21

Client/Server Software Architecture Case Study

Banking System

This chapter describes how the COMET/UML software modeling and design method is applied to the design of a client/server software architecture (see Chapter 15): a Banking System. In addition, the design of the ATM Client is an example of concurrent software design (see Chapter 18), and the design of the Banking Service is an example of sequential object-oriented software design (see Chapter 14).

The problem description is given in Section 21.1. Section 21.2 describes the use case model for the Banking System. Section 21.3 describes the static model, covering static modeling of both the system context and entity classes. Section 21.4 describes how to structure the system into objects. Section 21.5 describes dynamic modeling, in which interaction diagrams are developed for each of the use cases. Section 21.6 describes the ATM statechart. Sections 21.7 through 21.14 describe the design model for the Banking System.

21.1 PROBLEM DESCRIPTION

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.

21.2 USE CASE MODEL

The use cases are described in the **use case model**. There are two actors, namely, the `ATM Customer` and the `Operator`, who are the users of the system. The customer can withdraw funds from a checking or savings account, query the balance of the account, and transfer funds from one account to another.

The customer interacts with the system via the ATM card reader and the keyboard. It is the customer who is the actor, not the card reader and keyboard; these input devices provide the means for the customer to initiate the use case and respond to prompts from the system. The printer and cash dispenser are output devices; they are not actors, because it is the customer who benefits from the use cases.

The ATM operator can shut down the ATM, replenish the ATM cash dispenser, and start the ATM. Because an actor represents a role played by a user, there can be multiple customers and operators.

Consider the ATM operator use cases. One option is to have one operator use case in which the operator shuts down the ATM, adds cash, and then starts up the ATM. However, because it is possible to shut down the machine for a hardware problem without adding cash, and to start up the machine after it goes down unexpectedly, it is more flexible to have three separate use cases instead of one. These use cases are to `Add Cash` (in order to replenish the ATM cash locally), `Startup`, and `Shutdown`, as shown in Figure 21.1.

Consider the use cases initiated by the `ATM Customer`. One possibility is to have one use case for all customer interactions. However, there are three separate, quite distinct transaction types for withdrawal, query, and transfer that can be initiated by a customer.

We therefore start by considering three separate use cases: `Withdraw Funds`, `Query Account`, and `Transfer Funds`, one for each transaction type. Consider the `Withdraw Funds` use case. In this use case, the main sequence assumes a successful cash withdrawal by the customer. This involves reading the ATM card, validating the customer's PIN, checking that the customer has enough funds in the requested account, and then – providing the validation is successful – dispensing cash, printing a receipt, and ejecting the card.

