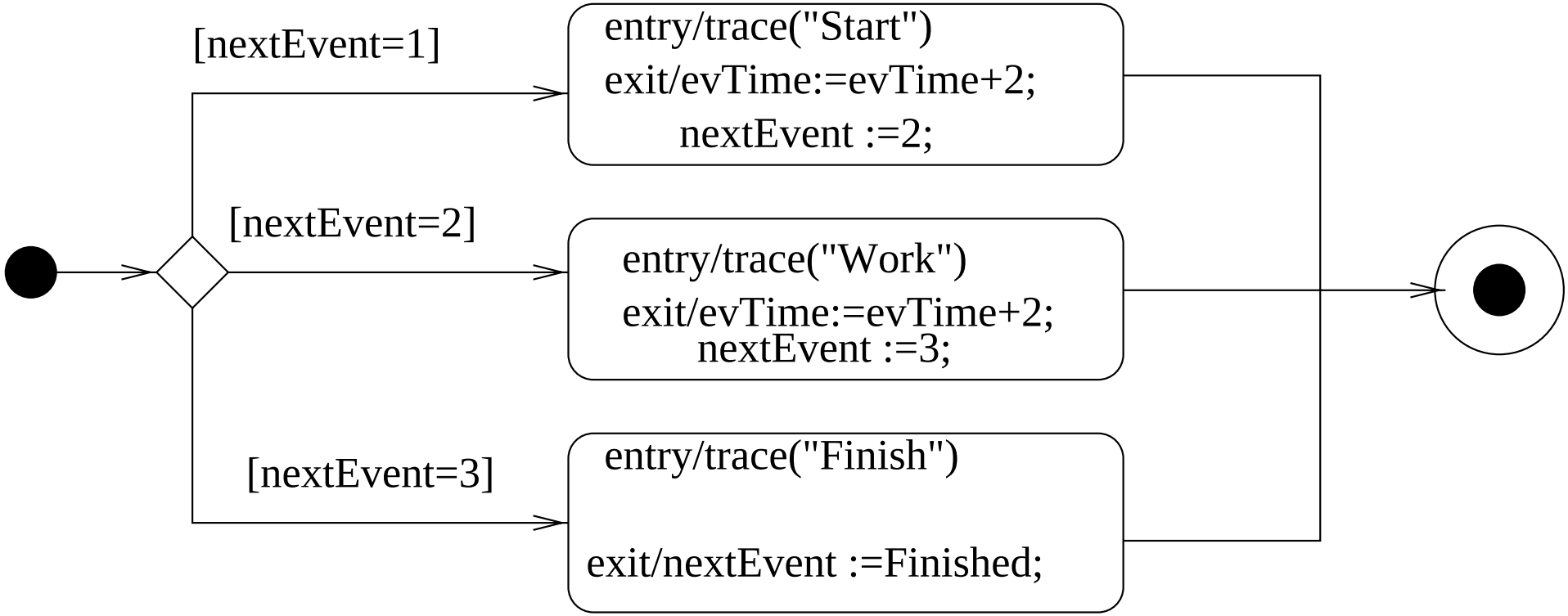
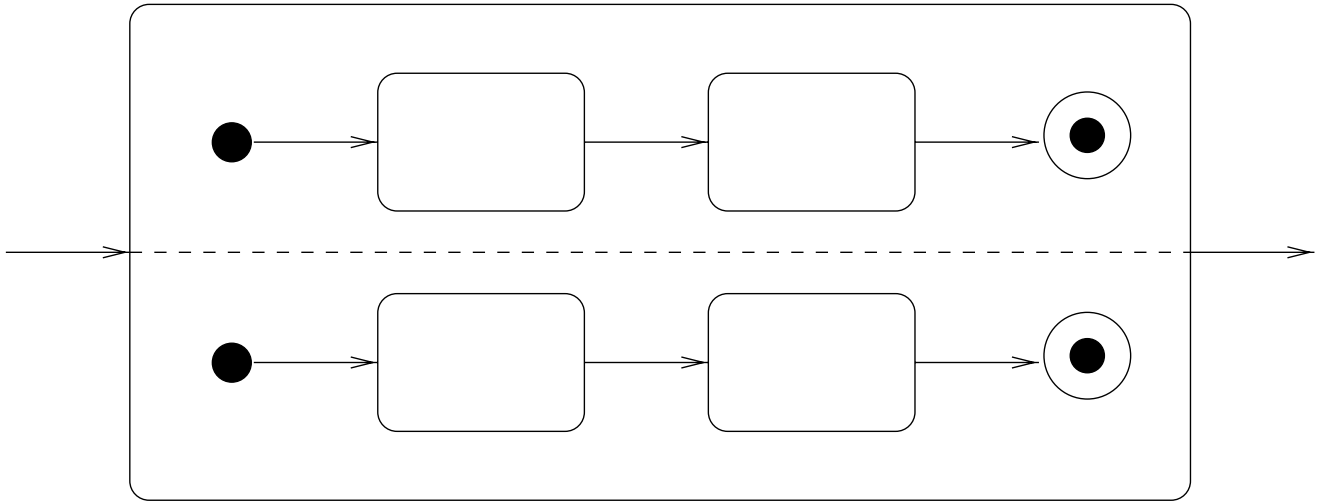
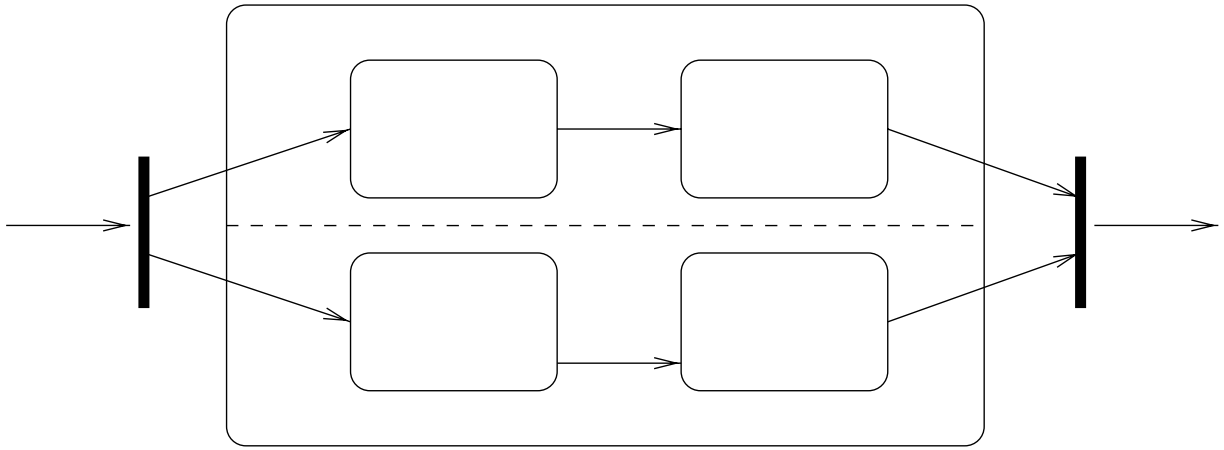


Figure 12.3 Nested state diagram `activeDetail` for class `Customer`'s `active` state.



(a) State with internal concurrency



(b) Equivalent state with external synchronization

Figure 12.4 State diagrams with concurrency.

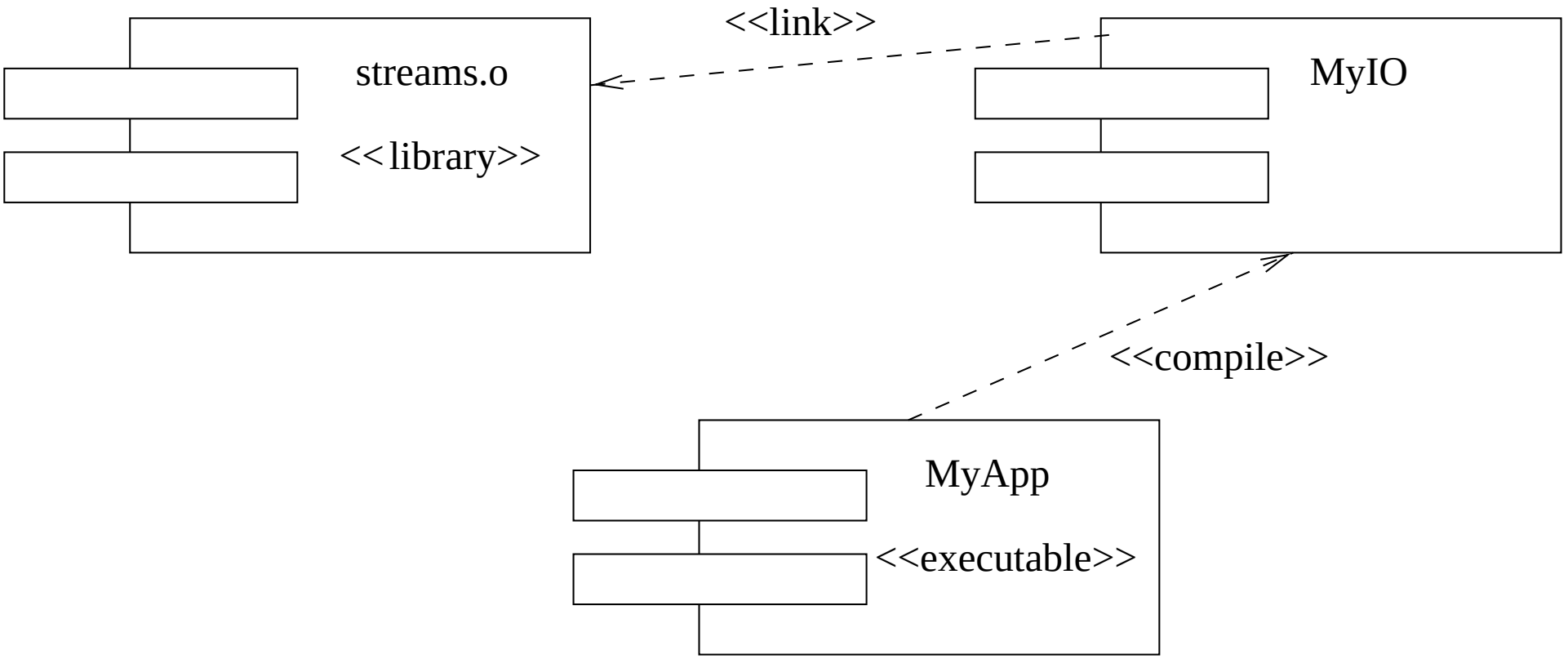


Figure 13.1 A component diagram showing compile time dependencies.

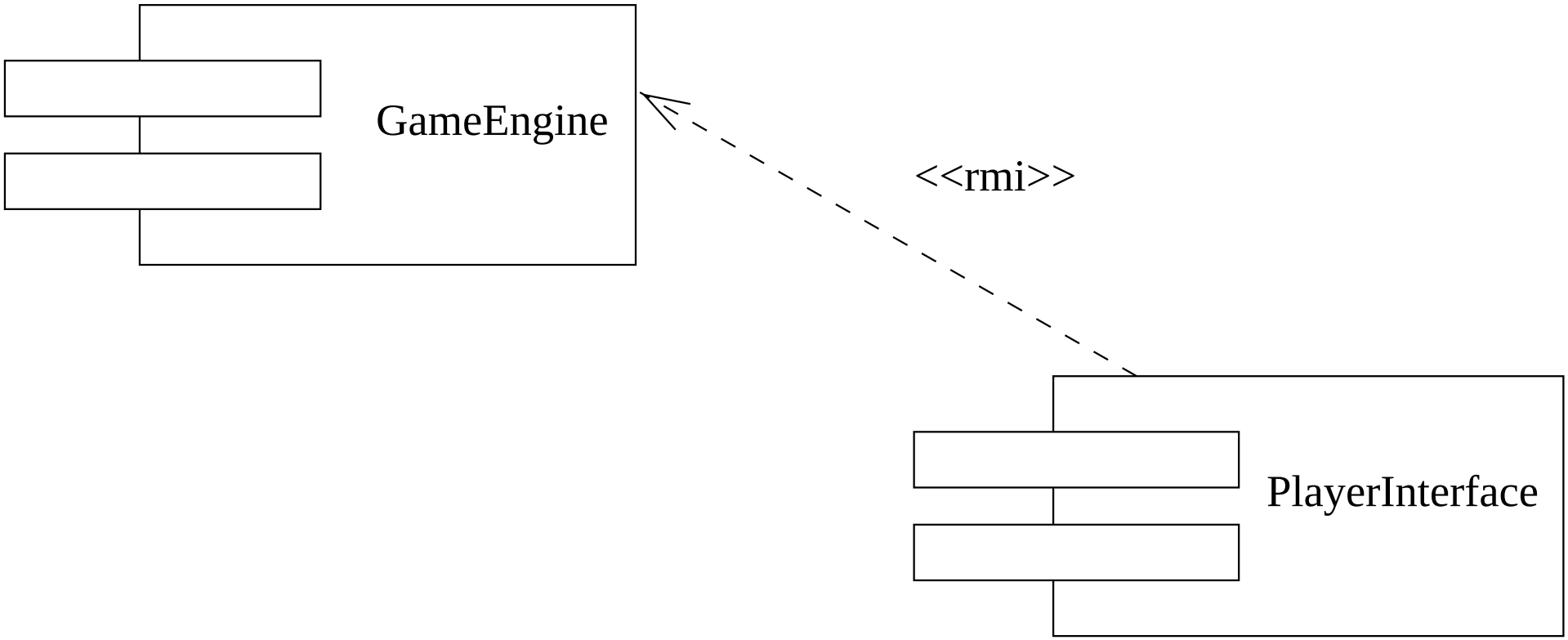


Figure 13.2 A component diagram showing runtime dependencies.

