COMP 433 Software Engineering

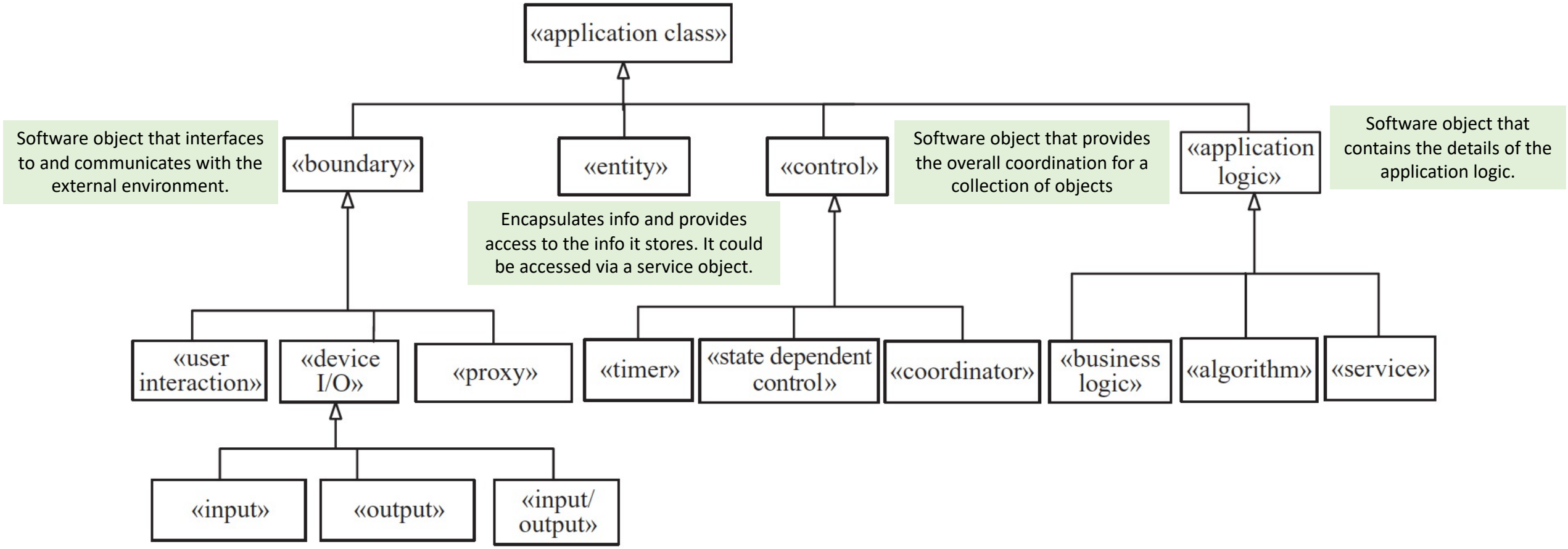
Module 5: *System Modeling with UML*

Ahmed Tamrawi

Modeling Application Classes

- In Class Diagram module, we discussed the static modeling of **entity classes**, which benefit most from static modeling in the analysis phase because they are *information-intensive*. **Entity classes**, however, are only one kind of software class within the system.
- Before dynamic modeling (e.g., sequence diagrams) can be undertaken, it is necessary to determine what **software classes** are needed to realize each **use case**.
- Identification of software classes can be greatly assisted by applying class structuring criteria, which categorizes software classes *by the roles they play in the application*.

Modeling Application Classes



Software object that interfaces to and communicates with the external environment.

Encapsulates info and provides access to the info it stores. It could be accessed via a service object.

Software object that provides the overall coordination for a collection of objects

Software object that contains the details of the application logic.

Modeling Application Classes

- Most applications will have objects from each of the four categories. However, different types of applications will have a greater number of classes in one or another category.
- **Information-intensive systems** will have several **entity** classes, which is why static modeling is so vital for these systems.
- **Real-time systems** are likely to have several device **I/O boundary** classes to interface to the various sensors and actuators. They are also likely to have complex state-dependent control classes because these systems are highly state-dependent.

Boundary Classes: *User Interaction Class*

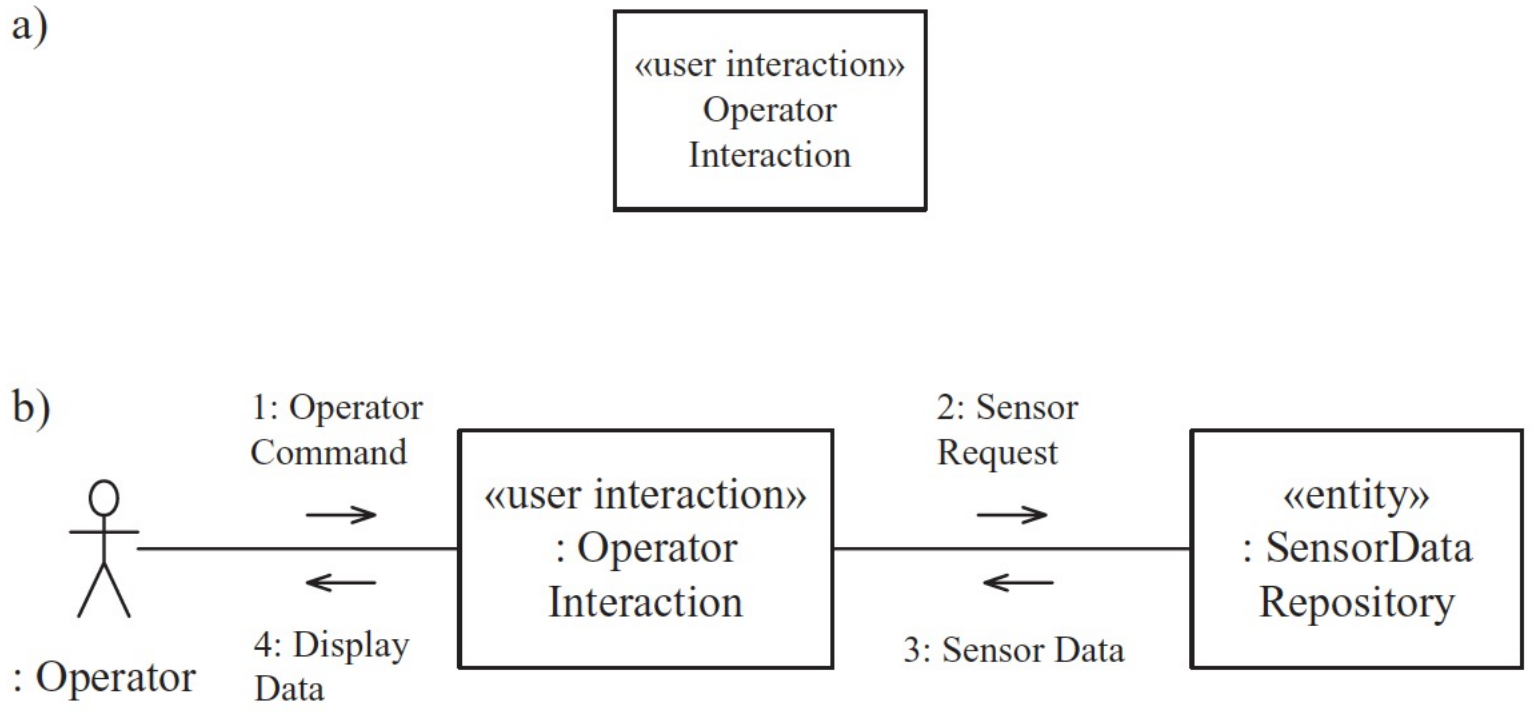
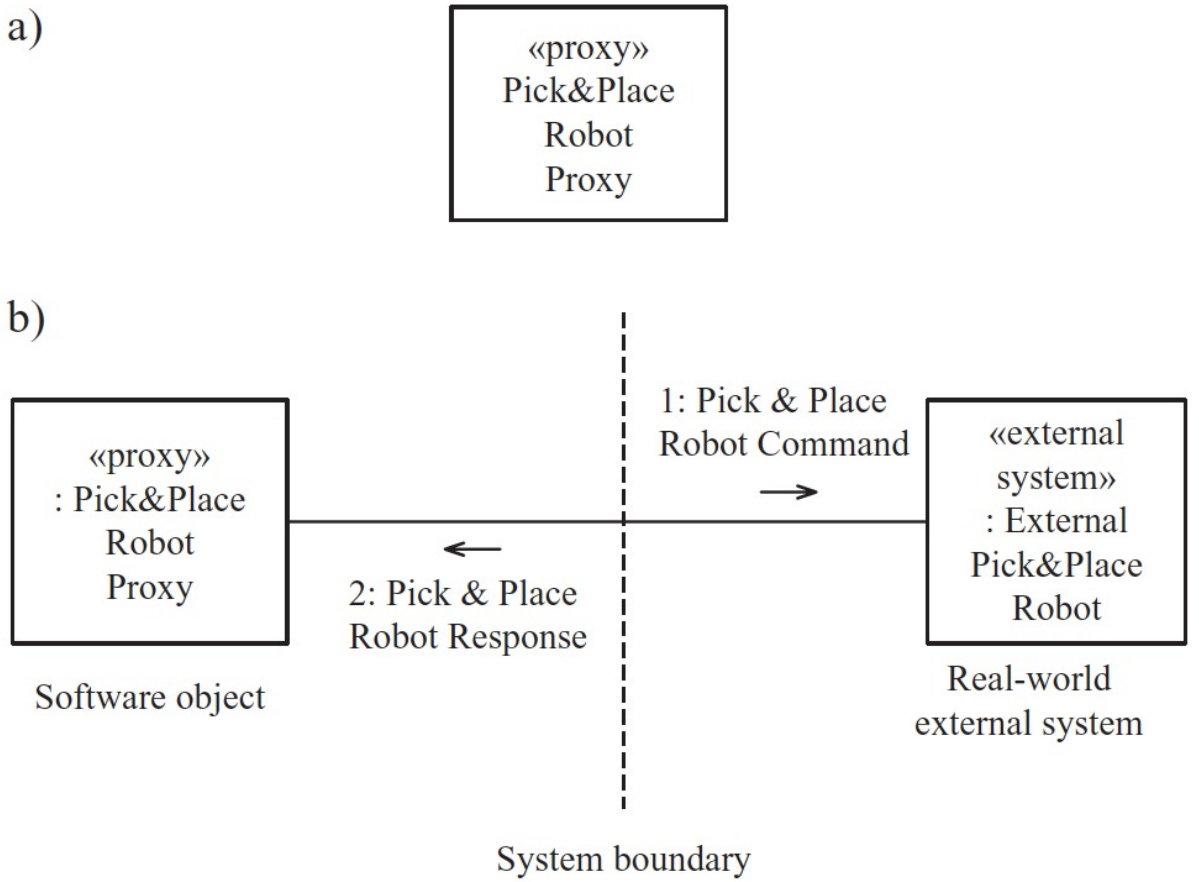