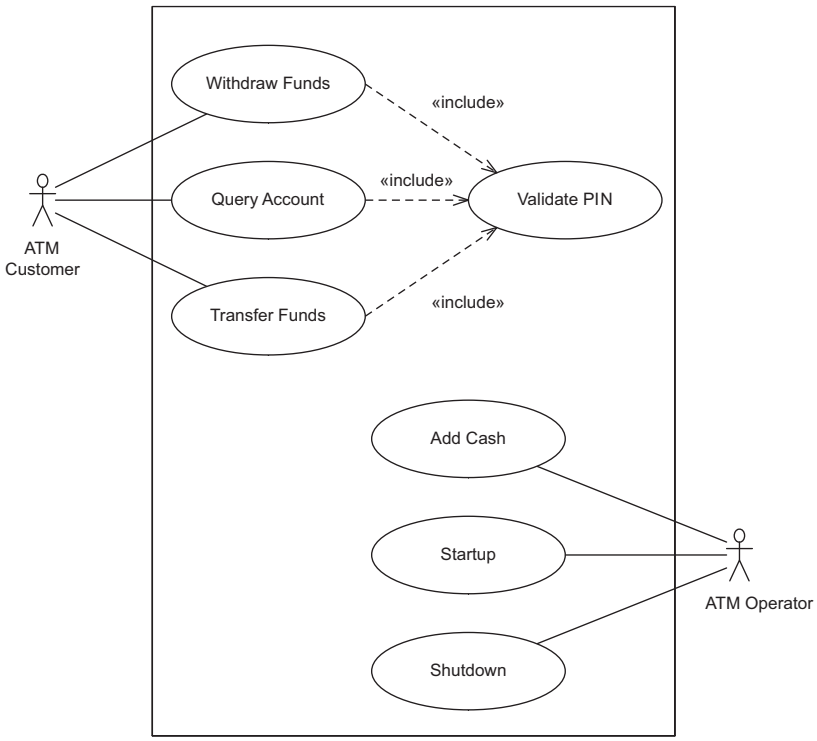


Figure 21.1. Banking System use case model

However, by comparing the three use cases, it can be seen that the first part of each use case – namely, reading the ATM card and validating the customer's PIN – is common to all three use cases. This common part of the three use cases is factored out as an inclusion use case called `Validate PIN`.

The `Withdraw Funds`, `Query Account`, and `Transfer Funds` use cases can then each be rewritten more concisely as concrete use cases that include the `Validate PIN` inclusion use case. The relationship between the use cases is shown in Figure 21.1. The concrete `Withdraw Funds` use case starts by including the description of the `Validate PIN` inclusion use case and then continues with the `Withdraw Funds` description. The concrete `Transfer Funds` use case also starts with the description of the `Validate PIN` inclusion use case, but then continues with the `Transfer Funds` description. The revised concrete `Query Account` use case is similarly organized. The inclusion use case and concrete use cases are described next.

The main sequence of the `Validate PIN` use case deals with reading the ATM card, validating the customer's PIN and card. If validation is successful, the system prompts the customer to select a transaction: withdrawal, query, or transfer. The alternative branches deal with all the possible error conditions, such as the customer enters the wrong PIN and must be re-prompted, or an ATM card is not recognized or has been reported stolen, and so on. Because these can be described quite simply in the alternative sequences, splitting them off into separate extension use cases is not necessary.

21.2.1 Validate PIN Use Case

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.

21.2.2 Withdraw Funds Concrete Use Case

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.

21.2.3 Query Account Concrete Use Case

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.

21.2.4 Transfer Funds Concrete Use Case

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.

21.3 STATIC MODELING

This section begins by considering the problem domain and the system context, and then continues with a discussion of static modeling of the entity classes. Refer also to Chapter 7, which describes static modeling in detail with some examples from the Banking System.

21.3.1 Static Modeling of the Problem Domain

The conceptual static model of the problem domain is given in the class diagram depicted in Figure 21.2. A bank has several ATMs. Each ATM is modeled as a composite class consisting of a `Card Reader`, a `Cash Dispenser`, a `Receipt Printer`, and a keyboard/display through which the user interacts, the `ATM Customer Keyboard Display`. The `ATM Customer` actor inserts the card into the `Card Reader` and responds to system prompts though the `ATM Customer Keyboard Display`. The `Cash Dispenser` dispenses cash to the `ATM Customer` actor. The `Receipt Printer` prints a receipt for the `ATM Customer` actor. In addition, the `Operator` actor is a user whose job is to maintain the ATM.