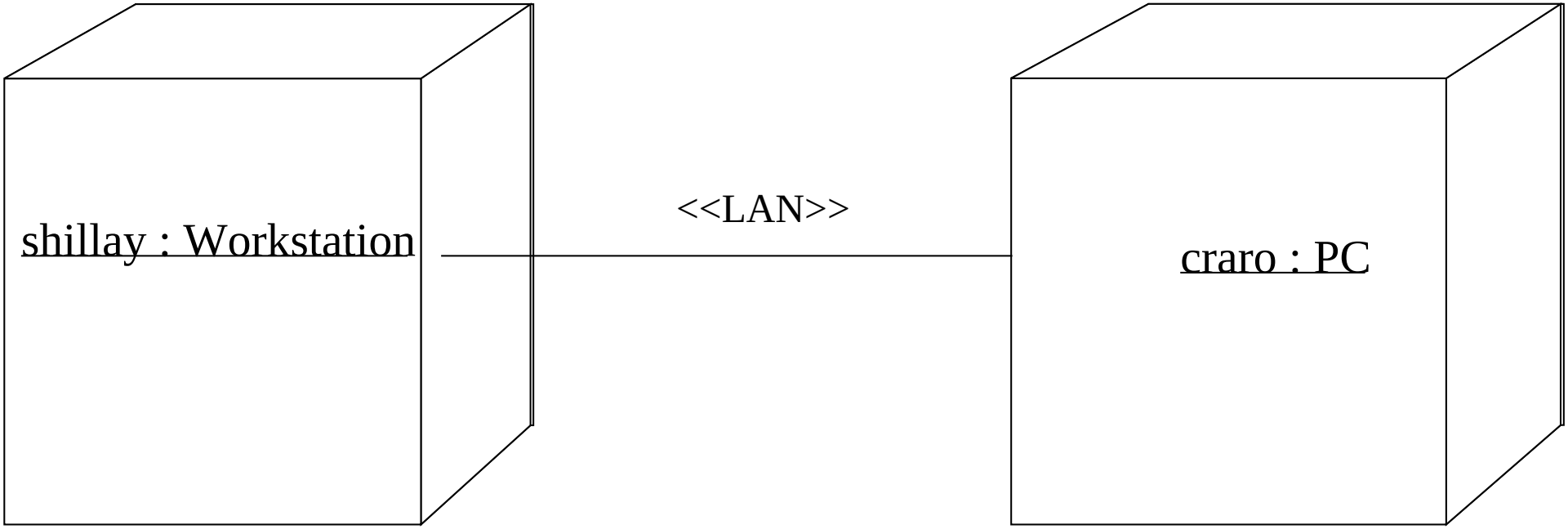


Figure 13.3 A deployment diagram without the software.

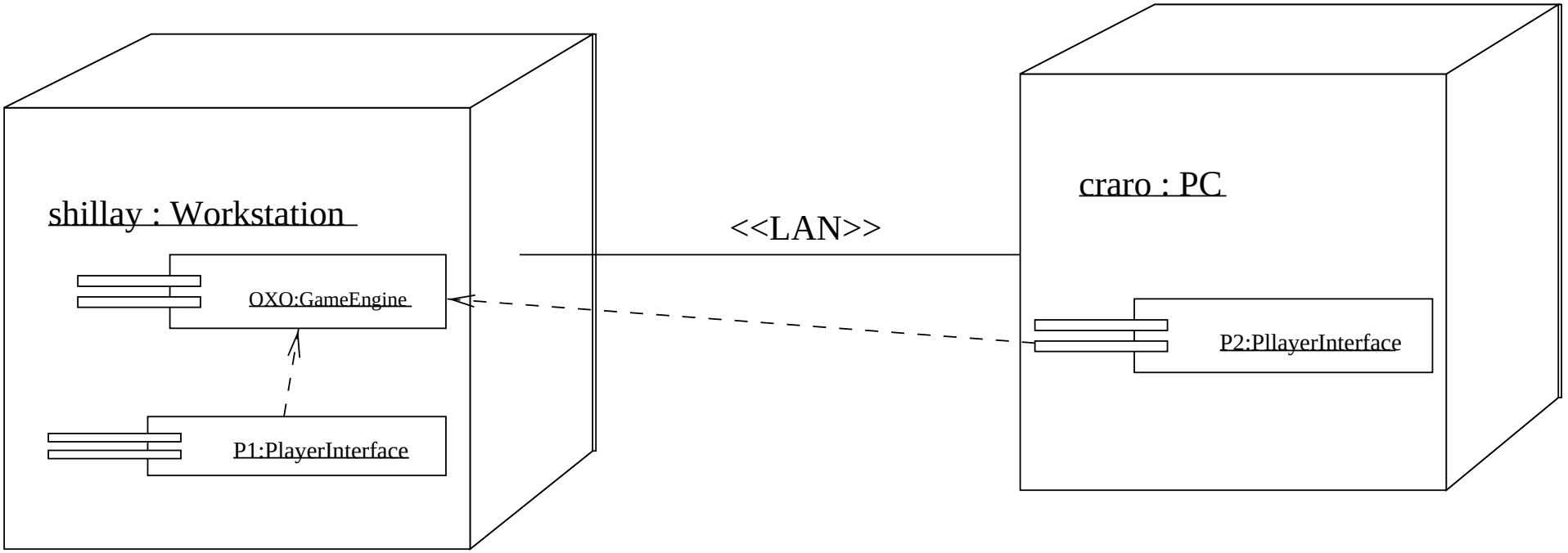


Figure 13.4 A deployment diagram with the software.

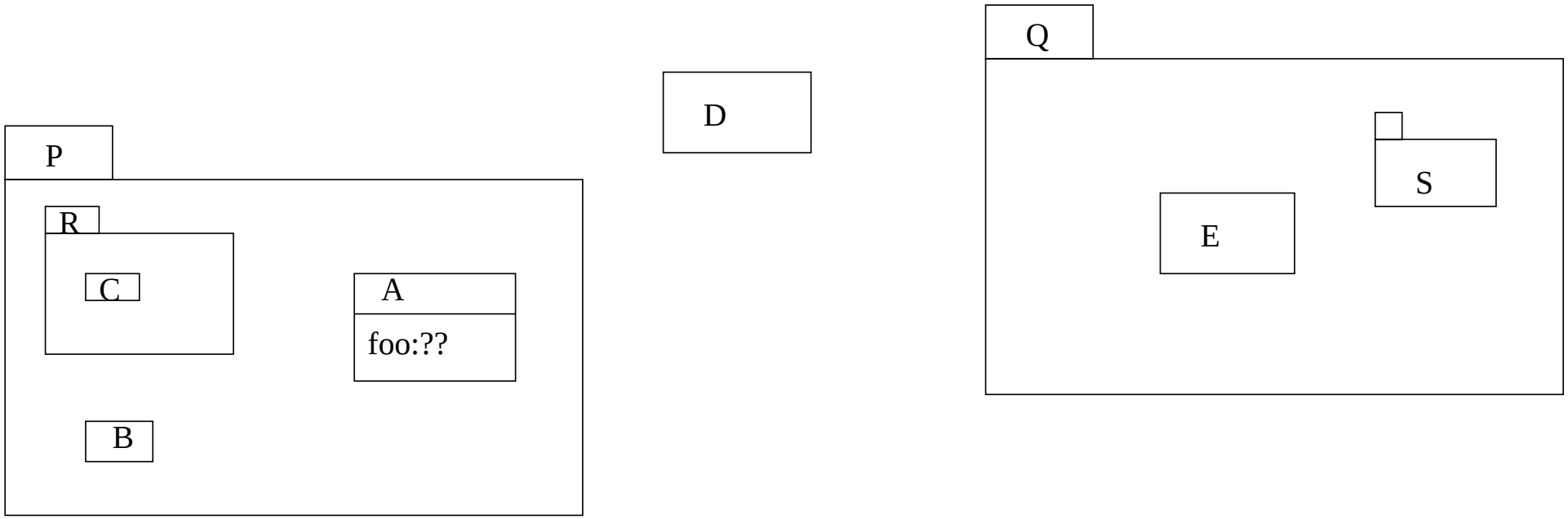


Figure 14.1 Packages and visibility example.

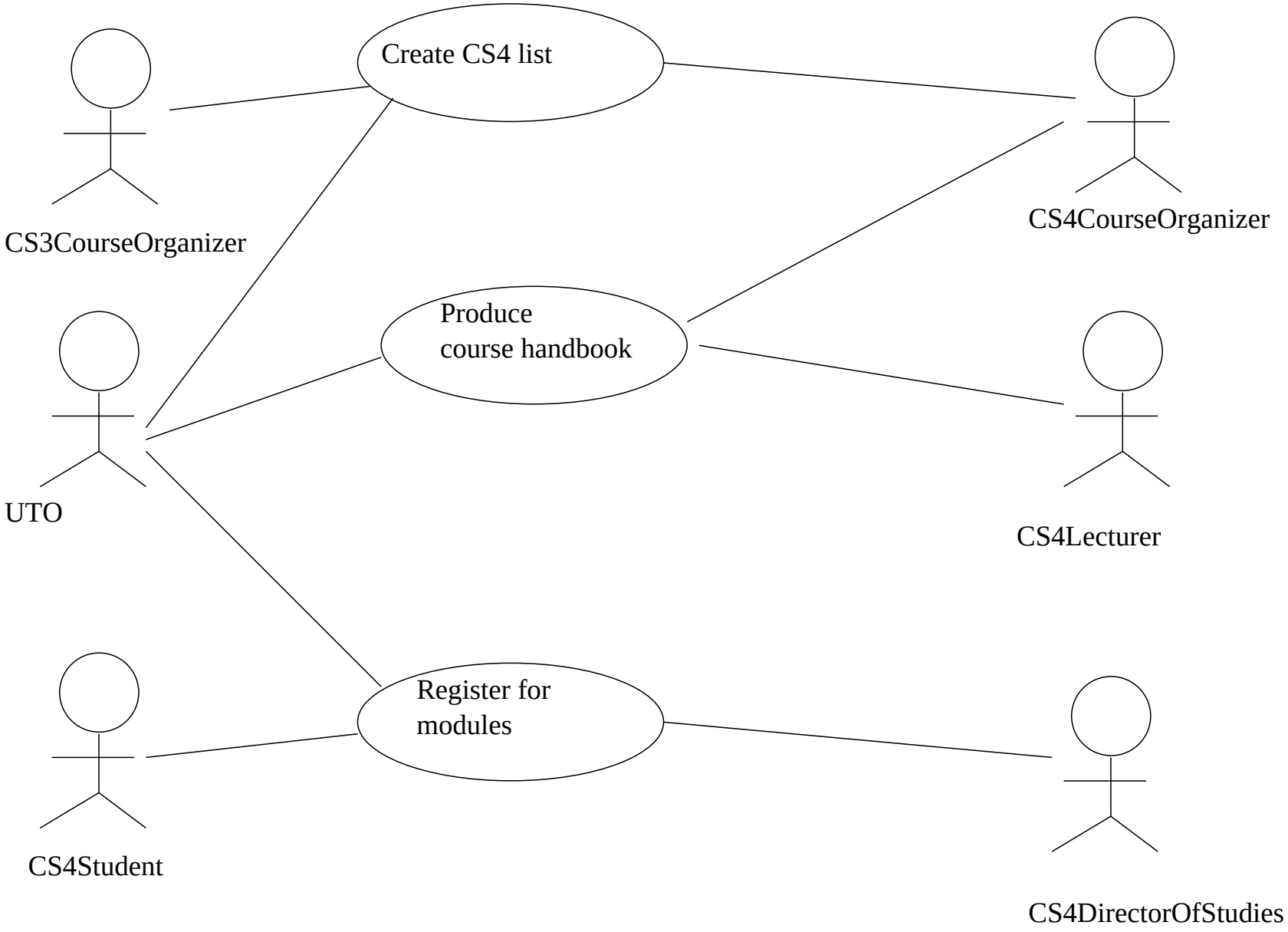


Figure 15.1 Use case model.

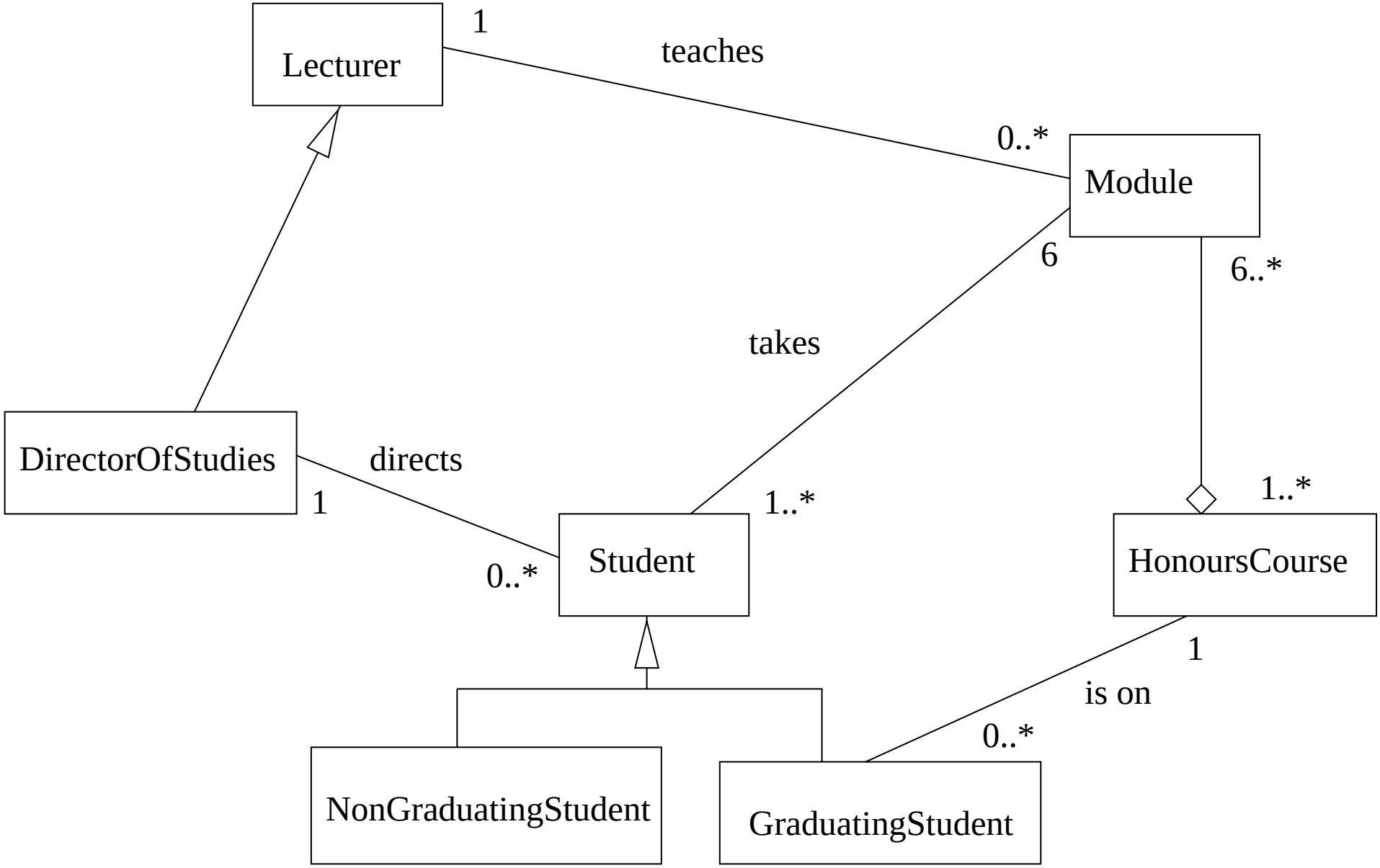


Figure 15.2 Class model.