Figure 8.2. Example of user interaction class and object

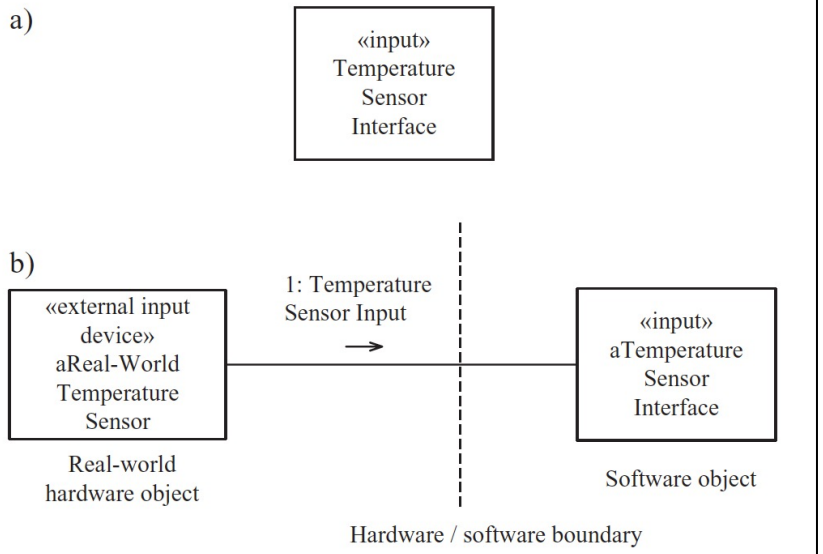
Boundary Classes: *Proxy Class*



(Note: the dashed line for the system boundary is for illustrative purpose only and does not conform to the UML notation.)

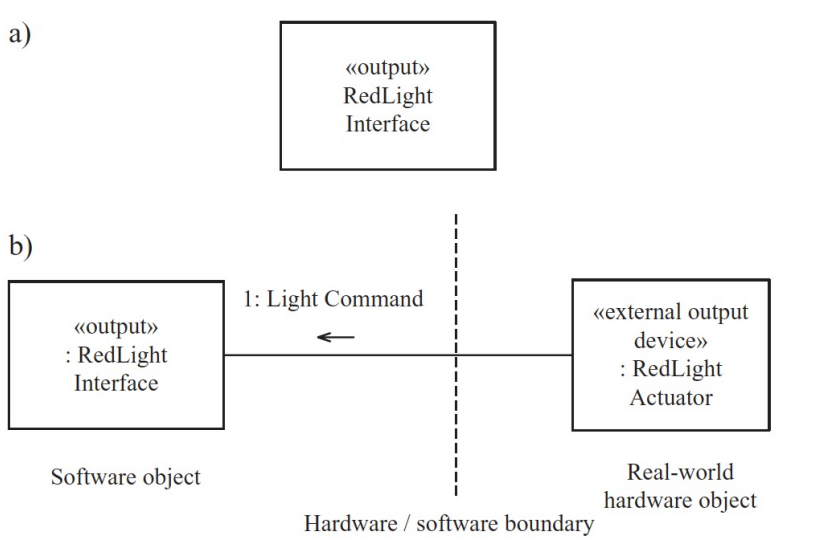
Figure 8.3. Example of proxy class and object

Boundary Classes: *I/O Class*



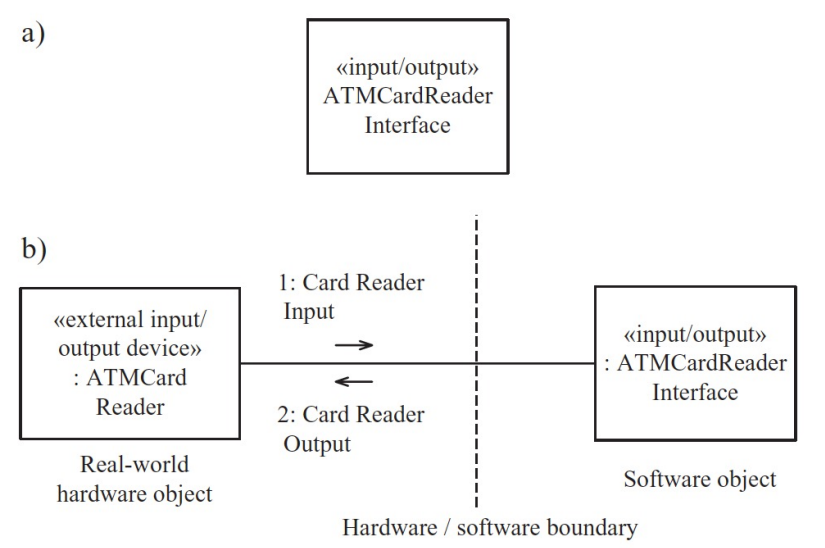
(Note: the dashed line for the hardware/software boundary is for illustrative purpose only and does not conform to the UML notation.)

Figure 8.4. Example of input class and object



(Note: the dashed line for the hardware/software boundary is for illustrative purpose only and does not conform to the UML notation.)

Figure 8.5. Example of output class and object



(Note: the dashed line for the hardware/software boundary is for illustrative purpose only and does not conform to the UML notation.)