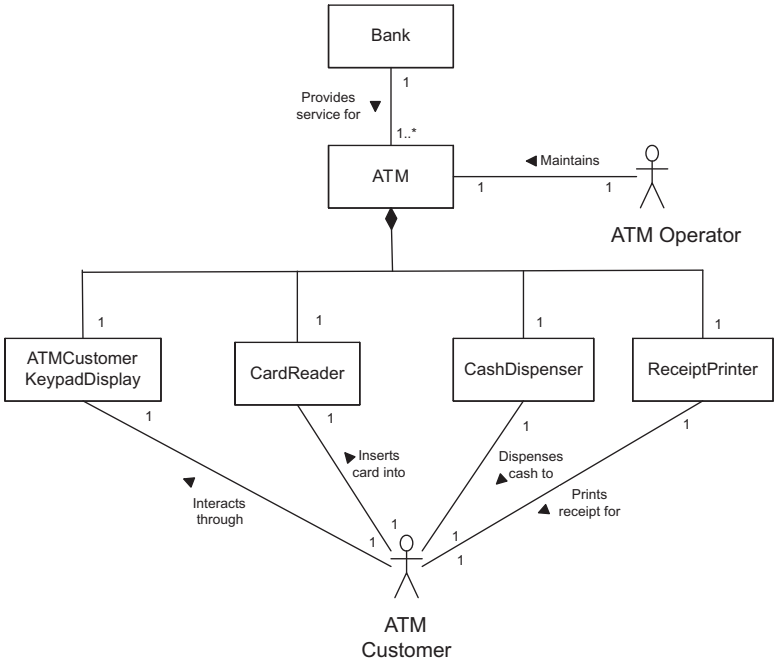


Figure 21.2. Conceptual static model for problem domain

21.3.2 Static Modeling of the System Context

The software system context class diagram, which uses the static modeling notation, is developed to show the external classes to which the Banking System, shown as one aggregate class, has to interface. We develop the context class diagram by considering the physical classes determined during static modeling of the problem domain, as described in detail in Chapter 7.

From the total system perspective – that is, both hardware and software – the ATM Customer and ATM Operator actors are external to the system, as shown in Figure 7.19. The ATM Operator interacts with the system via a keypad and display. The ATM Customer actor interacts with the system via four I/O devices, which are the card reader, cash dispenser, receipt printer, and ATM Customer keypad/display. From a total hardware/software system perspective, these I/O devices are part of the system. From a software perspective, the I/O devices are external to the software system. On the software system context class diagram, the I/O devices are modeled as external classes, as shown on Figure 21.3.

The four external classes used by the ATM Customer actor are the Card Reader, the Cash Dispenser, the Receipt Printer, and the ATM Customer Keypad/Display; the Operator interacts with the system via a keyboard/display. Both Customer Keypad/Display and Operator are modeled as external users, as described in Chapter 7. There is one instance of each of these external classes for each ATM. The software system context class diagram for the Banking System (see Figure 21.3) depicts the software system as one aggregate class that receives input from and provides output to the external classes.

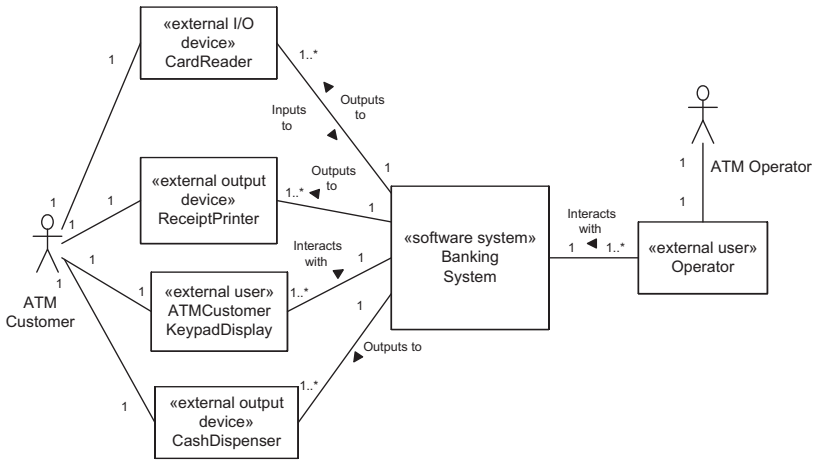


Figure 21.3. Banking System software context class diagram

21.3.3 Static Modeling of the Entity Classes

The static model of the entity classes, referred to as the entity class model, is shown in Figure 21.4. The attributes of each entity class are given in Figures 21.5, 21.6, and 21.7.