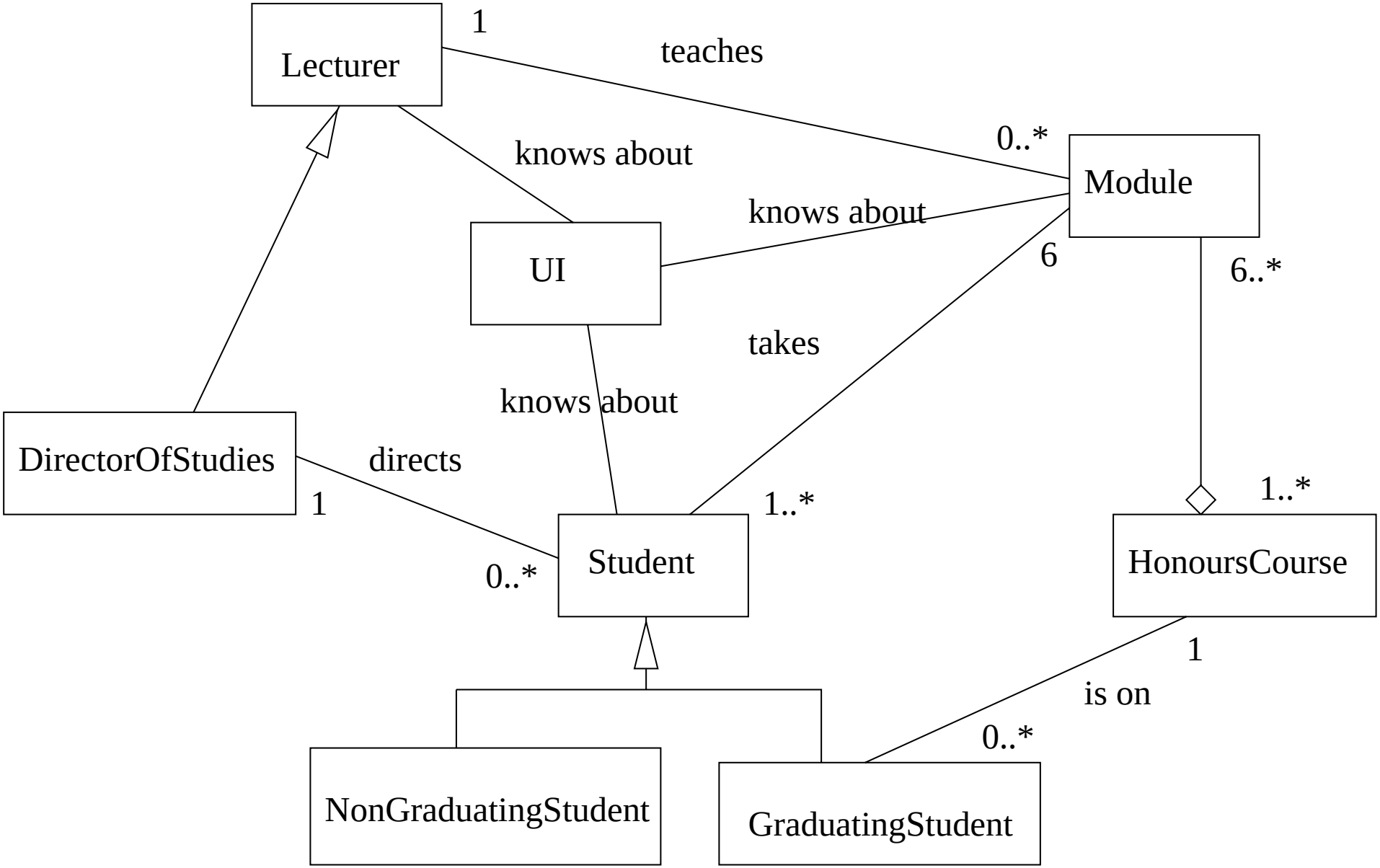


Figure 15.3 Another class model.

Class name: HonoursCourse	
Responsibilities	Collaborators
Keep collection of modules Generate course handbook text	Module

Class name: DirectorOfStudies	
Responsibilities	Collaborators
Provide human DoS's interface to the system	

Class name: Module	
Responsibilities	Collaborators
Keep description of course Keep Lecturer of course	

Figure 15.4 CRC cards needed for `Produce course handbook`.

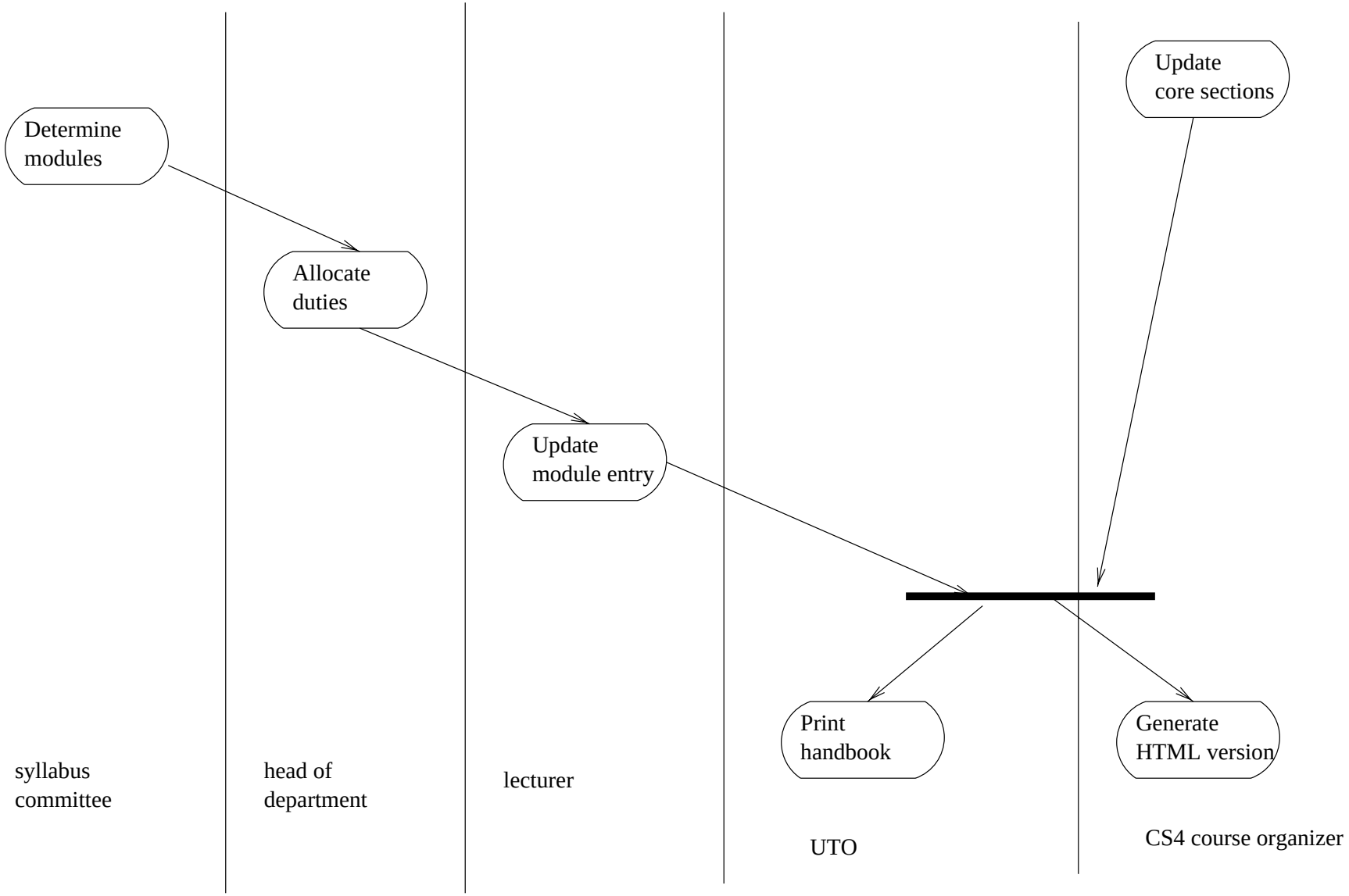


Figure 15.5 An activity diagram for course handbook preparation.

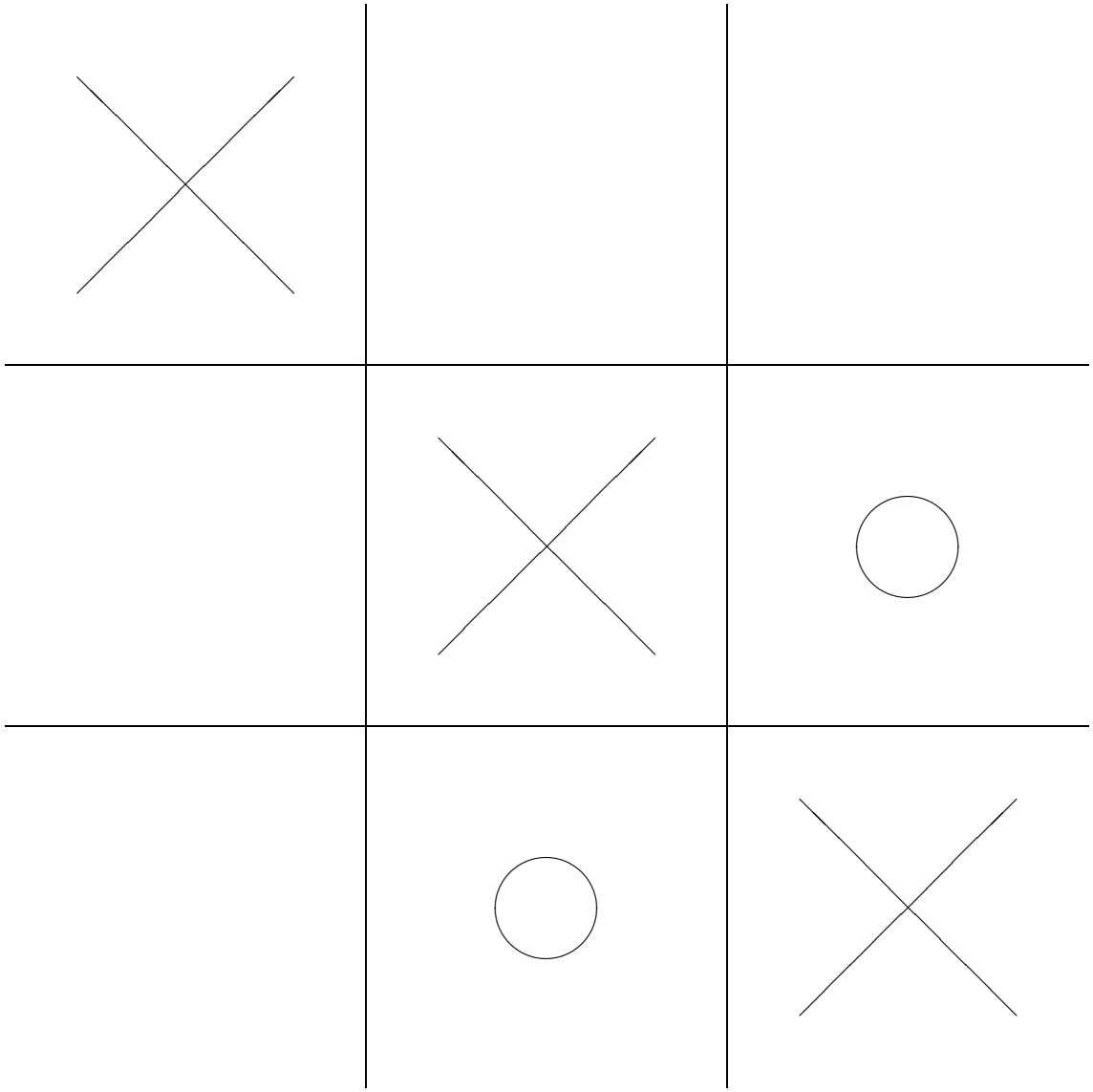


Figure 16.1 Noughts and Crosses (Tic-Tac-Toe).

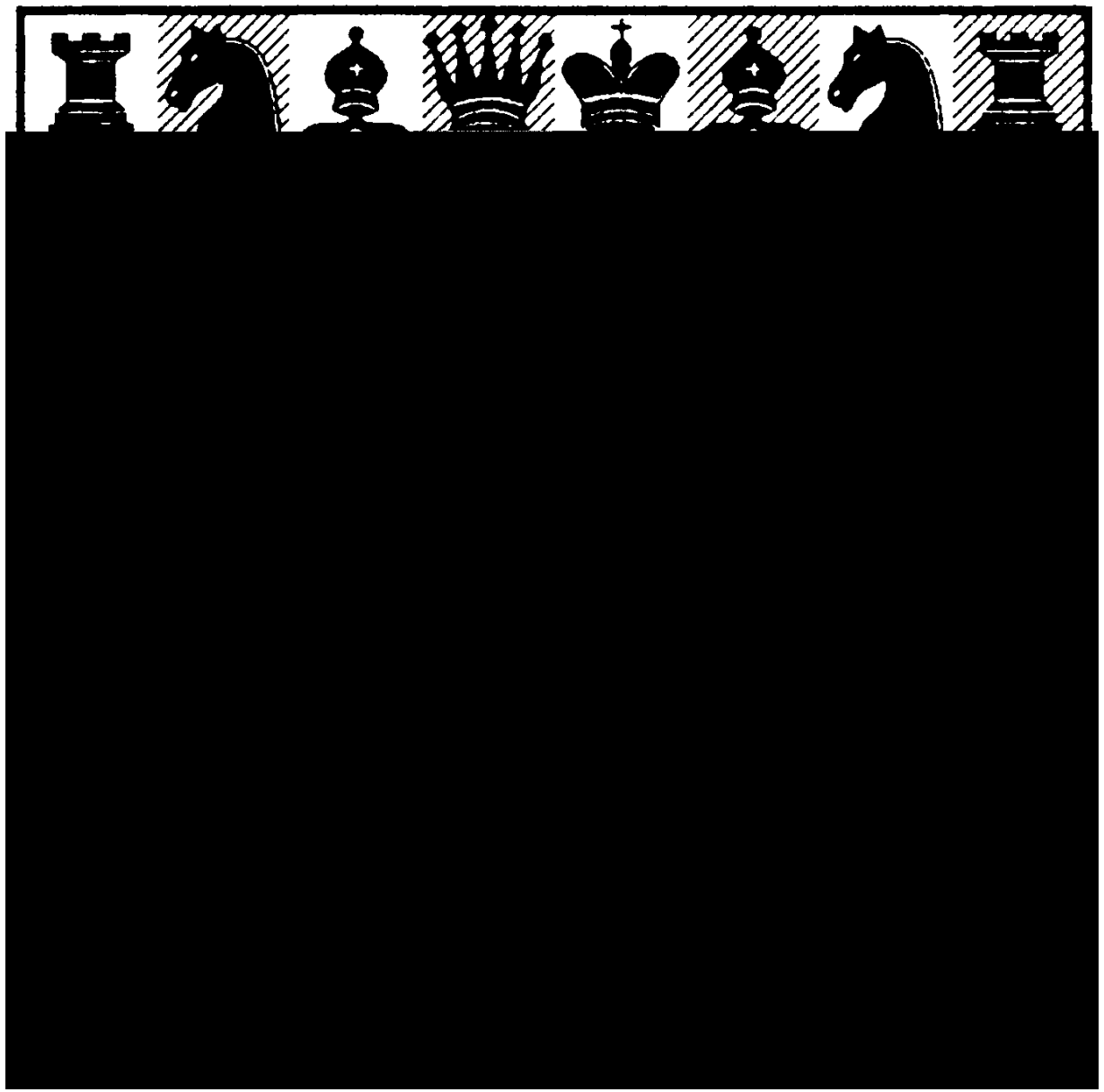


Figure 16.2 Chess.