Figure 8.6. Example of I/O class and object

Depicting External Classes and Boundary Classes

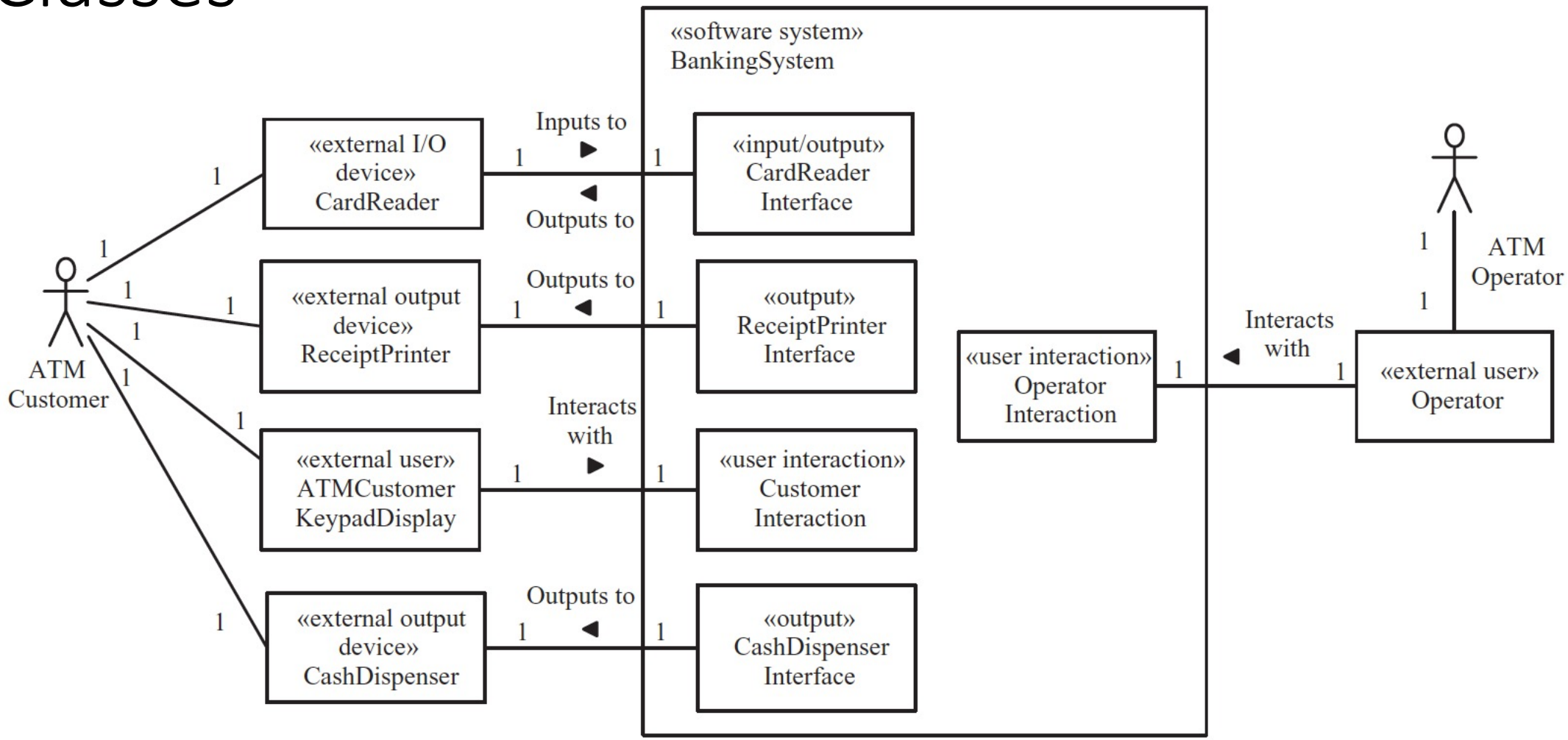


Figure 8.7. Banking System external classes and boundary classes

Control Classes: *Coordinator Class*

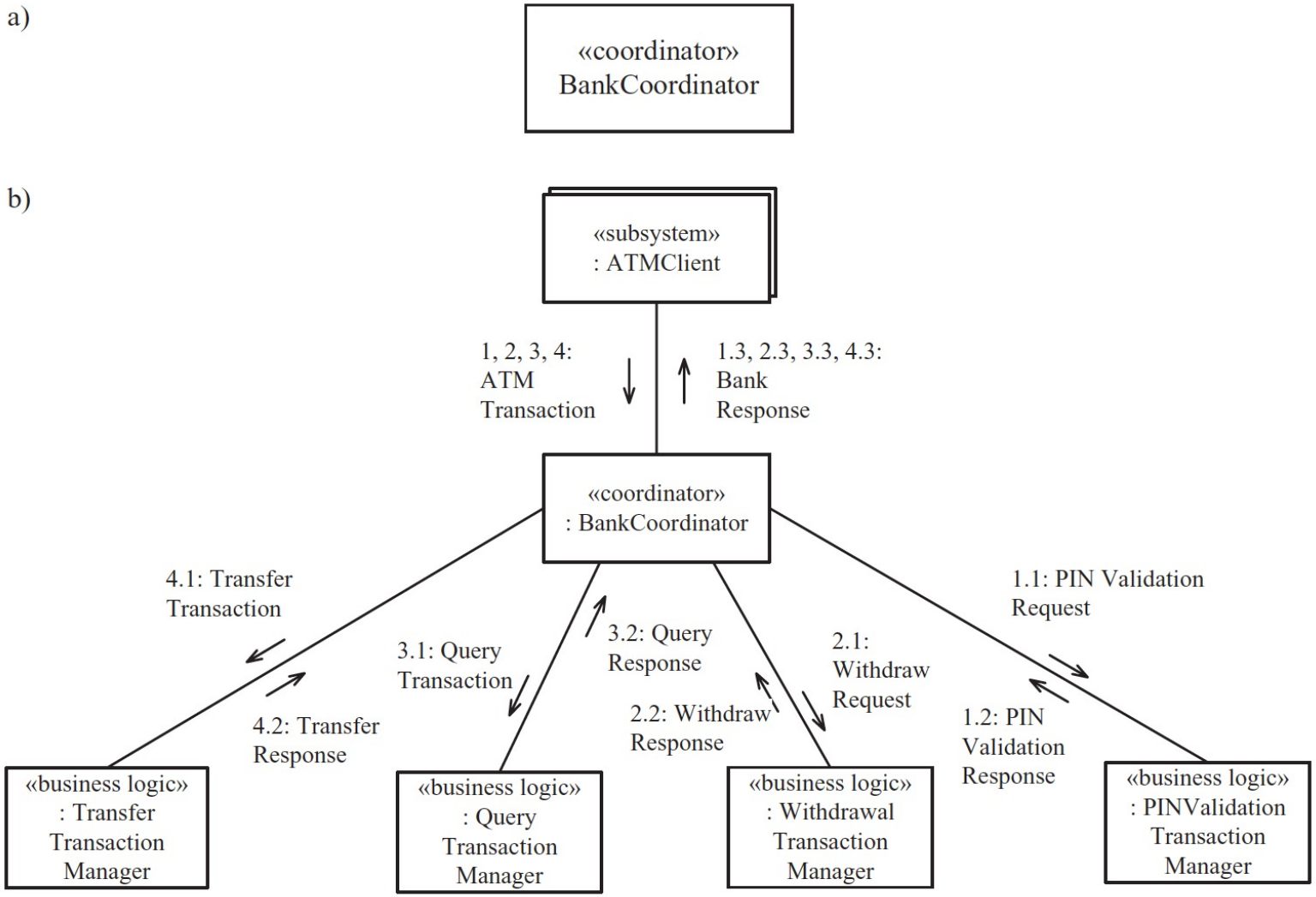


Figure 8.10. Example of coordinator class and object

Control Classes: *State-Dependent Class*

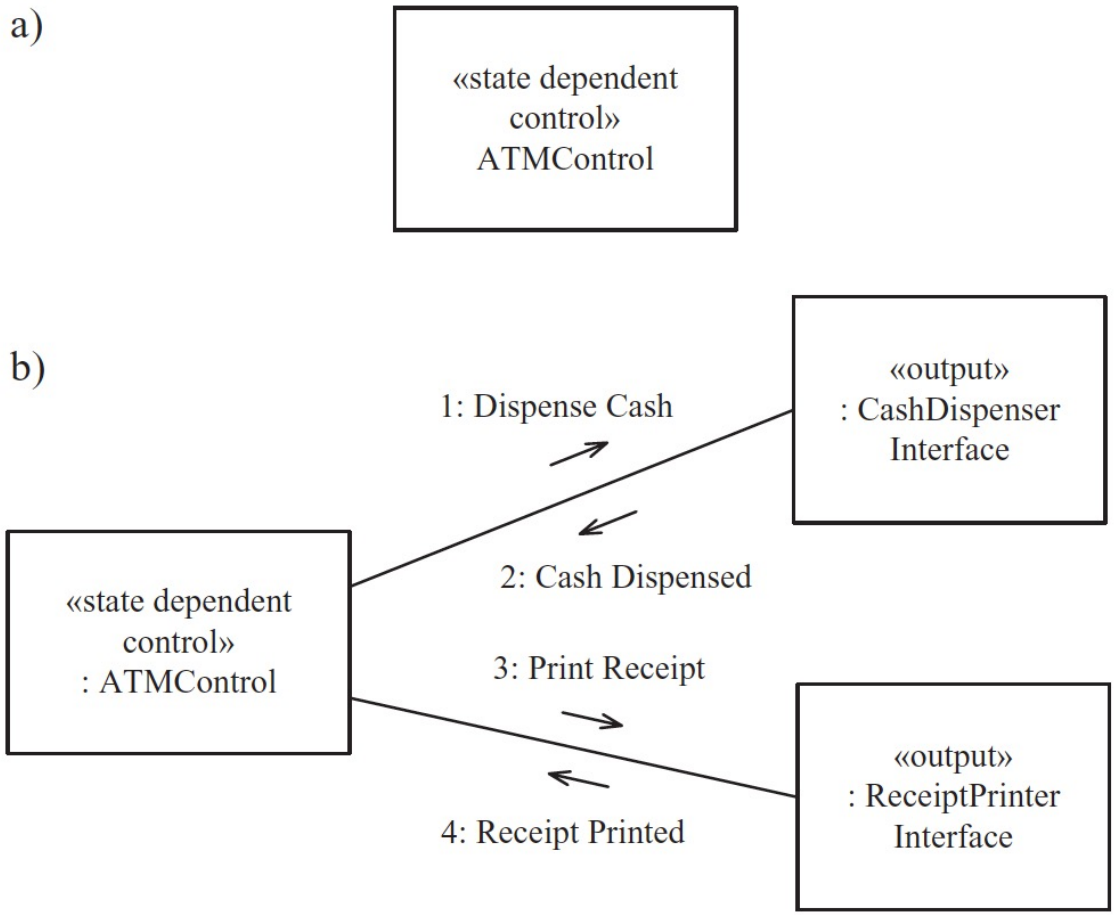


Figure 8.11. Example of state-dependent control class and object

Application Logic Classes: *Business Logic Class*

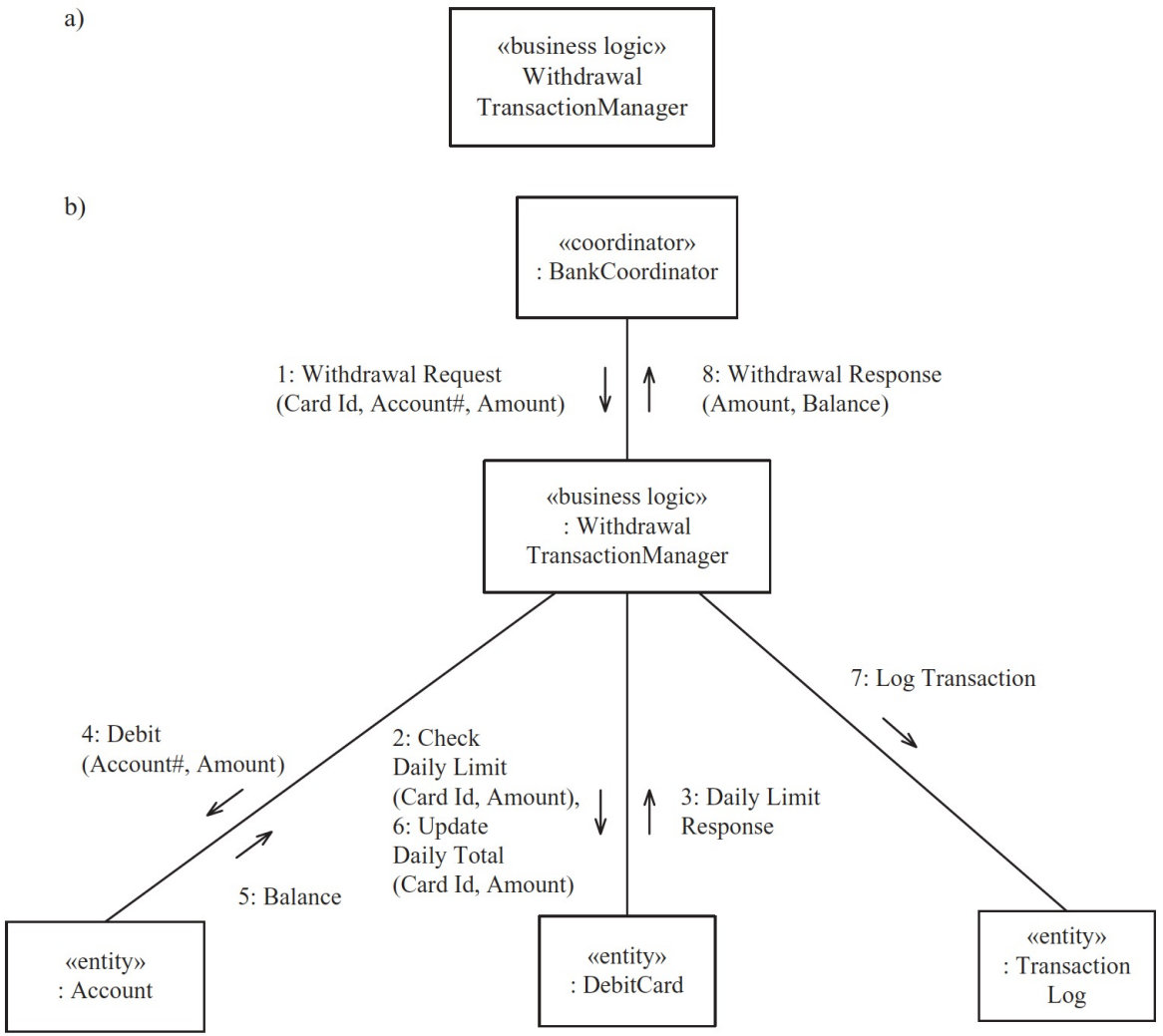


Figure 8.13. Example of business logic class and object

Application Logic Classes: *Algorithm Class*

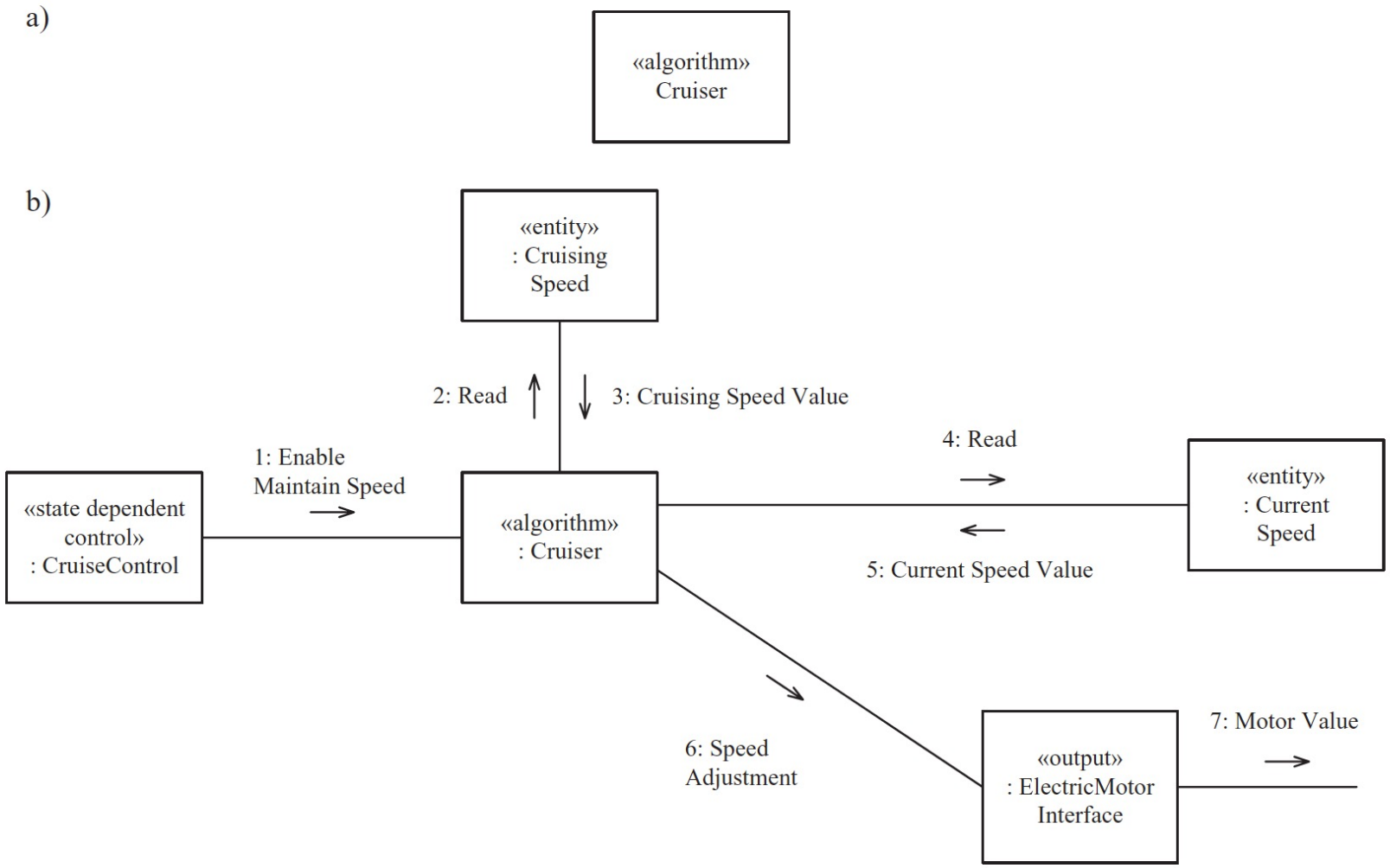


Figure 8.14. Example of algorithm class and object

Application Logic Classes: *Service Class*

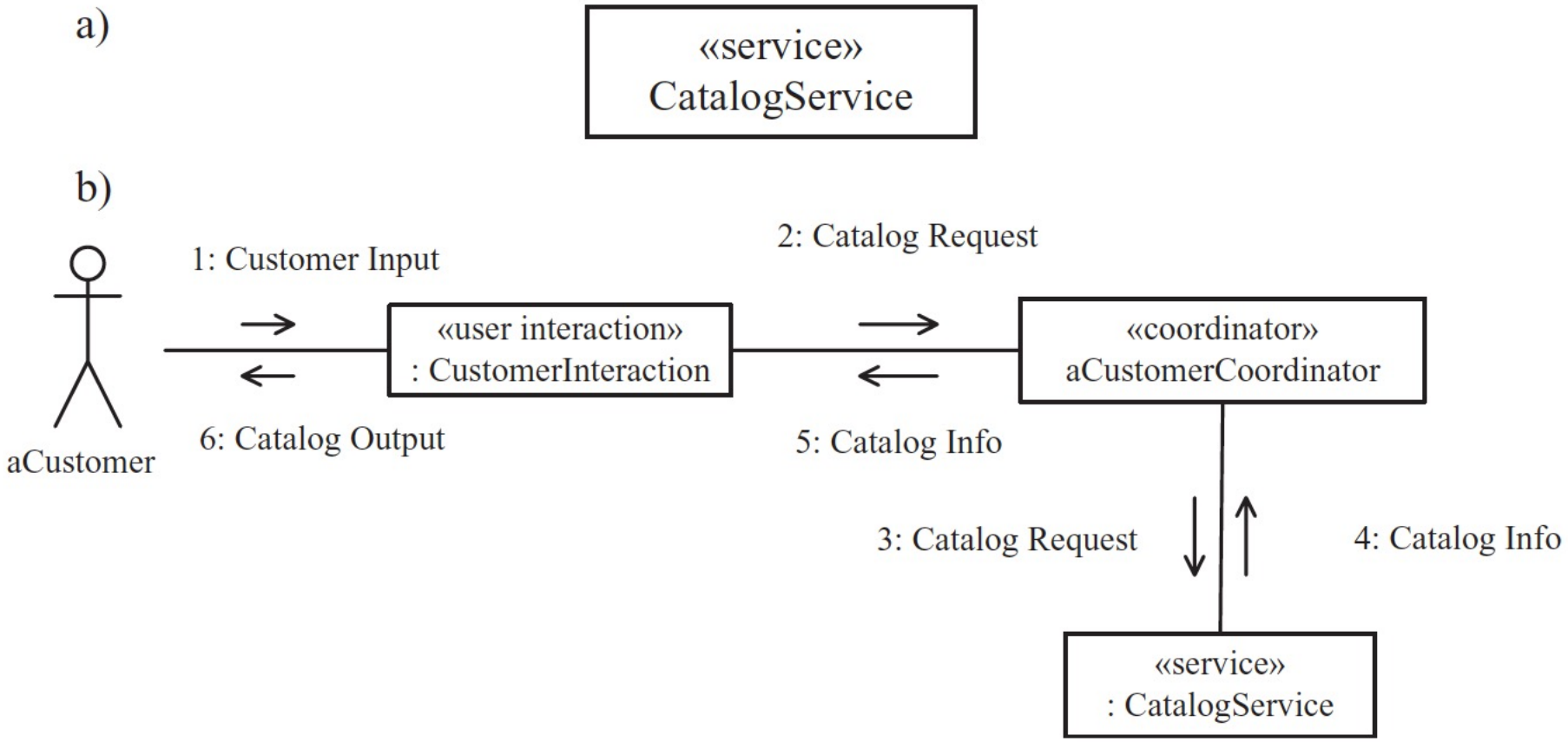
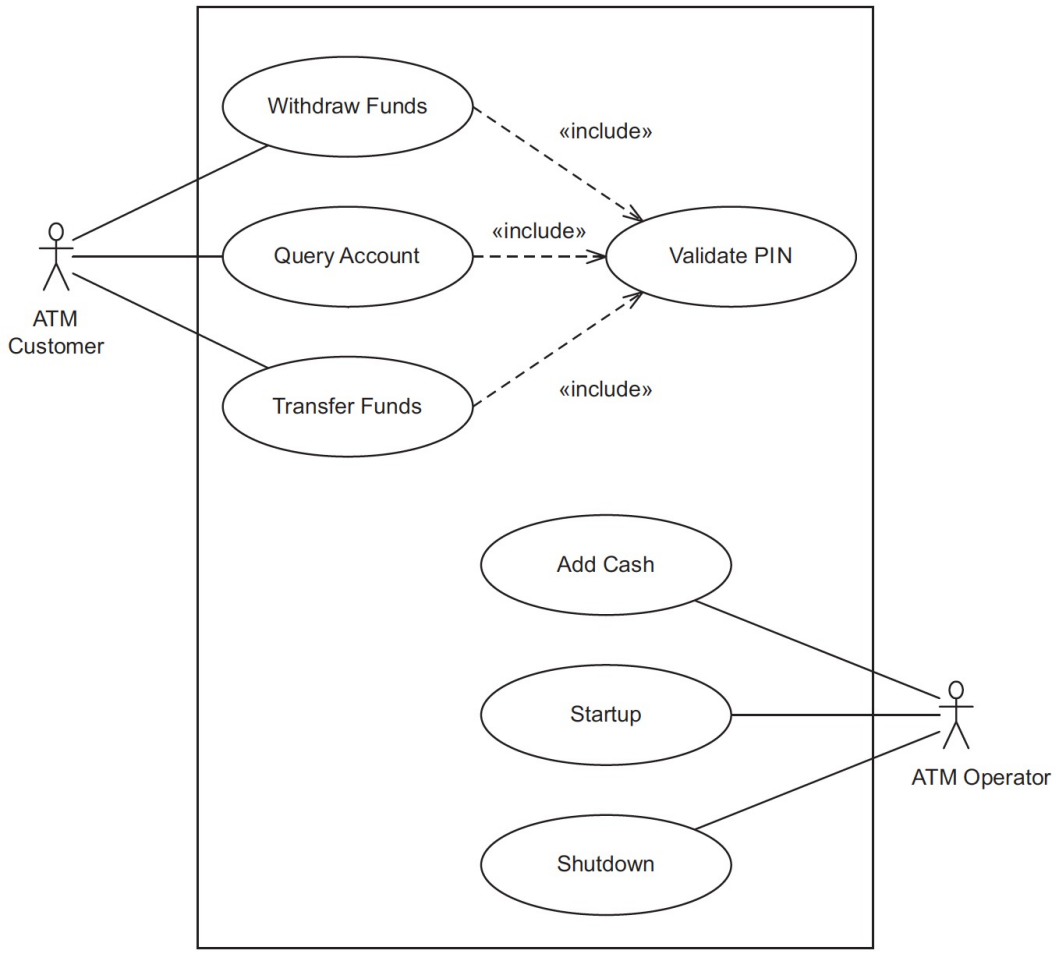


Figure 8.15. Example of service class and object

A bank has several automated teller machines (ATMs) that are geographically distributed and connected via a wide area network to a central server. Each ATM machine has a card reader, a cash dispenser, a keyboard/display, and a receipt printer. By using the ATM machine, a customer can withdraw cash from either a checking or savings account, query the balance of an account, or transfer funds from one account to another. A transaction is initiated when a customer inserts an ATM card into the card reader. Encoded on the magnetic strip on the back of the ATM card are the card number, the start date, and the expiration date. Assuming the card is recognized, the system validates the ATM card to determine that the expiration date has not passed, that the user-entered personal identification number, or PIN, matches the PIN maintained by the system, and that the card is not lost or stolen. The customer is allowed three attempts to enter the correct PIN; the card is confiscated if the third attempt fails. Cards that have been reported lost or stolen are also confiscated.

If the PIN is validated satisfactorily, the customer is prompted for a withdrawal, query, or transfer transaction. Before a withdrawal transaction can be approved, the system determines that sufficient funds exist in the requested account, that the maximum daily limit will not be exceeded, and that there are sufficient funds at the local cash dispenser. If the transaction is approved, the requested amount of cash is dispensed, a receipt is printed that contains information about the transaction, and the card is ejected. Before a transfer transaction can be approved, the system determines that the customer has at least two accounts and that there are sufficient funds in the account to be debited. For approved query and transfer requests, a receipt is printed and the card ejected. A customer may cancel a transaction at any time; the transaction is terminated, and the card is ejected. Customer records, account records, and debit card records are all maintained at the server.