Figure 21.4 shows the Bank entity class, which has a one-to-many relationship with the Customer class and the Debit Card class. The Bank class is unusual in that it will only have one instance; its attributes are the bank Name, bank Address, and bank Id. The Customer has a many-to-many relationship with Account. Because there are both checking accounts and savings accounts, which have some common attributes, the Account class is specialized to be either a Checking Account or a Savings Account. Thus, some attributes are common to all accounts, namely, the account Number, account Type, and balance. Other attributes are specific to Checking Account (e.g., last Deposit Amount) and Savings Account (e.g., the accumulated interest).

An Account is modified by an ATM Transaction, which is specialized to depict the different types of transactions as a Withdrawal Transaction, Query Transaction, Transfer Transaction, or PIN Validation Transaction. The common attributes of a transaction are in the superclass ATM Transaction and consist of transaction Id (which actually consists of four concatenated attributes – bank Id, ATM Id, date, and time), transaction Type, card Id, PIN, and status. Other attributes are specific to the particular type of transaction. Thus, for the Withdrawal Transaction, the specific attributes maintained by the subclass are account Number, amount, and balance. For a Transfer Transaction, the attributes maintained by the subclass are from Account Number (checking or savings), to Account Number (savings or checking), and amount.

There is also a Card Account association class. Association classes are needed in cases in which the attributes are of the association, rather than of the classes connected by the association. Thus, in the many-to-many association between Debit Card and Account, the individual accounts that can be accessed by a given debit card are attributes of the Card Account association class and not of either Debit Card or