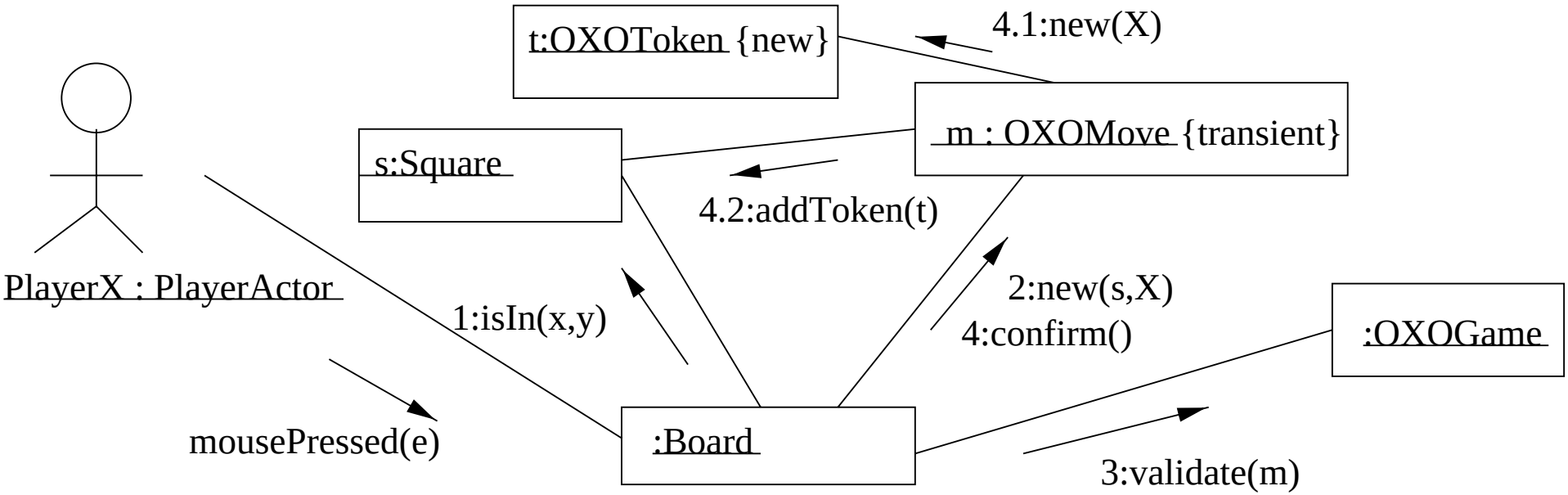


Figure 16.3 Collaboration diagram for an X move in Noughts and Crosses.

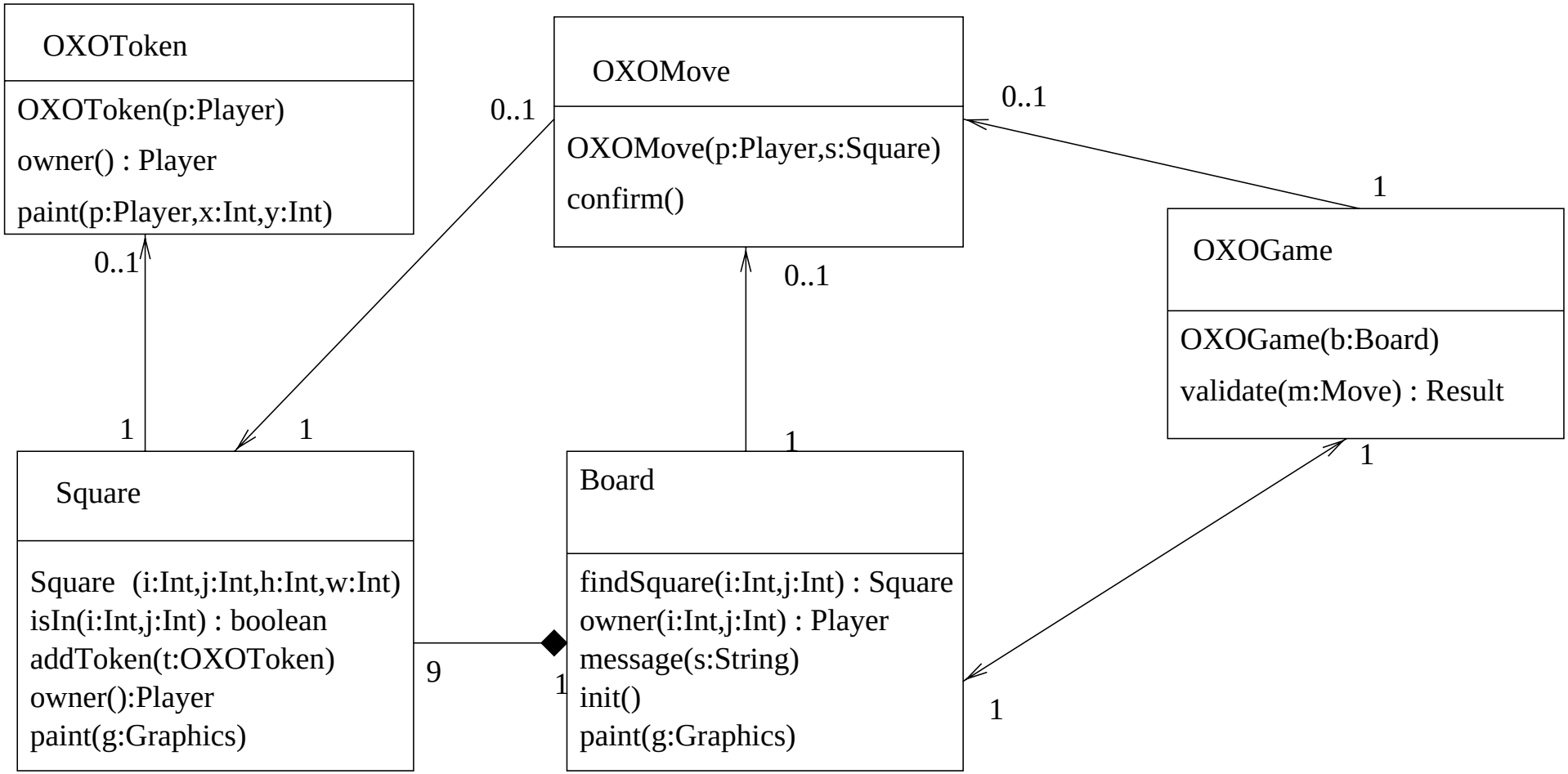


Figure 16.4 Class diagram for Noughts and Crosses.

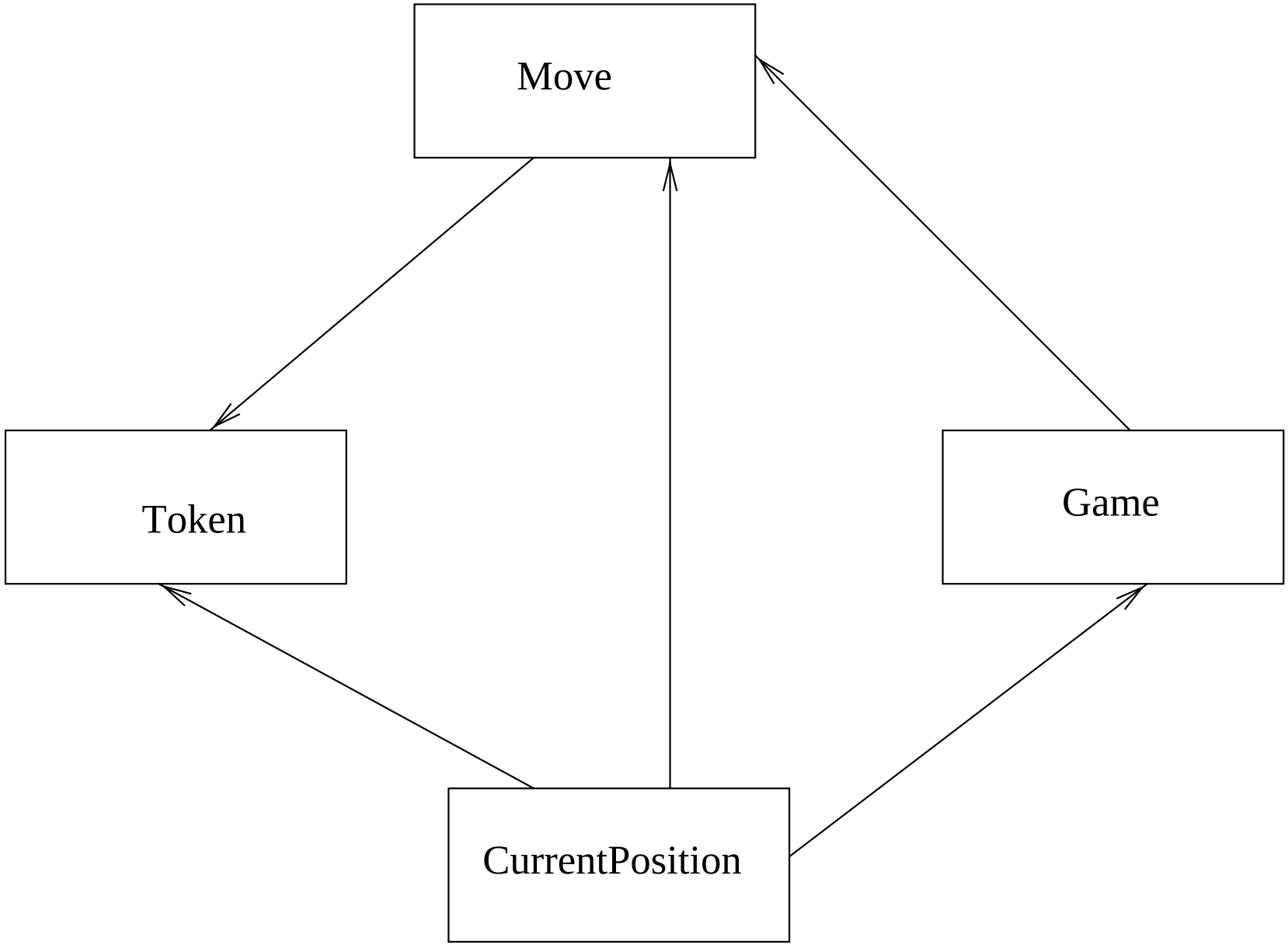


Figure 16.5 Class diagram for games framework.

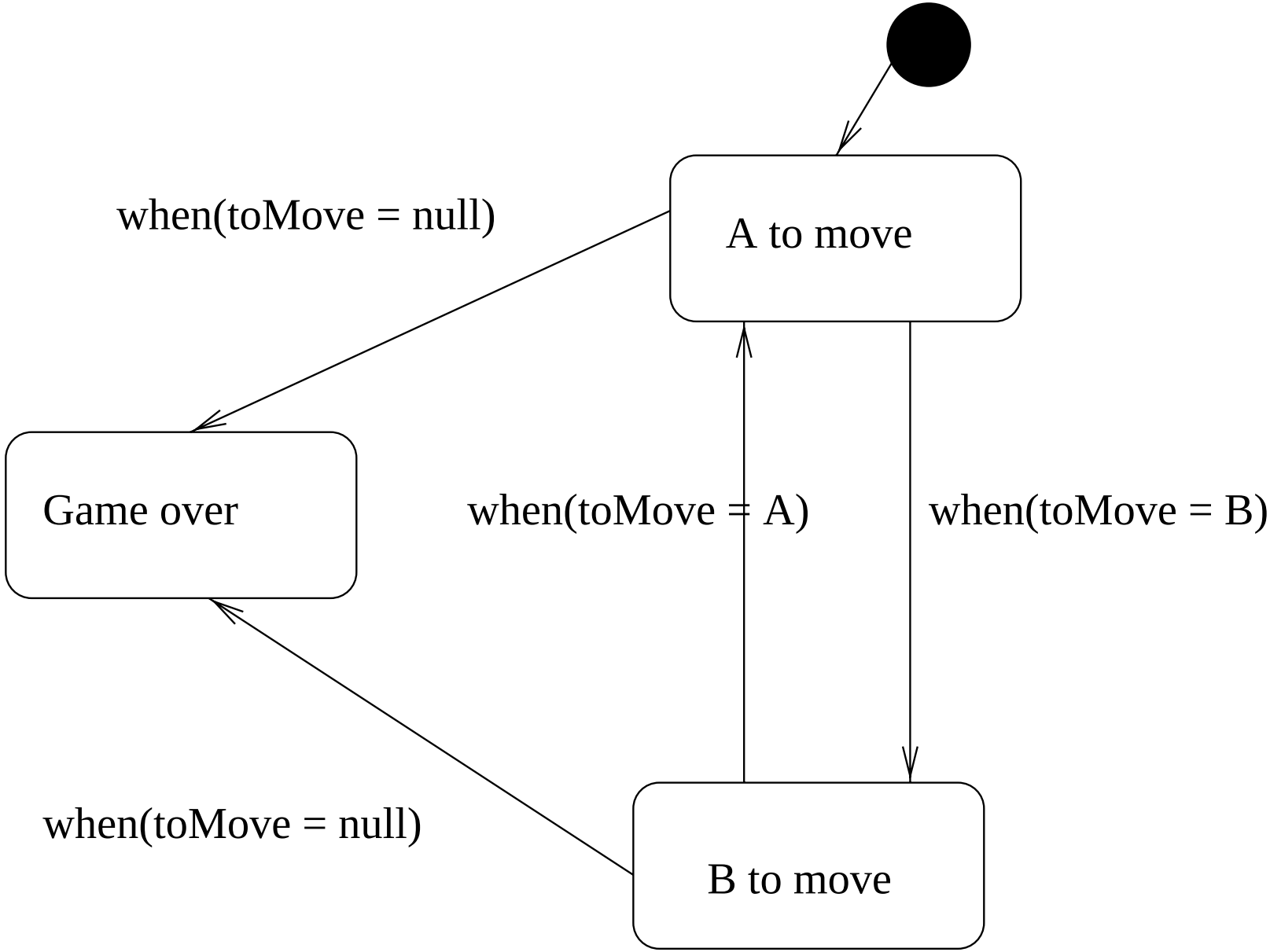


Figure 16.6 State diagram for `CurrentPosition`.