An ATM operator may start up and close down the ATM to replenish the ATM cash dispenser and for routine maintenance. It is assumed that functionality to open and close accounts and to create, update, and delete customer and debit card records is provided by an existing system and is not part of this problem.



Use case name: Validate PIN
Summary: System validates customer PIN
Actor: ATM Customer
Precondition: ATM is idle, displaying a Welcome message.
Main sequence:

1. Customer inserts the ATM card into the card reader.
2. If system recognizes the card, it reads the card number.
3. System prompts customer for PIN.
4. Customer enters PIN.
5. System checks the card's expiration date and whether the card has been reported as lost or stolen.
6. If card is valid, system then checks whether the user-entered PIN matches the card PIN maintained by the system.
7. If PIN numbers match, system checks what accounts are accessible with the ATM card.
8. System displays customer accounts and prompts customer for transaction type: withdrawal, query, or transfer.

Alternative sequences:
Step 2: If the system does not recognize the card, the system ejects the card.
Step 5: If the system determines that the card date has expired, the system confiscates the card.
Step 5: If the system determines that the card has been reported lost or stolen, the system confiscates the card.
Step 7: If the customer-entered PIN does not match the PIN number for this card, the system re-prompts for the PIN.
Step 7: If the customer enters the incorrect PIN three times, the system confiscates the card.
Steps 4–8: If the customer enters Cancel, the system cancels the transaction and ejects the card.
Postcondition: Customer PIN has been validated.

Use case name: Withdraw Funds
Summary: Customer withdraws a specific amount of funds from a valid bank account.
Actor: ATM Customer
Dependency: Include `Validate PIN` use case.
Precondition: ATM is idle, displaying a Welcome message.

Main sequence:

1. Include `Validate PIN` use case.
2. Customer selects Withdrawal, enters the amount, and selects the account number.
3. System checks whether customer has enough funds in the account and whether the daily limit will not be exceeded.
4. If all checks are successful, system authorizes dispensing of cash.
5. System dispenses the cash amount.
6. System prints a receipt showing transaction number, transaction type, amount withdrawn, and account balance.
7. System ejects card.
8. System displays Welcome message.

Alternative sequences:
Step 3: If the system determines that the account number is invalid, then it displays an error message and ejects the card.
Step 3: If the system determines that there are insufficient funds in the customer's account, then it displays an apology and ejects the card.
Step 3: If the system determines that the maximum allowable daily withdrawal amount has been exceeded, it displays an apology and ejects the card.
Step 5: If the ATM is out of funds, the system displays an apology, ejects the card, and shuts down the ATM.
Postcondition: Customer funds have been withdrawn.

Use case name: Query Account
Summary: Customer receives the balance of a valid bank account.
Actor: ATM Customer
Dependency: Include `Validate PIN` use case.
Precondition: ATM is idle, displaying a Welcome message.
Main sequence:

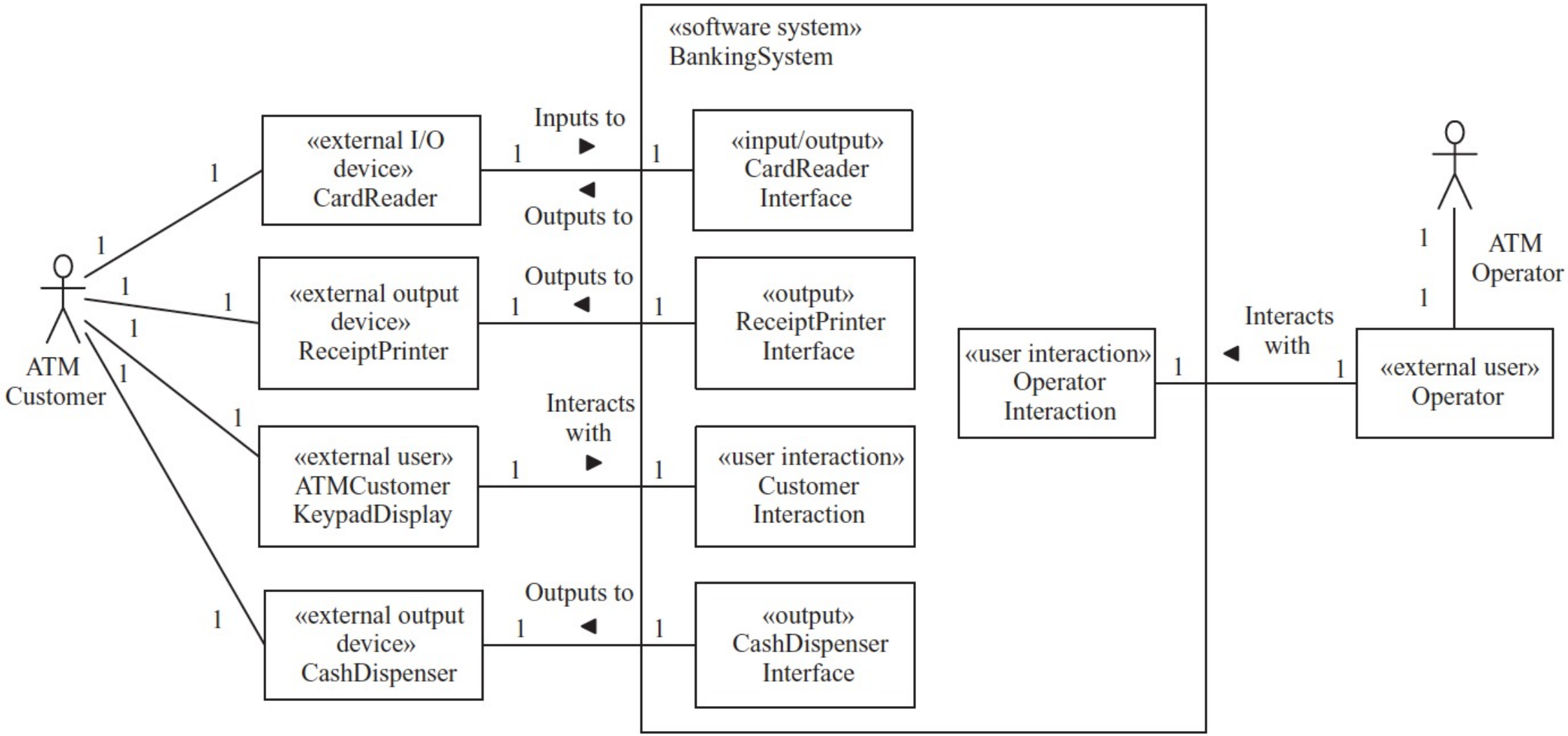
1. Include `Validate PIN` use case.
2. Customer selects Query, enters account number.
3. System reads account balance.
4. System prints a receipt that shows transaction number, transaction type, and account balance.
5. System ejects card.
6. System displays Welcome message.

Alternative sequence:
Step 3: If the system determines that the account number is invalid, it displays an error message and ejects the card.
Postcondition: Customer account has been queried.

Use case name: Transfer Funds
Summary: Customer transfers funds from one valid bank account to another.
Actor: ATM Customer
Dependency: Include `Validate PIN` use case.
Precondition: ATM is idle, displaying a Welcome message.
Main sequence:

1. Include `Validate PIN` use case.
2. Customer selects Transfer and enters `amount`, `from account`, and `to account`.
3. If the system determines the customer has enough funds in the `from account`, it performs the transfer.
4. System prints a receipt that shows transaction number, transaction type, amount transferred, and account balance.
5. System ejects card.
6. System displays Welcome message.

Alternative sequences:
Step 3: If the system determines that the `from account` number is invalid, it displays an error message and ejects the card.
Step 3: If the system determines that the `to account` number is invalid, it displays an error message and ejects the card.
Step 3: If the system determines that there are insufficient funds in the customer's `from account`, it displays an apology and ejects the card.
Postcondition: Customer funds have been transferred.


«software system»
BankingSystem
ATM
Customer
«external I/O
device»
CardReader
Inputs to
Outputs to
«input/output»
CardReader
Interface
«external output
device»
ReceiptPrinter
Outputs to
«output»
ReceiptPrinter
Interface
«external user»
ATMCustomer
KeypadDisplay
Interacts
with
«user interaction»
Customer
Interaction
«external output
device»
CashDispenser
Outputs to
«output»
CashDispenser
Interface
«user interaction»
Operator
Interaction
Interacts
with
«external user»
Operator
ATM
Operator
1