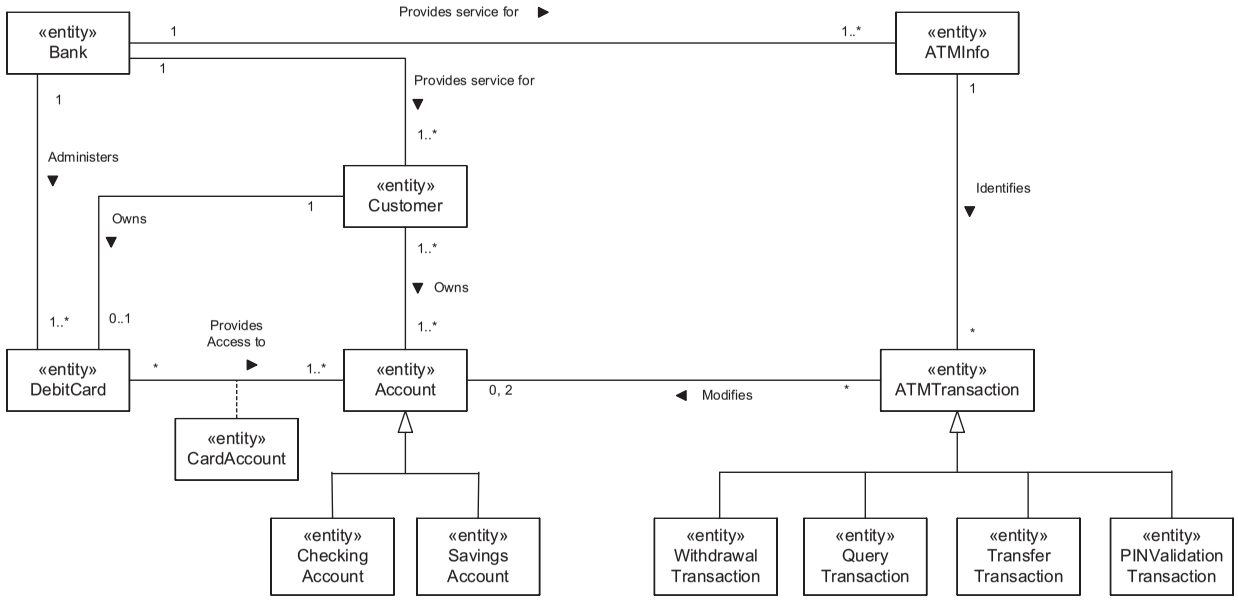


Figure 21.4. Conceptual static model for Banking System: entity classes

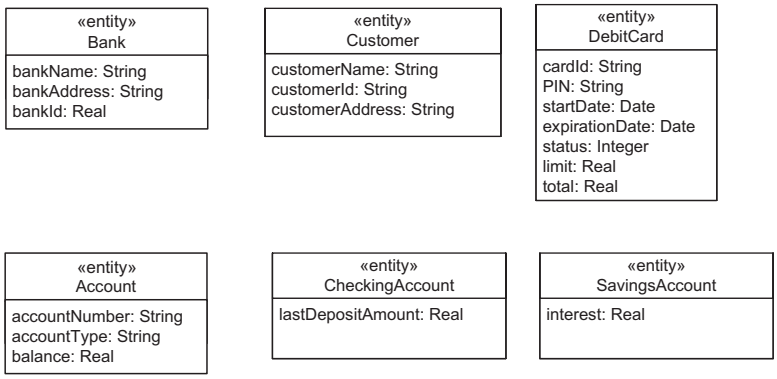


Figure 21.5. Conceptual static model for Banking System: class attributes

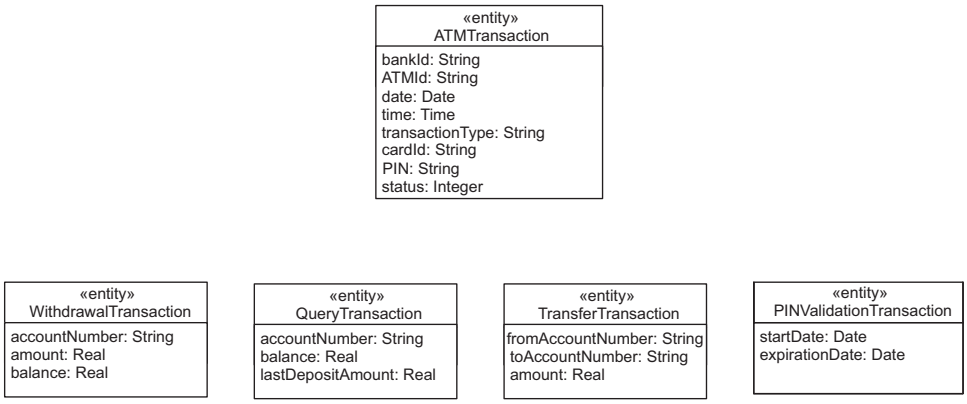


Figure 21.6. Conceptual static model for Banking System: class attributes (continued)

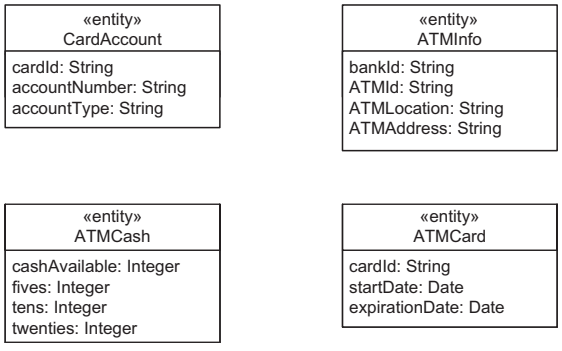


Figure 21.7. Conceptual static model for Banking System: class attributes (continued)

Account. The attributes of Card Account are Card Id, account Number, and account Type.

Entity classes are also required to model other information described in Section 21.2. These include ATM Card, which represents the information read off the magnetic strip on the plastic card. ATM Cash holds the amount of cash maintained at an ATM, in five-, ten-, and twenty-dollar bills. The Receipt holds information about a transaction, and because it holds similar information to the Transaction class described earlier, a separate entity class is unnecessary.

21.4 OBJECT STRUCTURING

We next consider structuring the system into objects in preparation for defining the dynamic model. The object structuring criteria help determine the objects in the system. After the objects and classes have been determined, a communication diagram or sequence diagram is developed for each use case to show the objects that participate in the use case and the dynamic sequence of interactions between them.