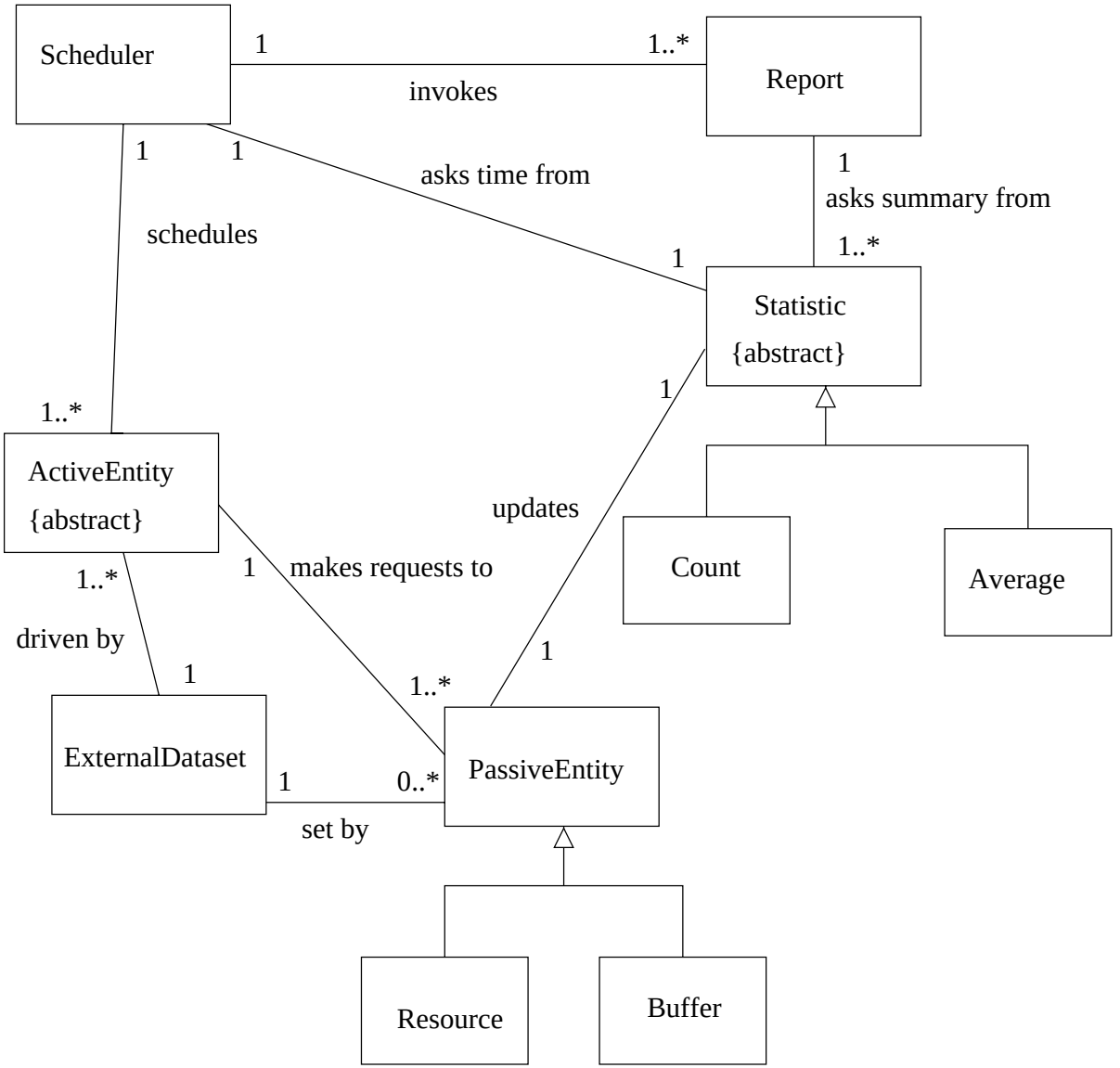
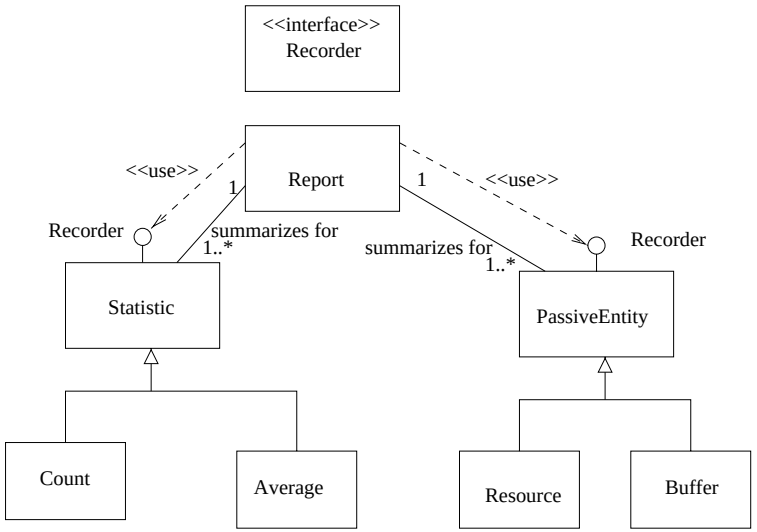
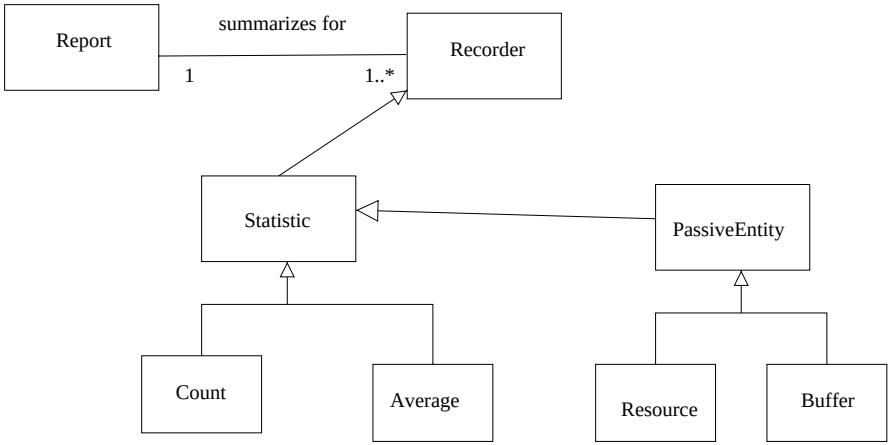


Figure 17.1 Class diagram of discrete event simulation system.



(a) Using an interface to show common behavior



(b) One way to use generalization to show common behavior

Figure 17.2 Some alternatives for classes used in reporting behavior.

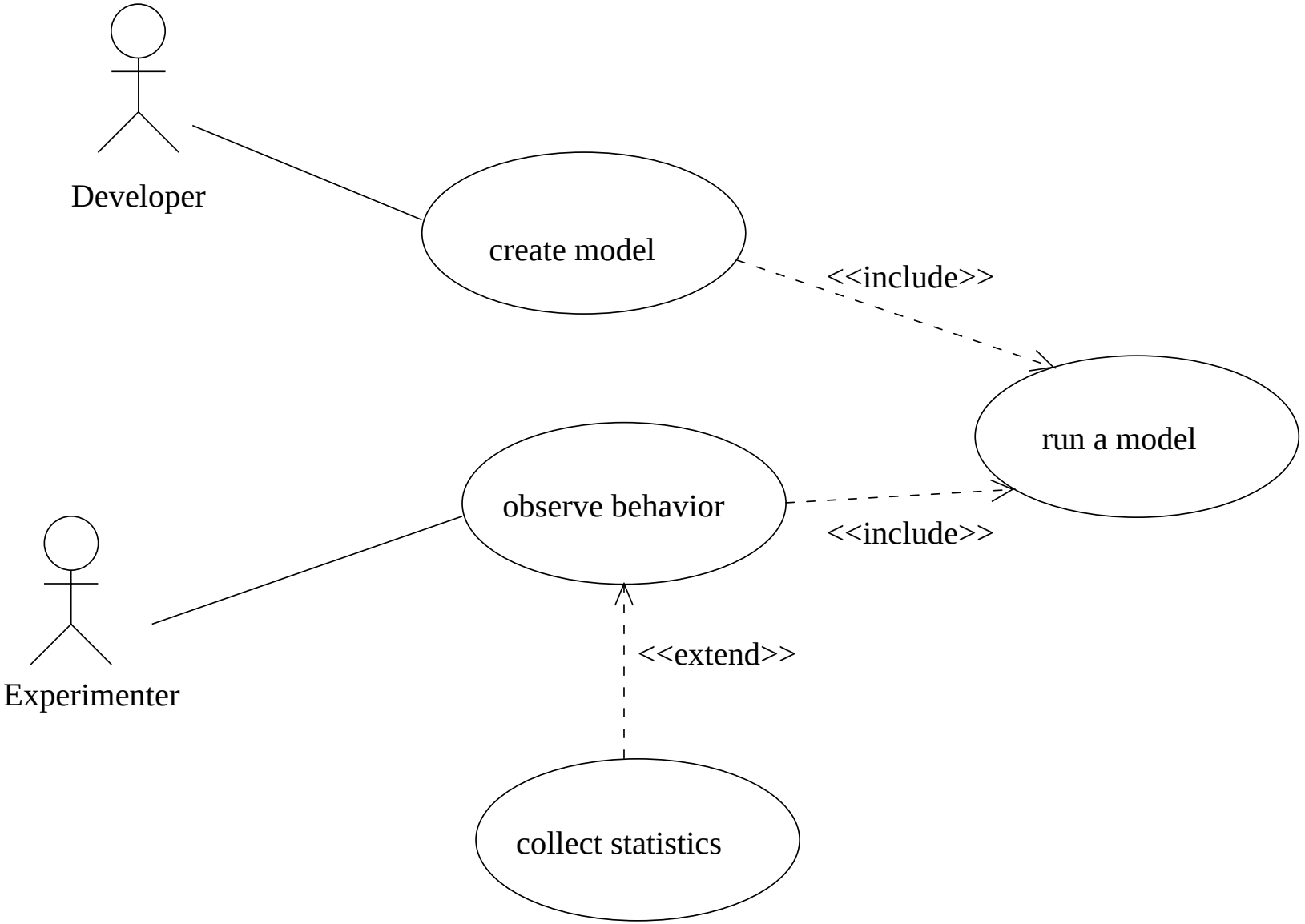


Figure 17.3 Use case diagram of discrete event simulation system.

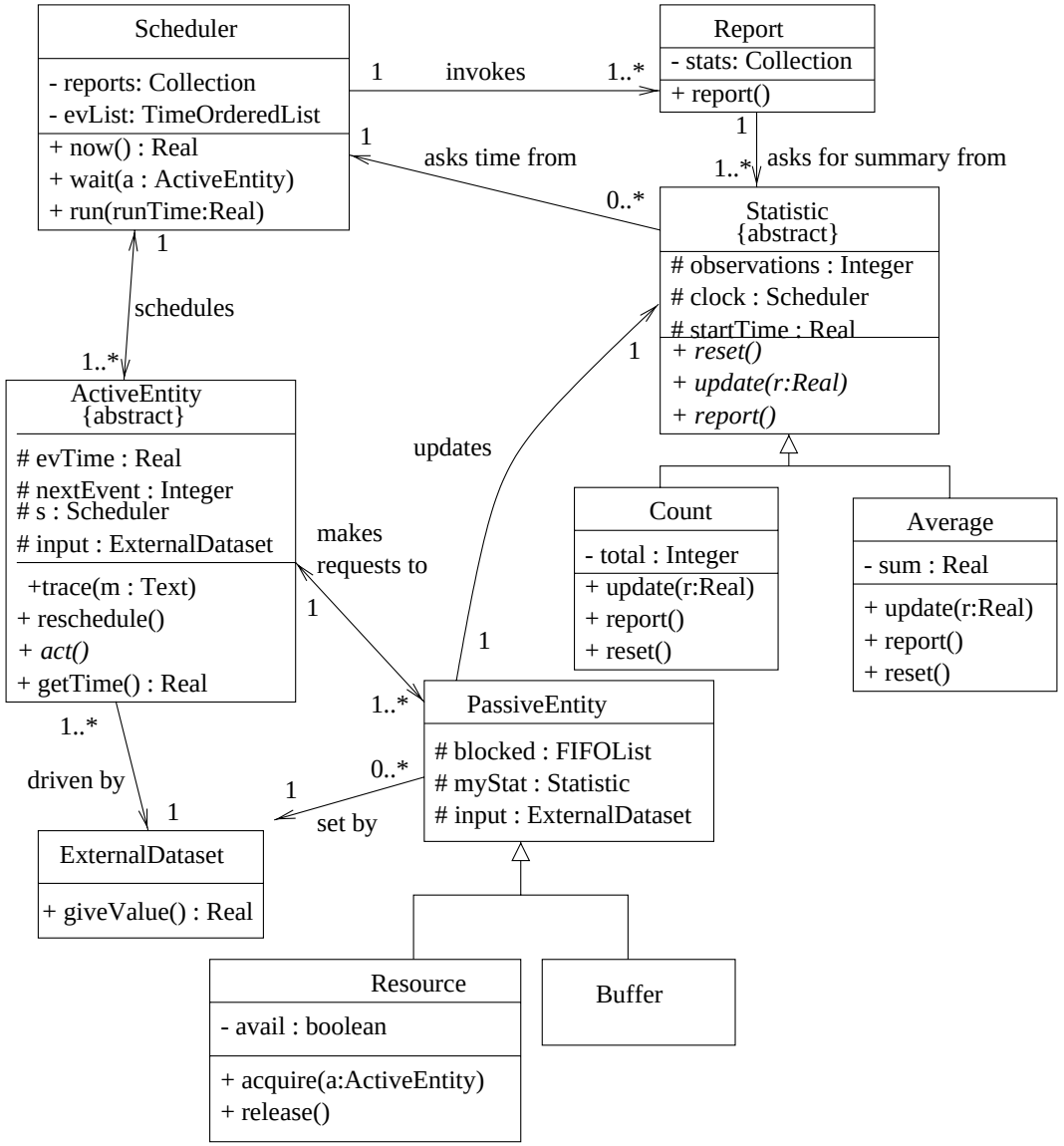


Figure 17.4 Detailed class diagram for a simulation experiment.