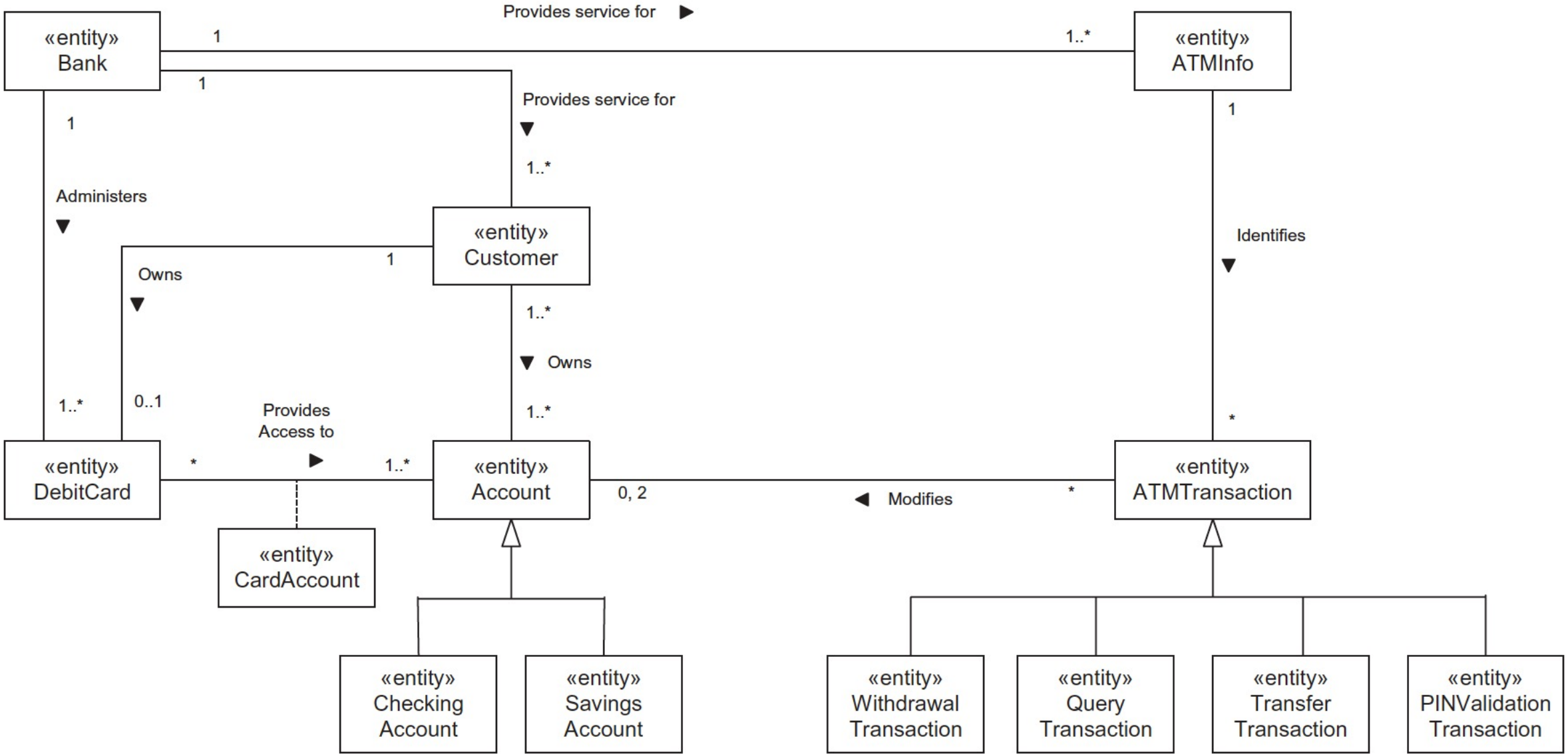
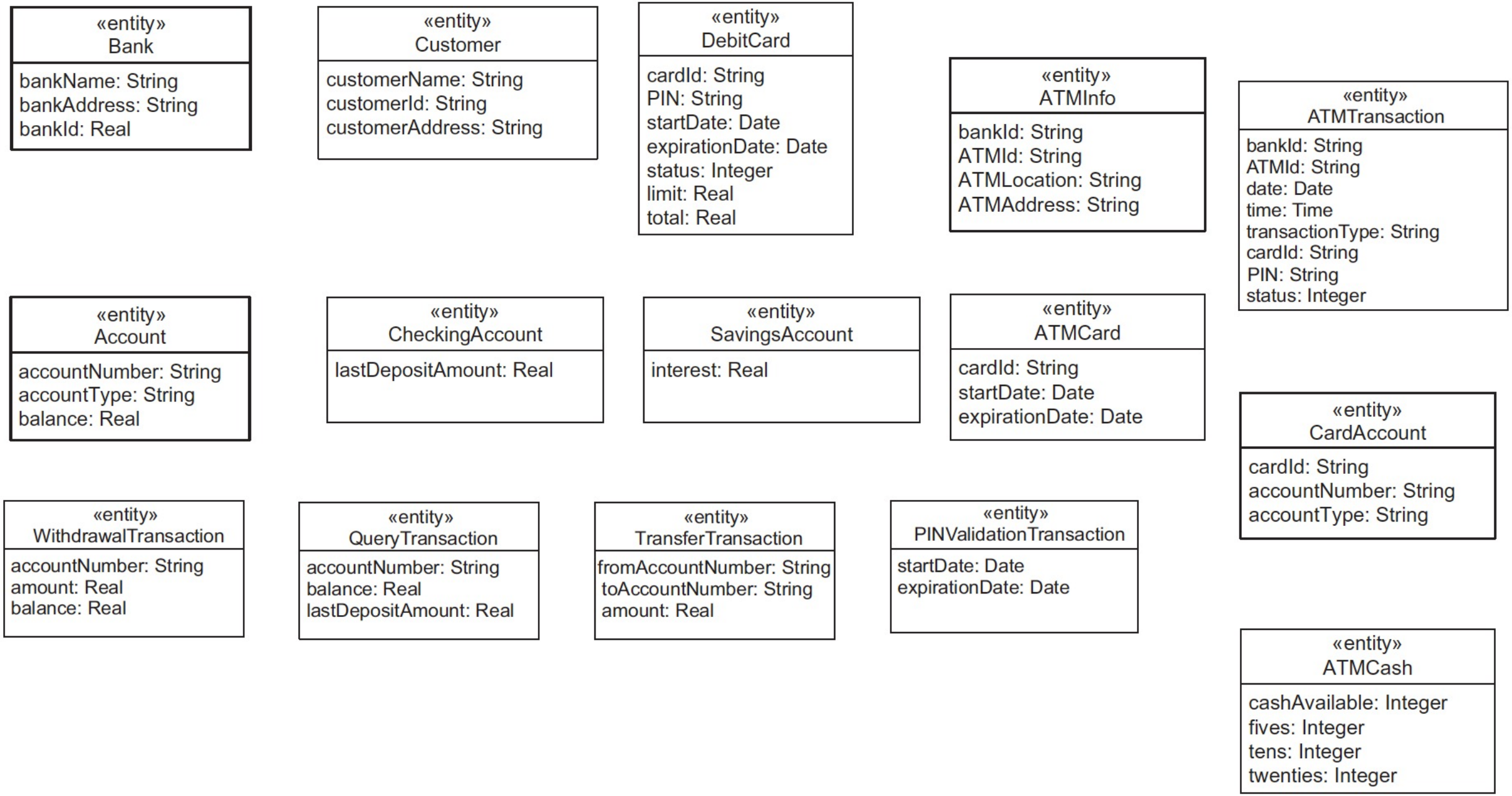


Figure 21.4. Conceptual static model for Banking System: entity classes


«entity»
Bank
bankName: String
bankAddress: String
bankId: Real
«entity»
Customer
customerName: String
customerId: String
customerAddress: String
«entity»
DebitCard
cardId: String
PIN: String
startDate: Date
expirationDate: Date
status: Integer
limit: Real
total: Real
«entity»
ATMInfo
bankId: String
ATMId: String
ATMLocation: String
ATMAddress: String
«entity»
ATMTransaction
bankId: String
ATMId: String
date: Date
time: Time
transactionType: String
cardId: String
PIN: String
status: Integer
«entity»
Account
accountNumber: String
accountType: String
balance: Real
«entity»
CheckingAccount
lastDepositAmount: Real
«entity»
SavingsAccount
interest: Real
«entity»
ATMCard
cardId: String
startDate: Date
expirationDate: Date
«entity»
CardAccount
cardId: String
accountNumber: String
accountType: String
«entity»
WithdrawalTransaction
accountNumber: String
amount: Real
balance: Real
«entity»
QueryTransaction
accountNumber: String
balance: Real
lastDepositAmount: Real
«entity»
TransferTransaction
fromAccountNumber: String
toAccountNumber: String
amount: Real
«entity»
PINValidationTransaction
startDate: Date
expirationDate: Date
«entity»
ATMCash
cashAvailable: Integer
fives: Integer
tens: Integer
twenties: Integer