21.4.1 Client/Server Subsystem Structuring

Because the Banking System is a client/server application, some of the objects are part of the ATM client and some objects are part of the banking service, so we start by identifying subsystems, which are aggregate or composite objects. In client/server systems, the subsystems are often easily identifiable. Thus, in the Banking System, there is a client subsystem called ATM Client Subsystem, of which one instance is located at each ATM machine. There is also a service subsystem, the Banking Service Subsystem, of which there is one instance (Figure 21.8). This is an example of geographical subsystem structuring, in which the geographical distribution of the

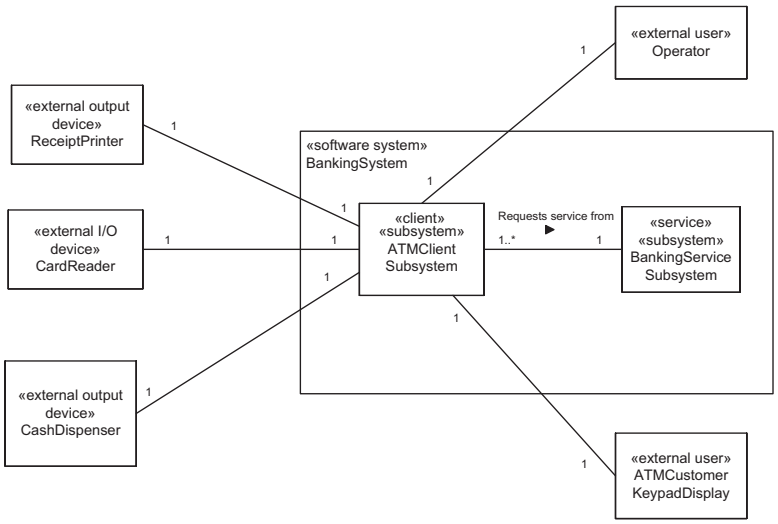


Figure 21.8. Banking System: major subsystems

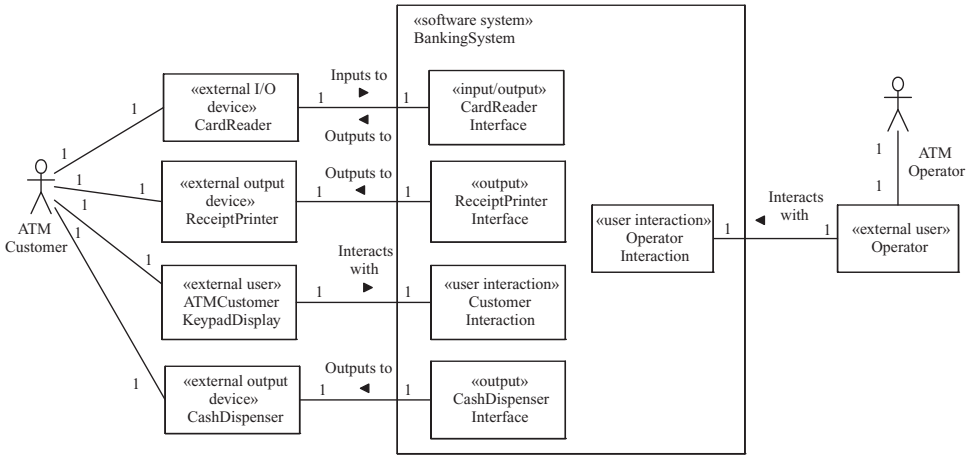


Figure 21.9. Banking System external classes and boundary classes

system is given in the problem description. Both subsystems are depicted as aggregate classes, with a one-to-many association between the `Banking Service Subsystem` and the `ATM Client Subsystem`. All the external classes interface to and communicate with the `ATM Client Subsystem`.

21.4.2 ATM Client Object and Class Structuring: Boundary Objects

The next step is to determine the software objects and classes at the `ATM Client`. First, consider the boundary objects and classes. The boundary classes are determined from the software system context diagram, as shown in Figure 21.9, which shows the Banking System as an aggregate class.