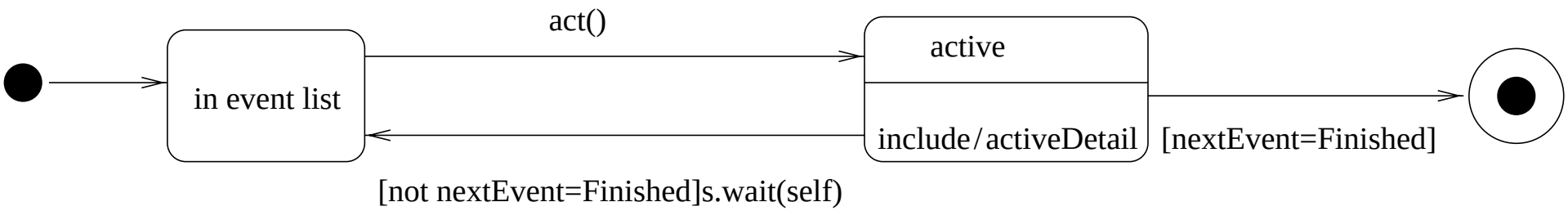


Figure 17.5 State diagram of the generic `ActiveEntity`.

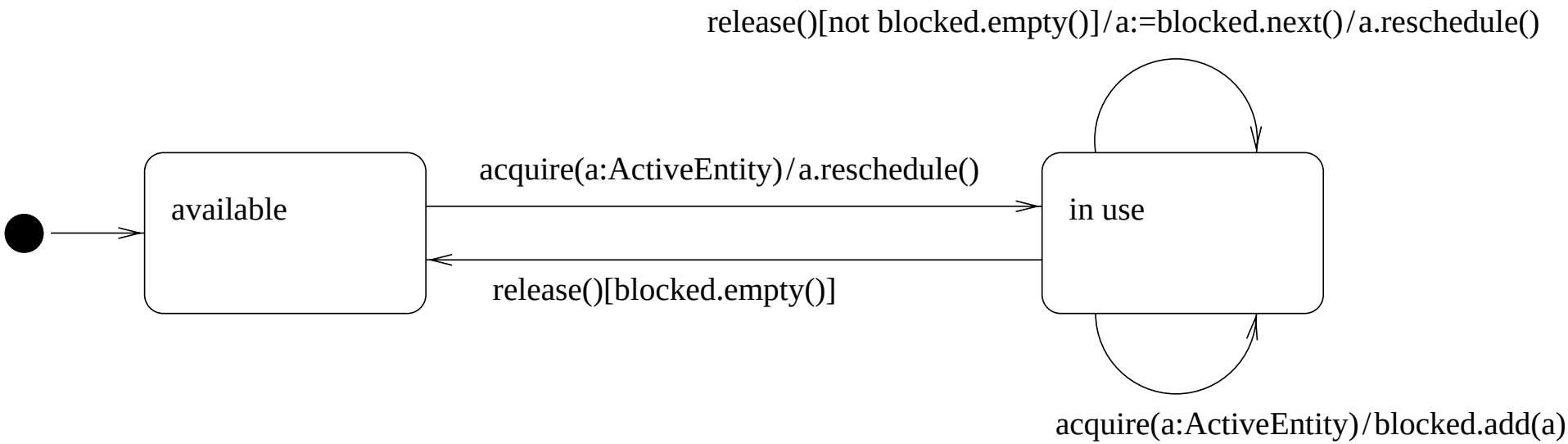


Figure 17.6 State diagram of `Resource`.

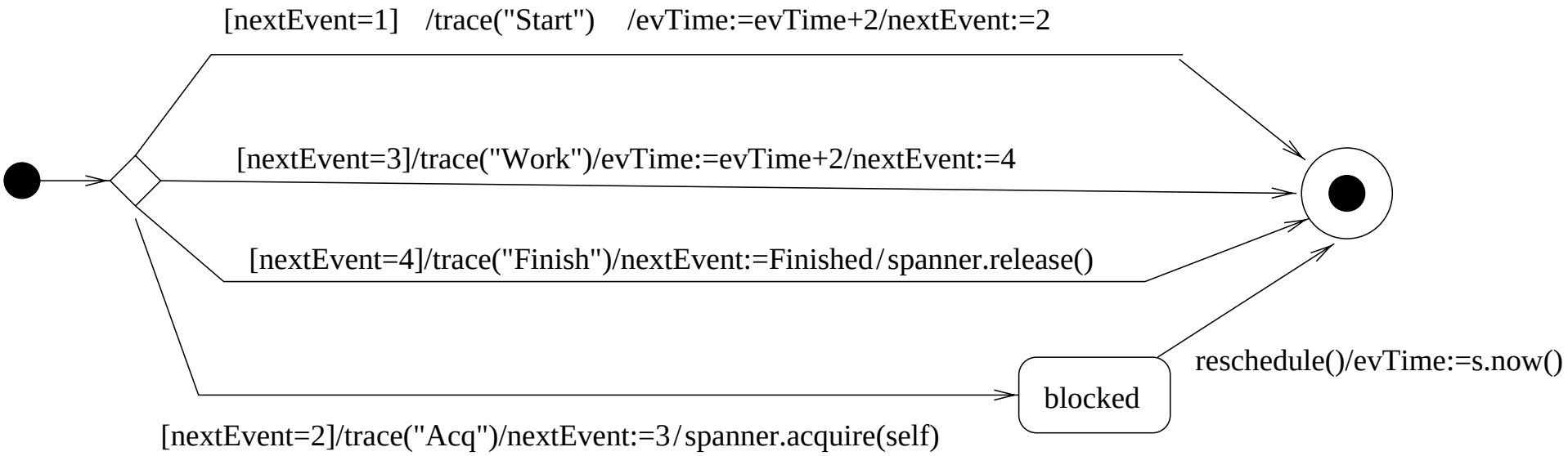


Figure 17.7 `activeDetail` state diagram of class `Worker`.

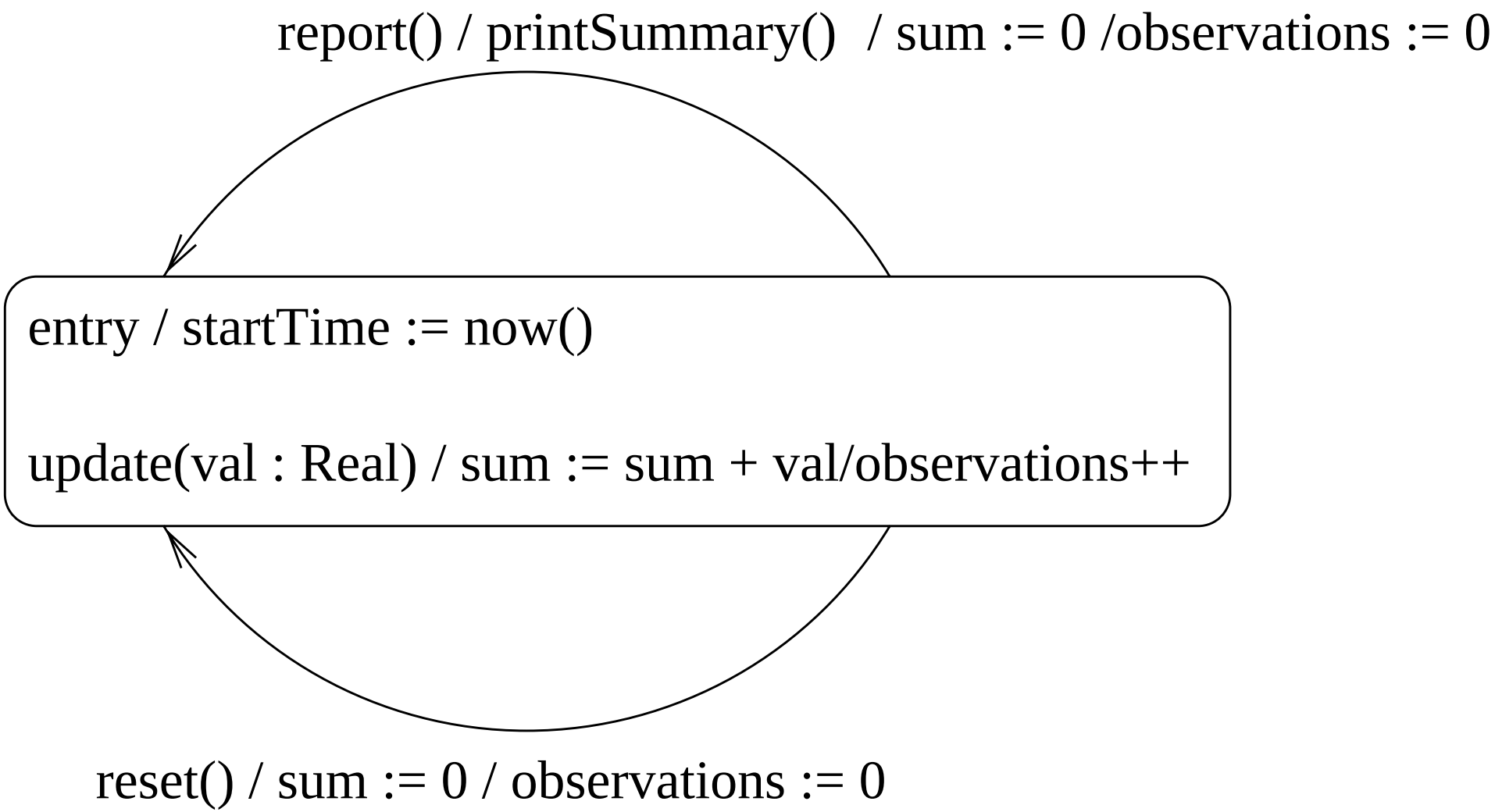


Figure 17.8 State diagram of `Average`.

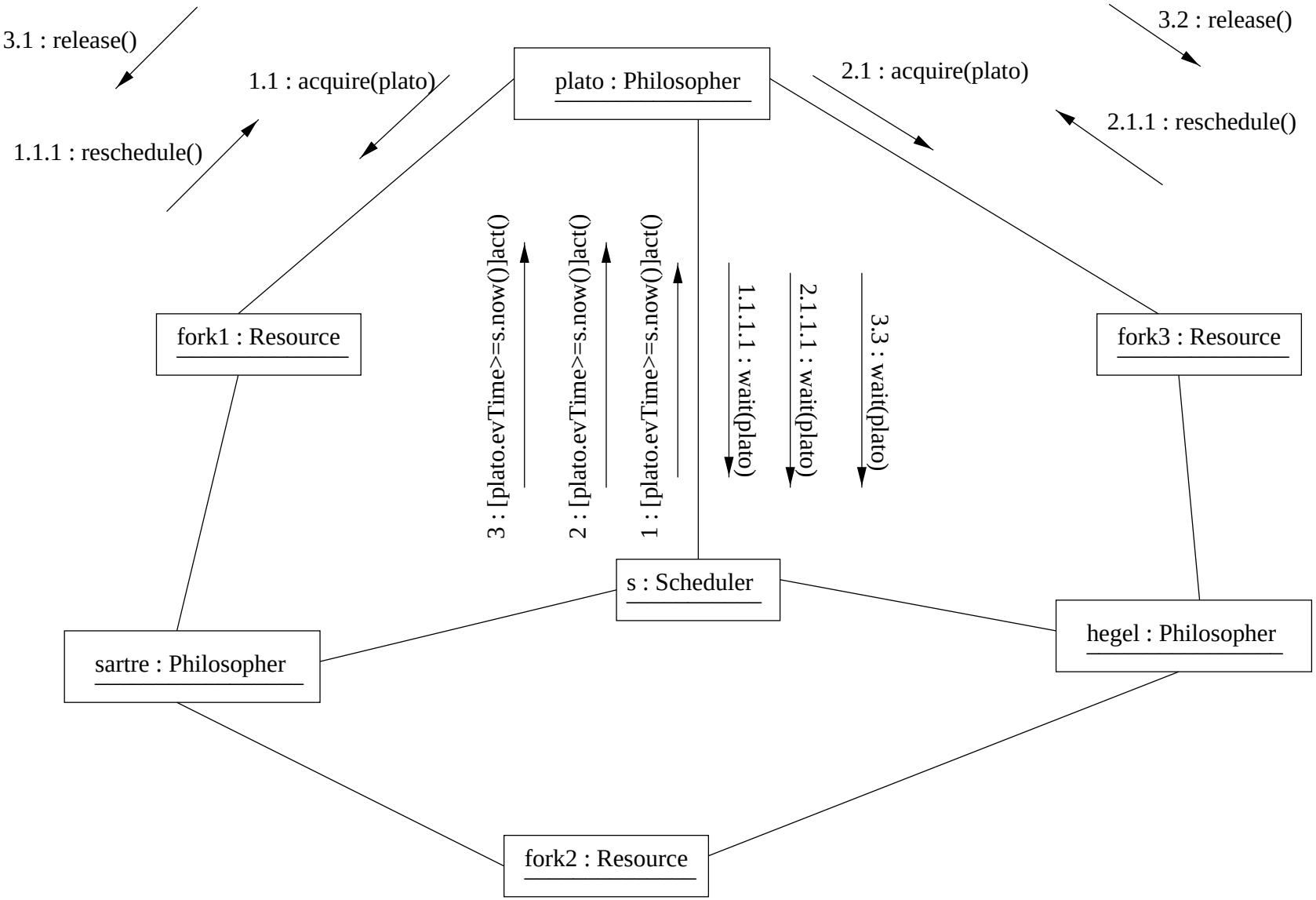


Figure 17.9 Collaboration diagram of the dining philosophers.