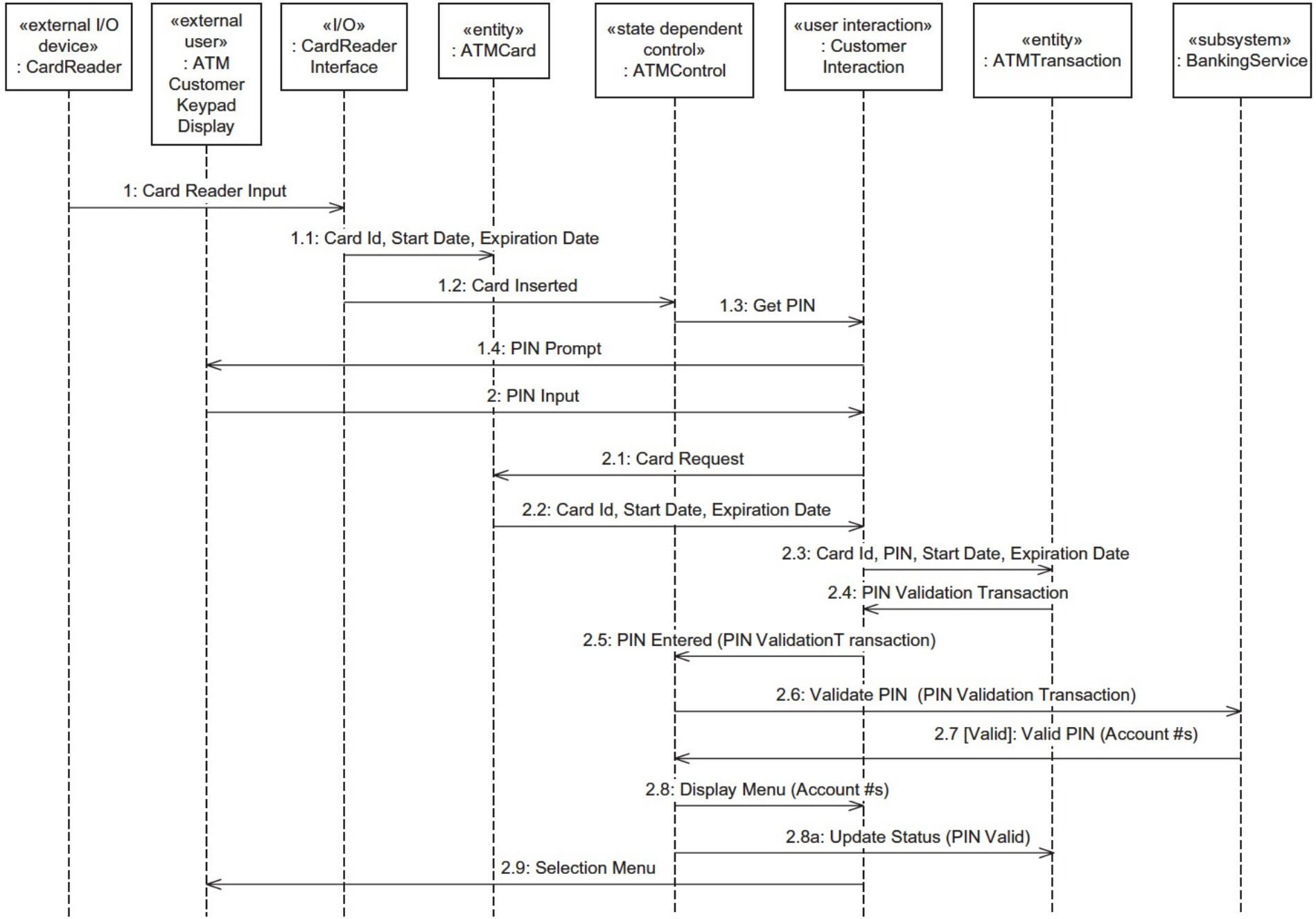


Figure 21.12. Sequence diagram: ATM client Validate PIN use case

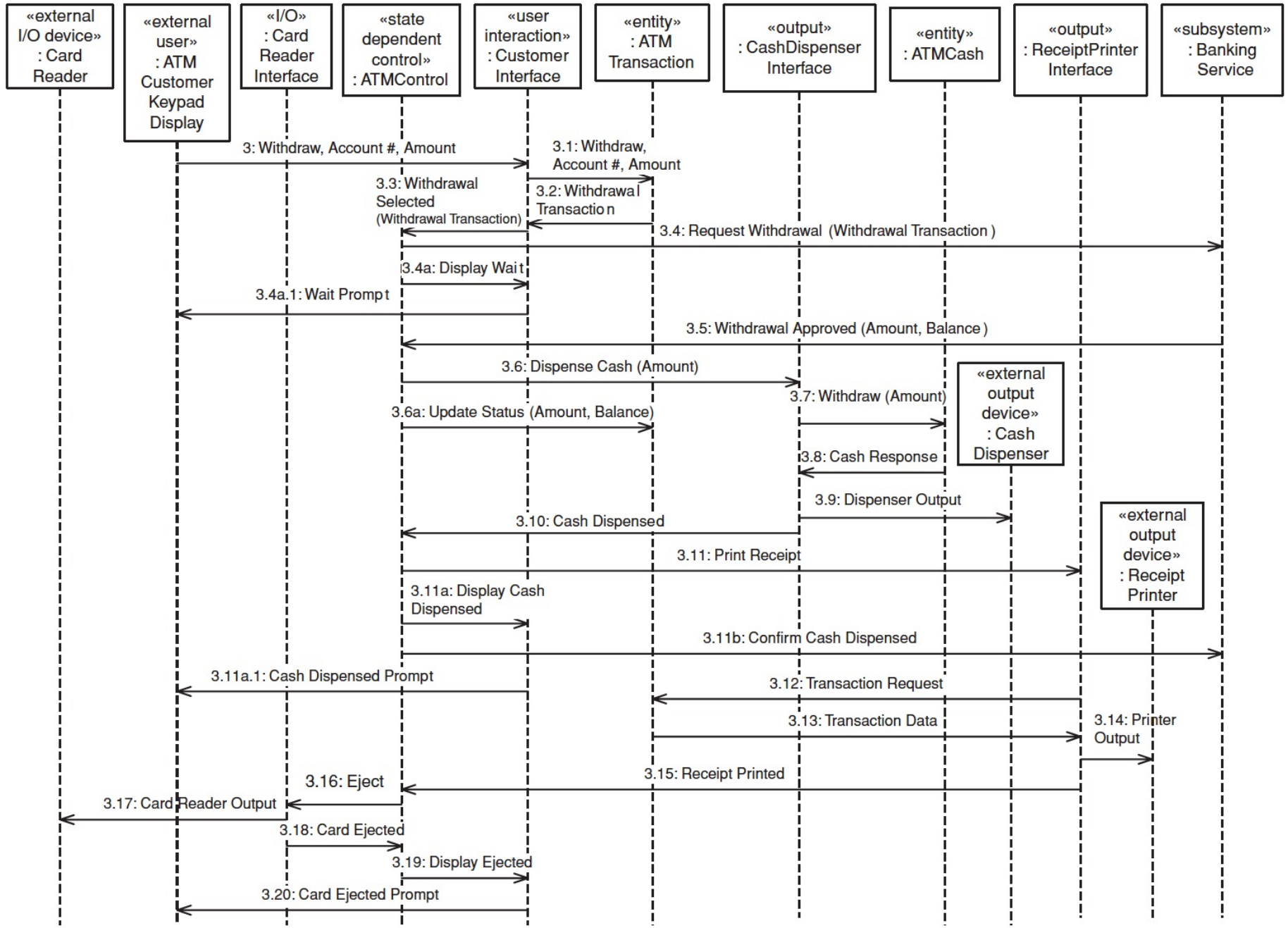
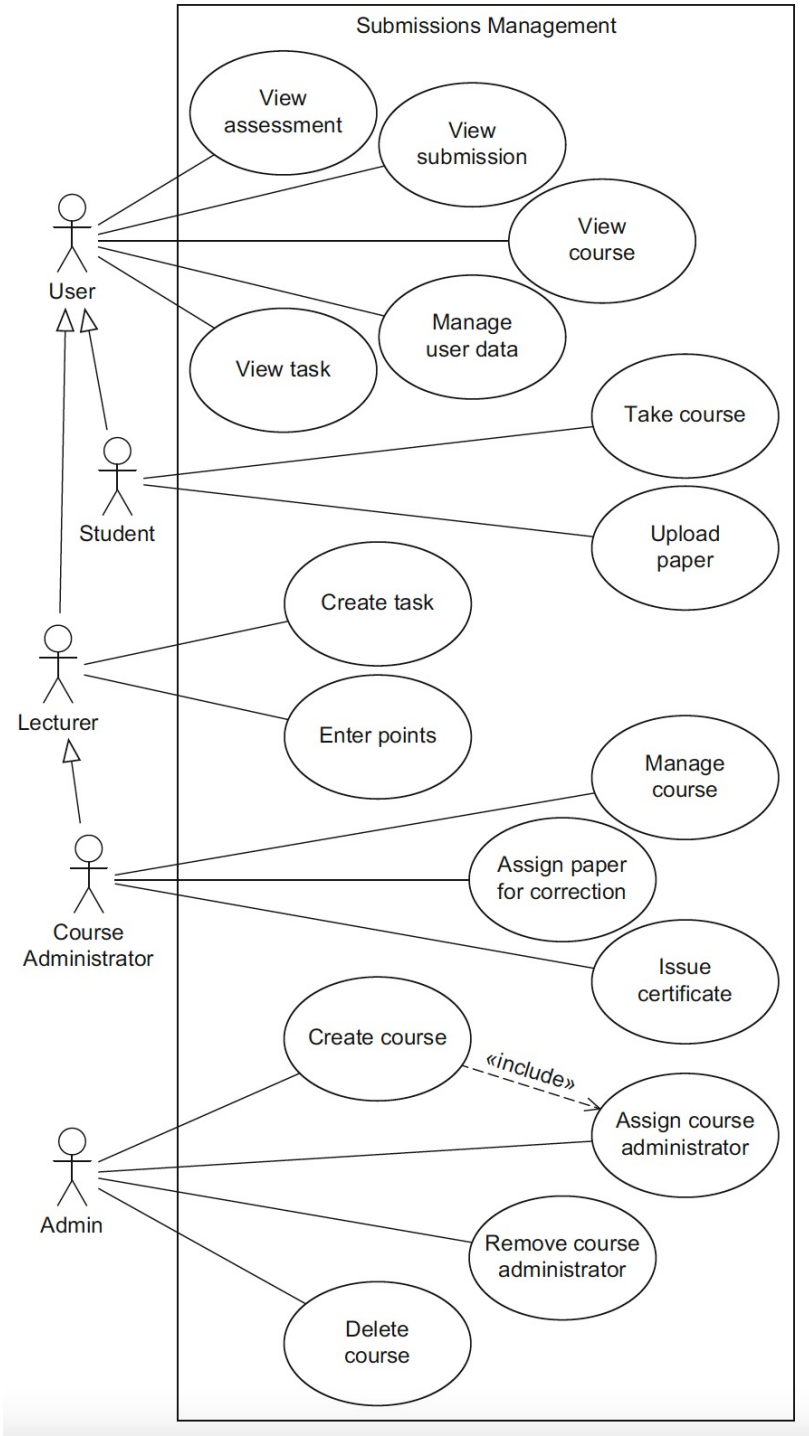
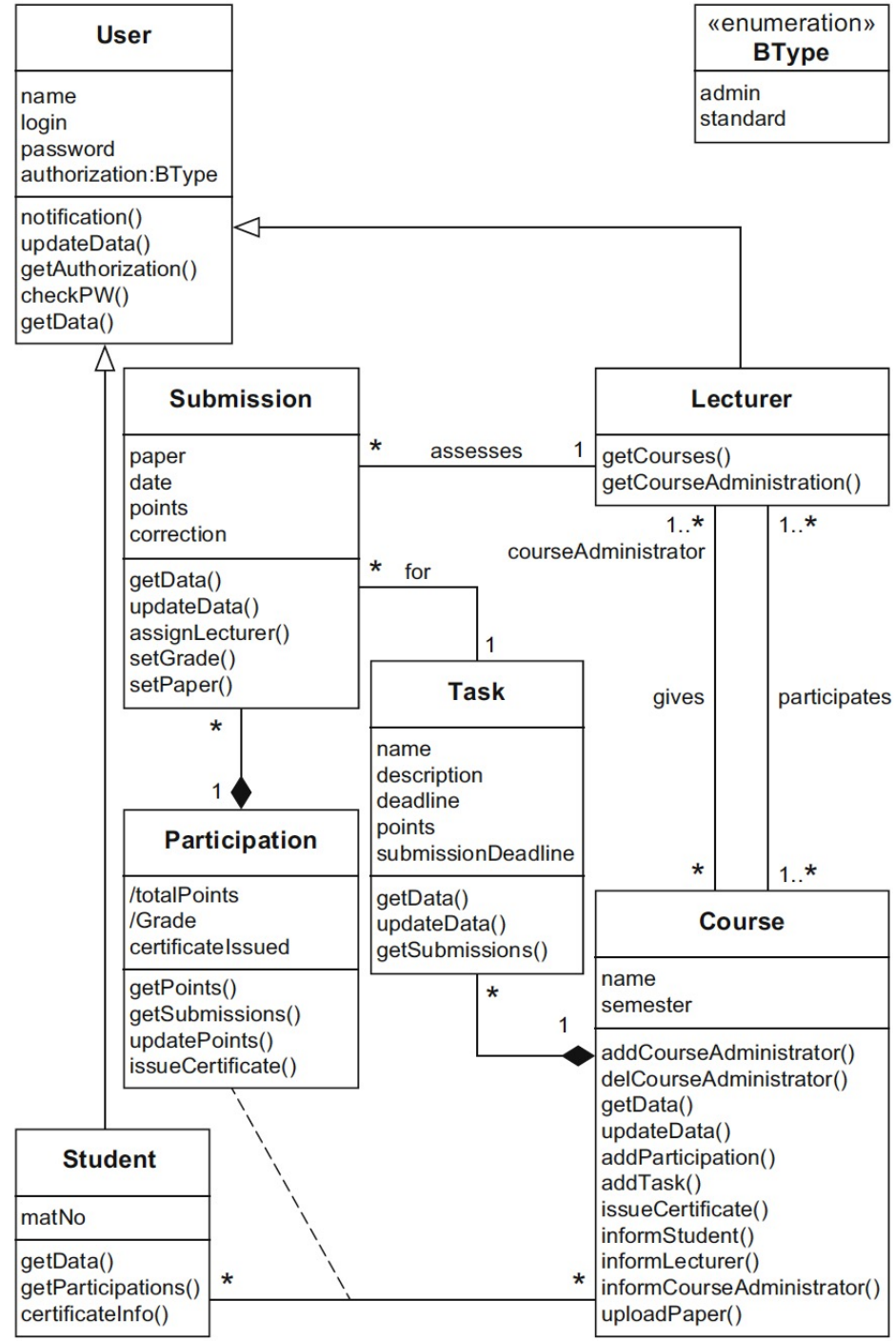