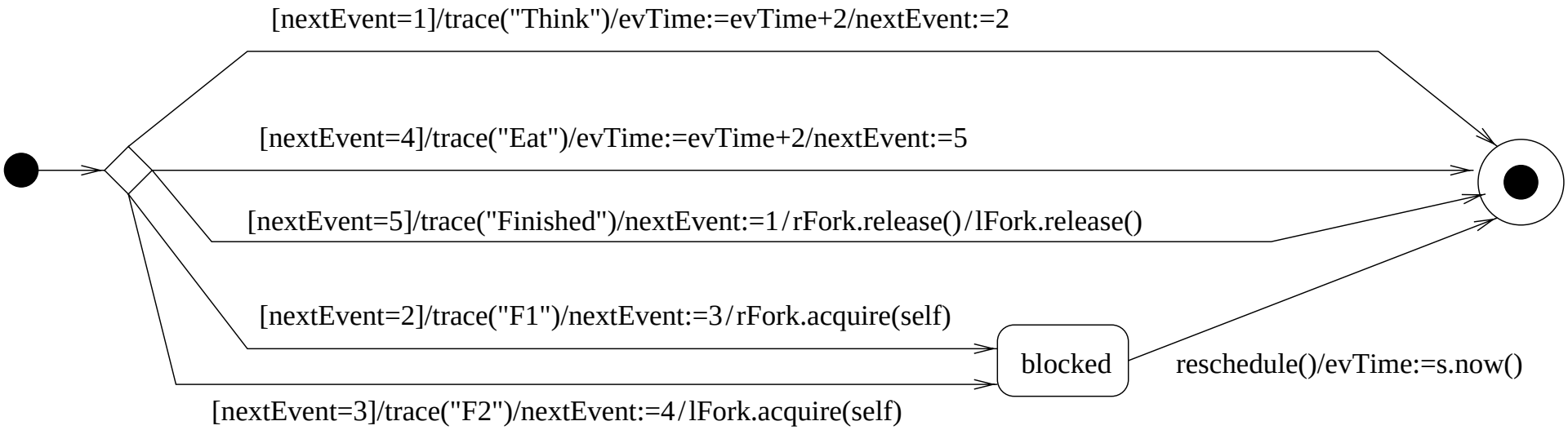


Figure 17.10 `activeDetail` state diagram of class `Philosopher`.

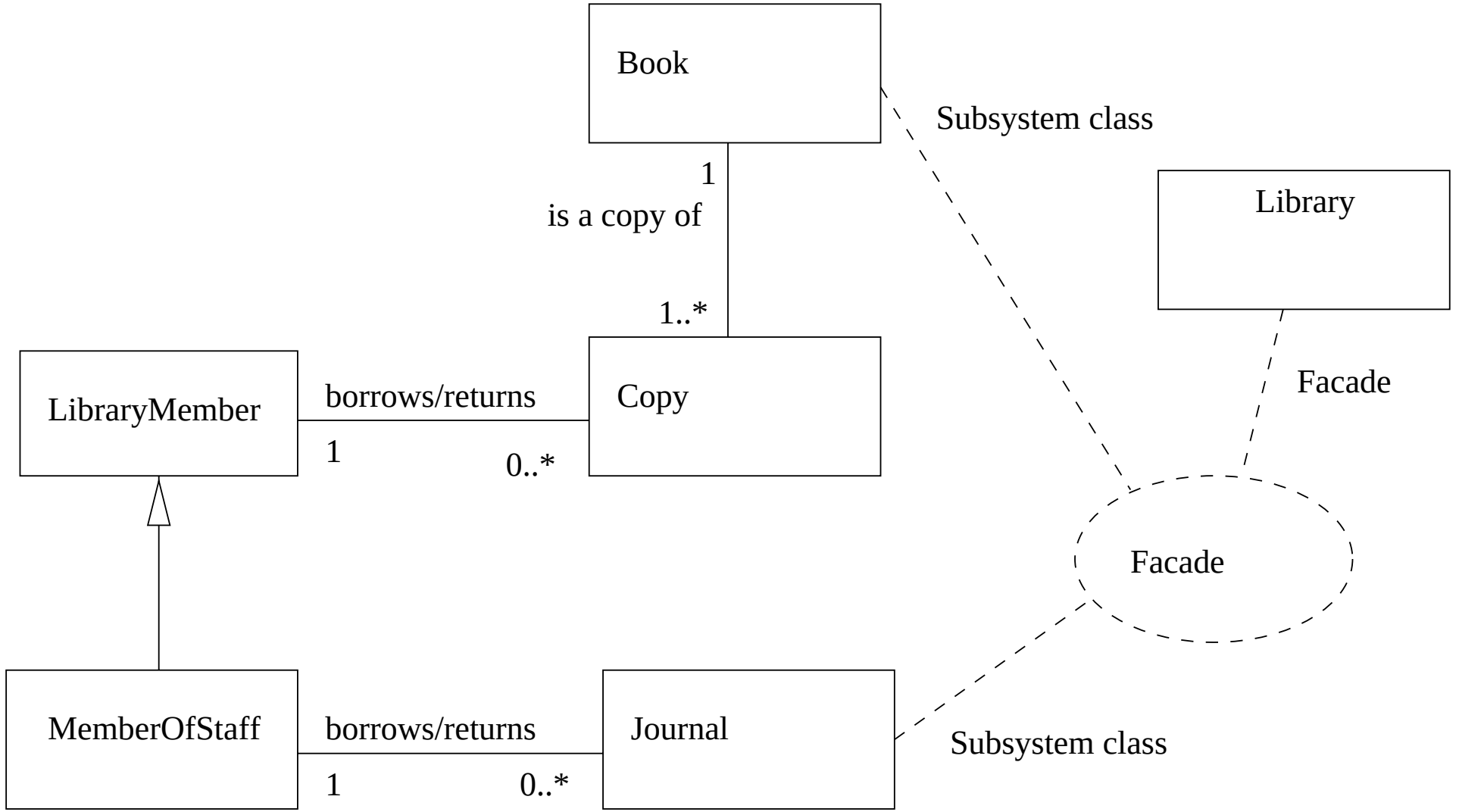
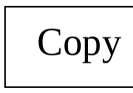
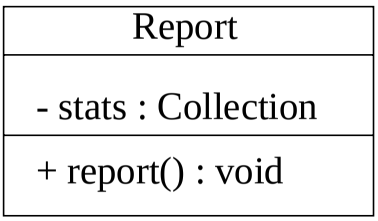


Figure 18.1 The Façade pattern applied to the library.

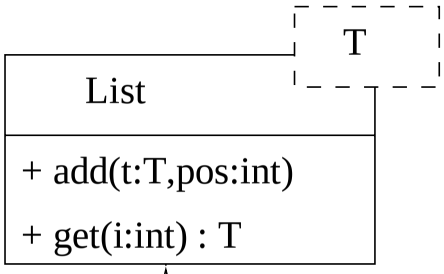
Classes - Chapters 5 and 6

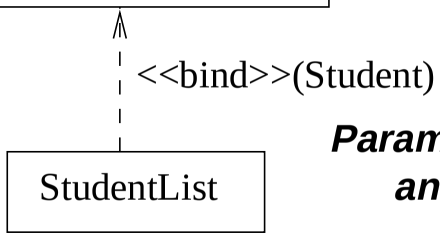


Simple class

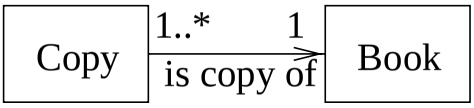


Class with attribute and operation

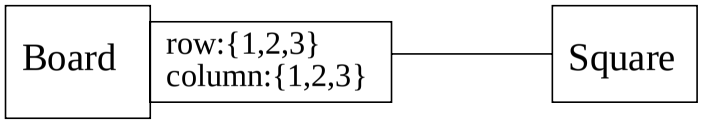




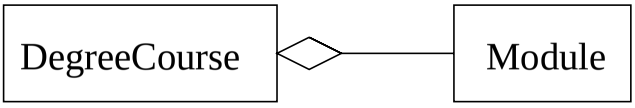
Parameterized class and its uses



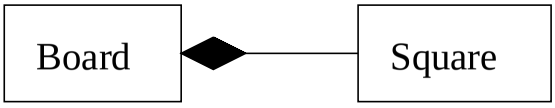
Association with multiplicities and navigability