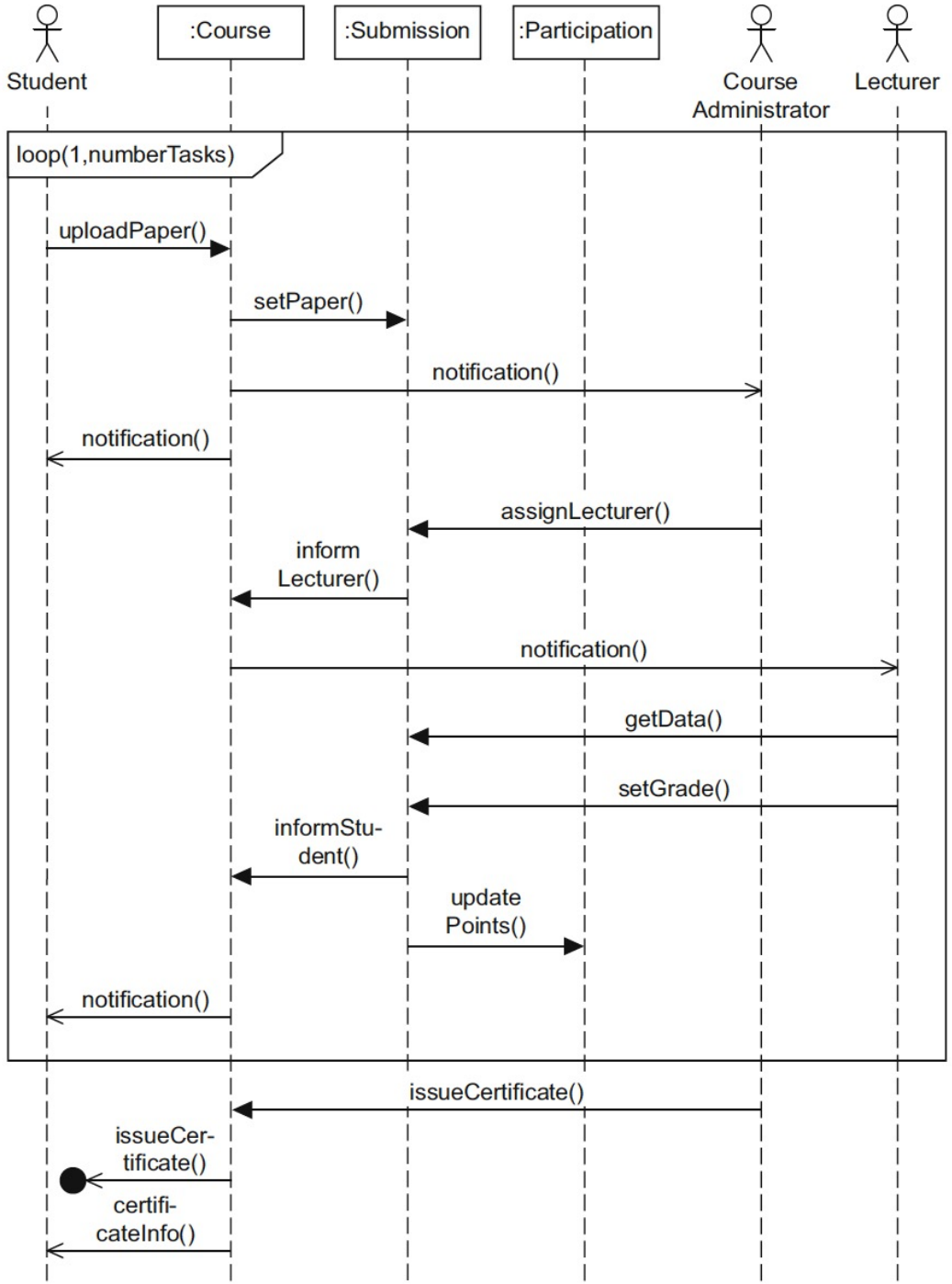
Figure 21.17. Sequence diagram: ATM client Withdraw Funds use case

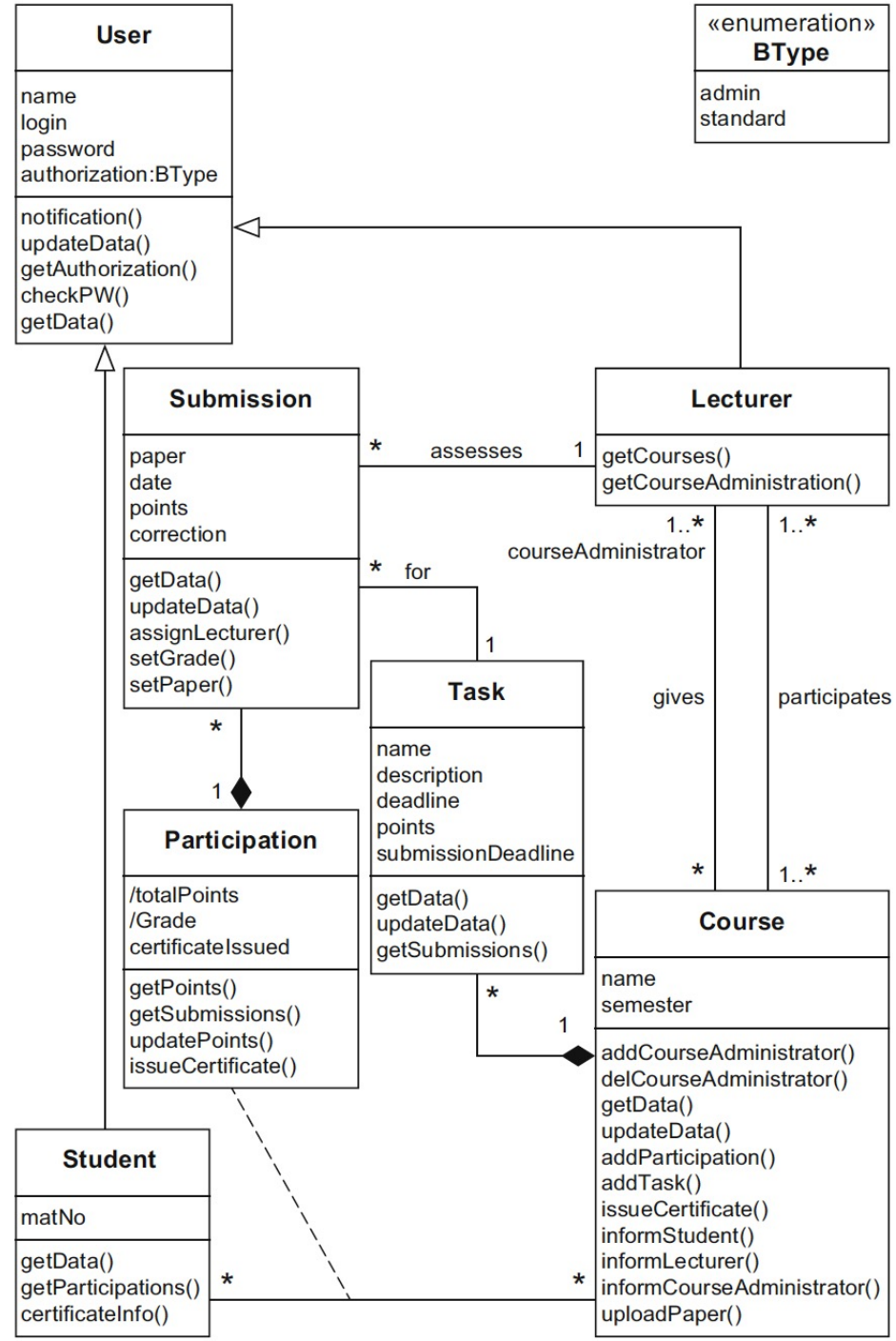
- Every course in the system has lecturers assigned to it. This is done by one of the course administrators, who is also a lecturer. As part of a course, lecturers may create tasks and assess papers submitted by students. Therefore, the lecturers award points and give feedback.
- The course administrator defines which lecturer assesses which papers. At the end of the course, the course administrator also arranges for certificates to be issued. A student's grade is calculated based on the total number of points achieved for the submissions handed in.
- Students can take courses and upload papers.
- All users—students and lecturers—can manage their user data, view the courses and the tasks set for the courses (provided the respective user is involved in the course), and view submitted papers as well as grade points. However, students can only view their own papers and the related grades. Lecturers can only view the papers assigned to them and the grades they have given. The course administrator has access rights for all data.
- A course is created and deleted by an administrator.
- When a course is created, at least one administrator must be assigned to it. Further course administrators can be assigned at a later point in time or assignments to courses can be deleted. The administrator can also delete whole courses.
- Information about users and administrators is automatically transferred from another system. Therefore, functions that allow the creation of user data are not necessary.
- All of the system functions can only be used by persons who are logged in.



- Every course in the system has lecturers assigned to it. This is done by one of the course administrators, who is also a lecturer. As part of a course, lecturers may create tasks and assess papers submitted by students. Therefore, the lecturers award points and give feedback.
- The course administrator defines which lecturer assesses which papers. At the end of the course, the course administrator also arranges for certificates to be issued. A student's grade is calculated based on the total number of points achieved for the submissions handed in.
- Students can take courses and upload papers.
- All users—students and lecturers—can manage their user data, view the courses and the tasks set for the courses (provided the respective user is involved in the course), and view submitted papers as well as grade points. However, students can only view their own papers and the related grades. Lecturers can only view the papers assigned to them and the grades they have given. The course administrator has access rights for all data.
- A course is created and deleted by an administrator.
- When a course is created, at least one administrator must be assigned to it. Further course administrators can be assigned at a later point in time or assignments to courses can be deleted. The administrator can also delete whole courses.
- Information about users and administrators is automatically transferred from another system. Therefore, functions that allow the creation of user data are not necessary.
- All of the system functions can only be used by persons who are logged in.




Student
:Course
:Submission
:Participation
Course Administrator
Lecturer
loop(1,numberTasks)
uploadPaper()
setPaper()
notification()
notification()
assignLecturer()
inform Lecturer()
notification()
getData()
setGrade()
informStu-dent()
update Points()
notification()
issueCertificate()
issueCer-tificate()
certifi-cateInfo()


User
name
login
password
authorization:BType
notification()
updateData()
getAuthorization()
checkPW()
getData()
«enumeration»
BType
admin
standard
Submission
paper
date
points
correction
getData()
updateData()
assignLecturer()
setGrade()
setPaper()
assesses
for
Lecturer
getCourses()
getCourseAdministration()
courseAdministrator
gives
participates
Task
name
description
deadline
points
submissionDeadline
getData()
updateData()
getSubmissions()
Participation
/totalPoints
/Grade
certificateIssued
getPoints()
getSubmissions()
updatePoints()
issueCertificate()
Course
name
semester
addCourseAdministrator()
delCourseAdministrator()
getData()
updateData()
addParticipation()
addTask()
issueCertificate()
informStudent()
informLecturer()
informCourseAdministrator()
uploadPaper()
Student
matNo
getData()
getParticipations()
certificateInfo()