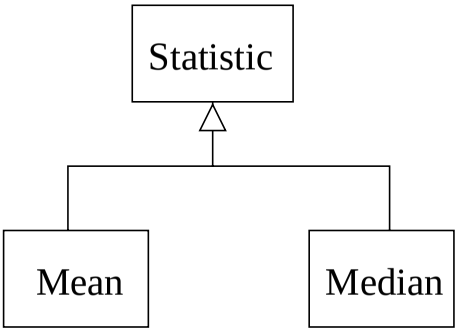
Qualified association



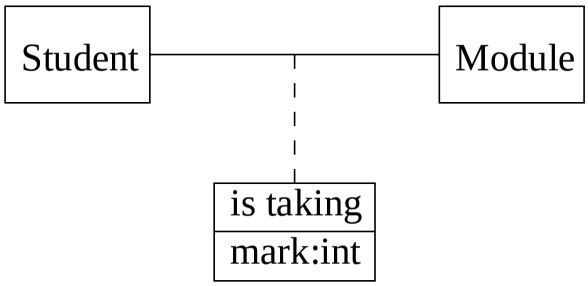
Aggregation



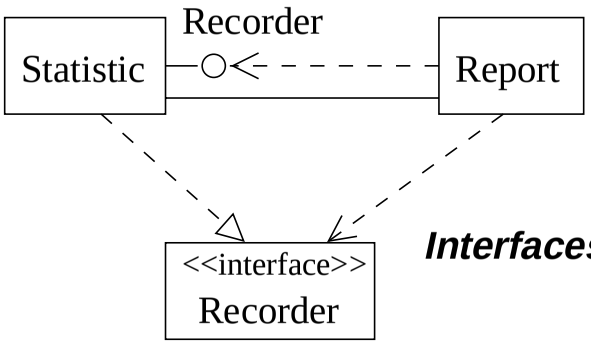
Composition



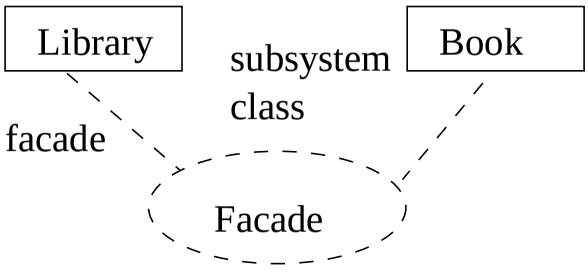
Generalization



Association class

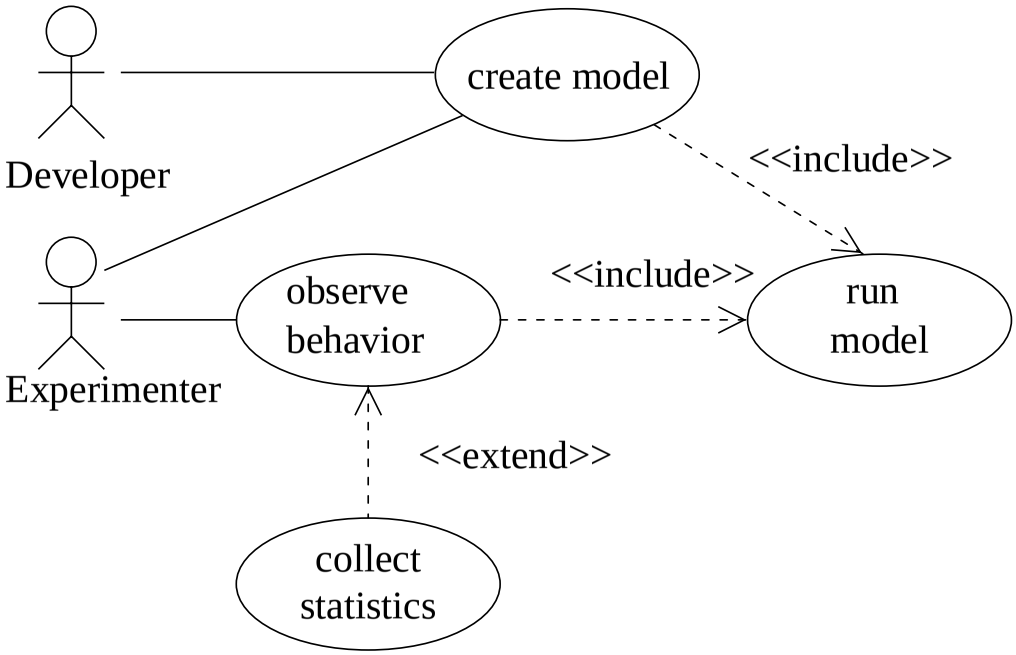


Interfaces

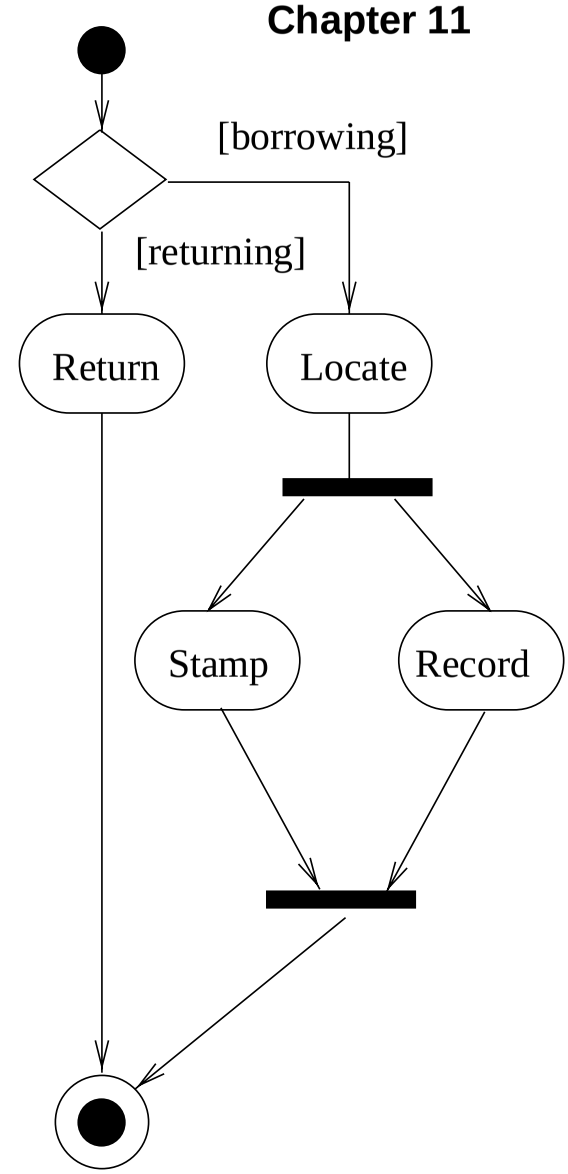


Pattern - Chapter 18

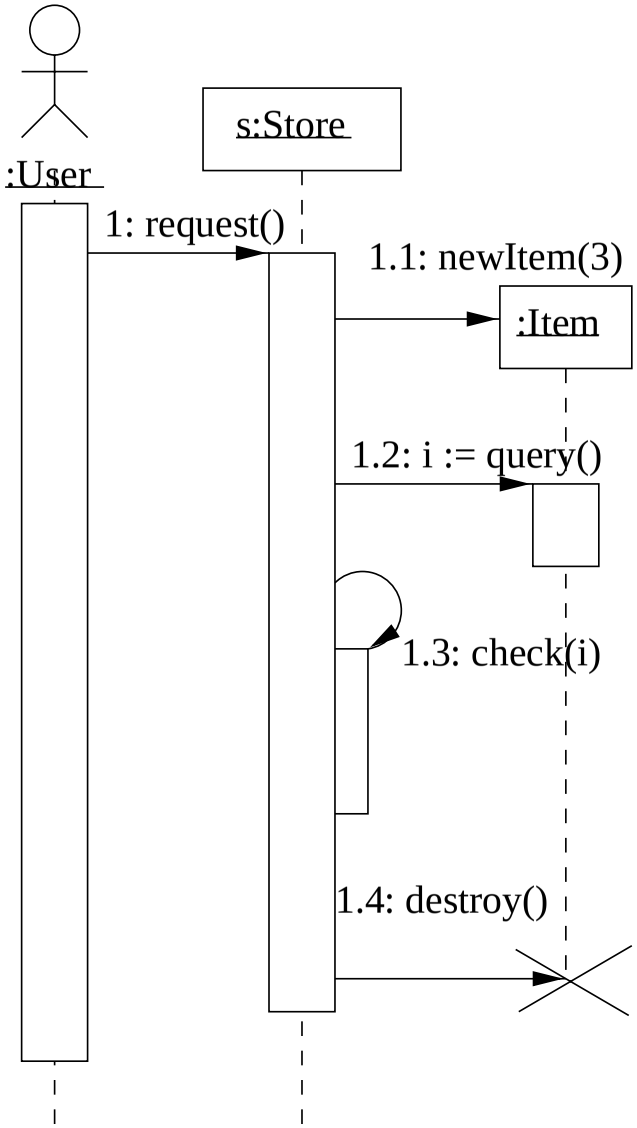
Use cases - Chapters 7 and 8



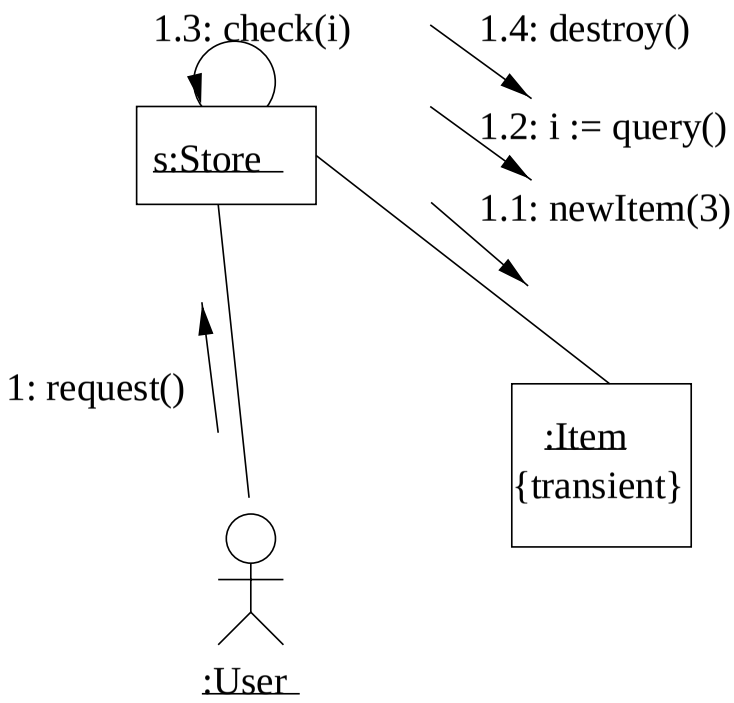
Activity diagrams - Chapter 11

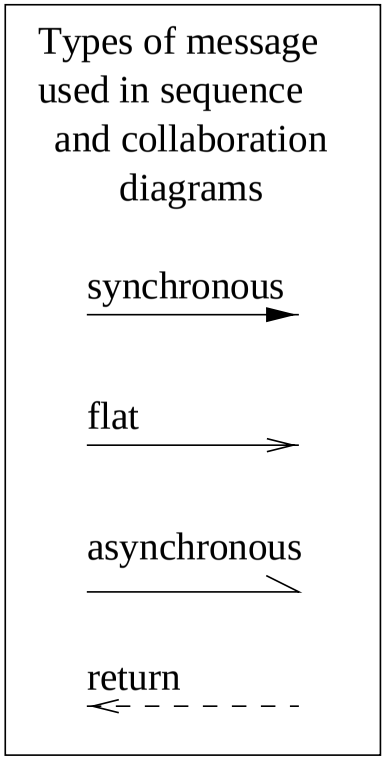


Sequence diagrams - Chapters 9 and 10

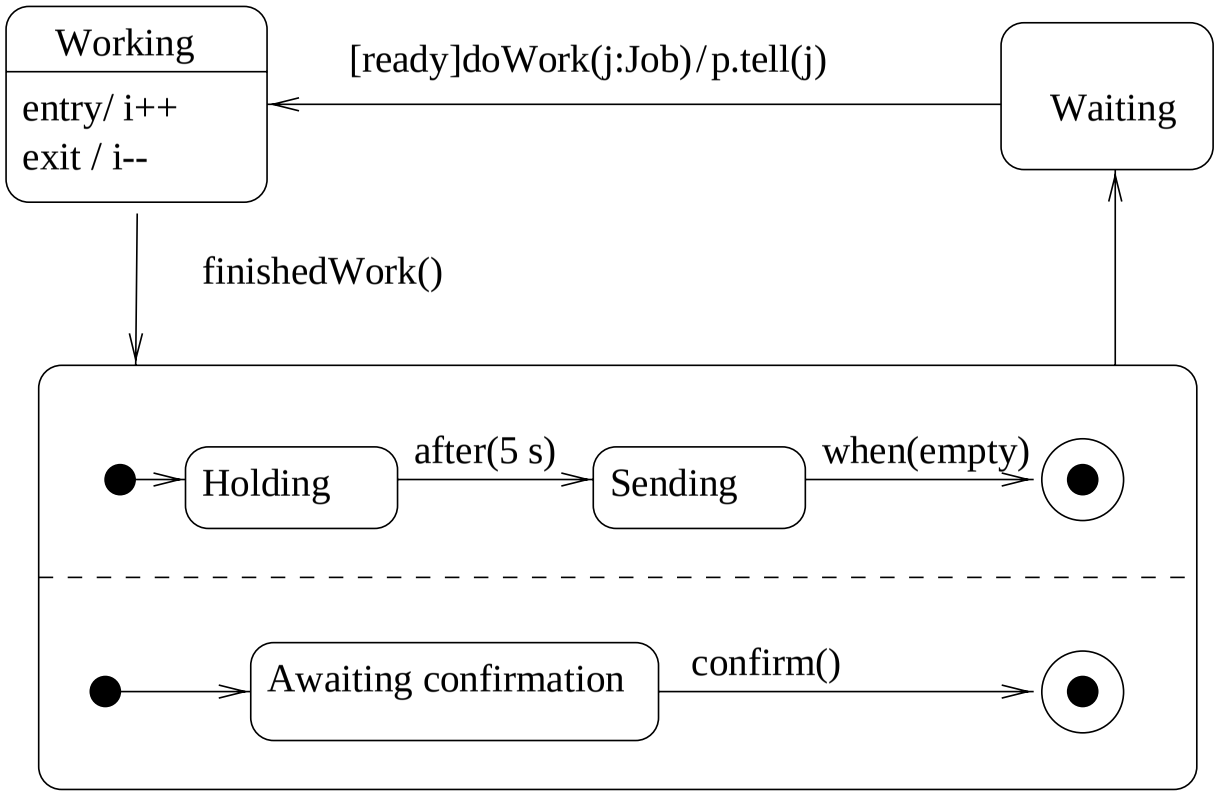


Collaboration diagrams - Chapters 7 and 8



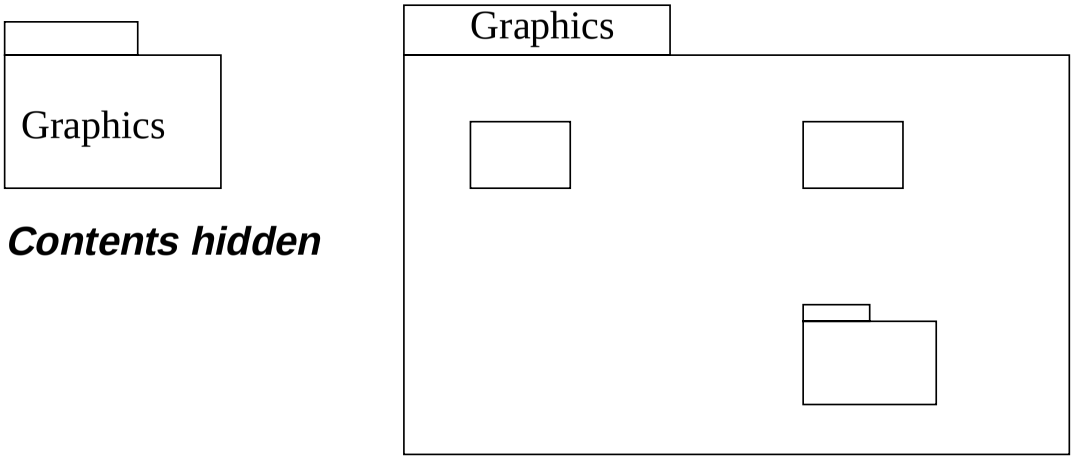


State diagrams - Chapters 11 and 12



Nested concurrent state diagram

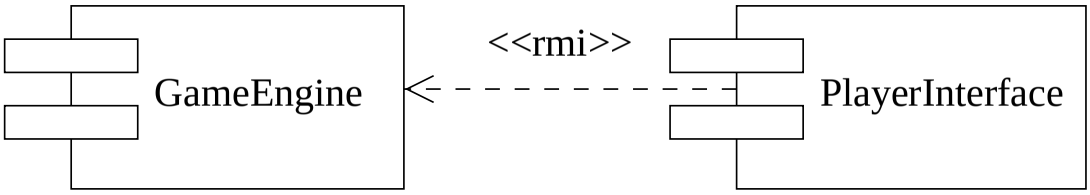
Packages - Chapter 14



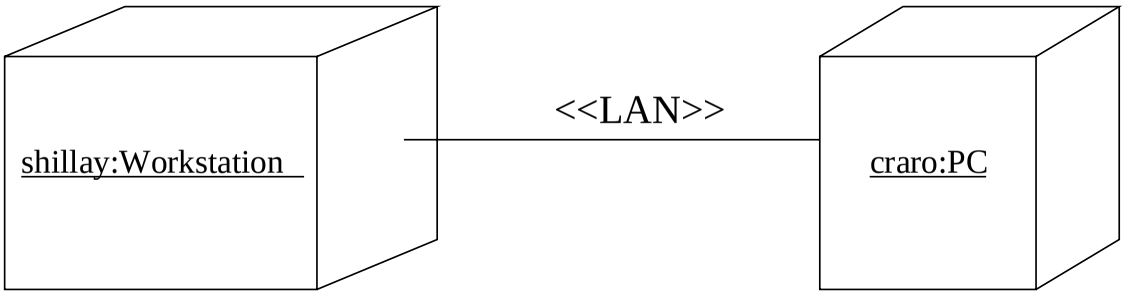
Contents hidden

Contents shown

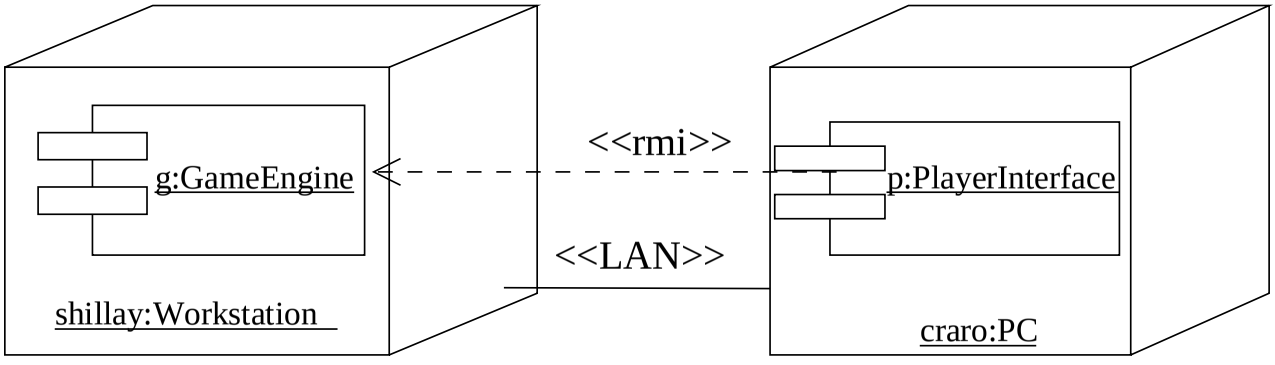
Implementation diagrams - Chapter 13



Dependency between two components



Physical nodes without software



Software deployed on nodes