We design one boundary class for each external class. The device I/O classes are the `Card Reader Interface`, through which ATM cards are read, the `Cash Dispenser Interface`, which dispenses cash, and the `Receipt Printer Interface`, which prints receipts. There is also `Customer Interaction`, which is the user interaction class that interacts with the customer via the keyboard/display, displaying textual messages, prompting the customer, and receiving the customer's inputs. The `Operator Interaction` class is a user interaction class that interacts with the ATM operator, who replenishes the ATM machine with cash. There is one instance of each of these boundary classes for each ATM.

21.4.3 ATM Client Object and Class Structuring: Objects Participating in Use Cases

Next, consider the individual use cases and determine the objects that participate in them. First, consider the `Validate PIN` inclusion use case, which describes the customer inserting the ATM Card into the card reader, the system prompting for the PIN, and the system checking whether the customer-entered PIN matches the PIN maintained by the system for that ATM card number. From this use case, we first

determine the need for the Card Reader Interface object to read the ATM card. The information read off the ATM card needs to be stored, so we identify the need for an entity object to store the ATM Card information. The Customer Interaction object is used for interacting with the customer via the keyboard/display, in this case to prompt for the PIN. The information to be sent to the Banking Service Subsystem for PIN validation is stored in an ATM Transaction. For PIN validation, the transaction information needs to contain the PIN number and the ATM Card number. To control the sequence in which actions at the ATM take place, we identify the need for a control object, ATM Control.

Next consider the objects in the Withdraw Funds use case, which is entered if the PIN is valid and the customer selects withdrawal. In this use case, the customer enters the amount to be withdrawn and the account to be debited, the system checks whether the withdrawal should be authorized, and if positive, dispenses the cash, prints the receipt, and ejects the card. For this use case, additional objects are needed. The information about the customer withdrawal, including the account number and withdrawal amount, needs to be stored in the ATM Transaction object. To dispense the cash, a Cash Dispenser Interface object is needed. We also need to maintain the amount of cash in the ATM, so we identify the need for an entity object called ATM Cash, which is decremented every time there is a cash withdrawal. Finally, we need a Receipt Printer Interface object to print the receipt. As before, the ATM Control object controls the sequencing of the use case.

Inspecting the other use cases reveals that one additional object is needed, namely, the Operator Interaction object, which participates in all use cases initiated by the Operator actor. The Operator Interaction object needs to send startup and shutdown events to ATM Control, because operator maintenance and ATM customer activities are mutually exclusive.

Given the preceding analysis, Figure 21.10 shows the classes in the ATM Client Subsystem, which is depicted as an aggregate class. In addition to the three device

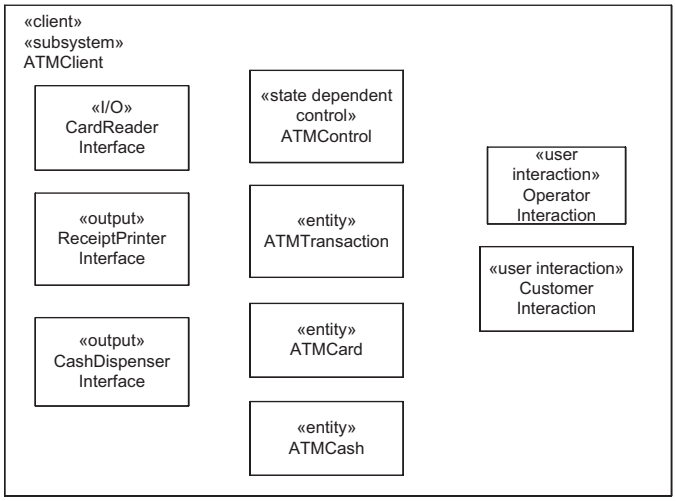


Figure 21.10. ATM Client subsystem classes

I/O classes and two user interaction classes depicted in Figure 21.9, there are also three entity classes and one state-dependent control class.

21.4.4 Object Structuring in Service Subsystem

Several entity objects are bank-wide and need to be accessible from any ATM. Consequently, these objects must be stored in the Banking Service subsystem at the server. These objects include Customer objects that hold information about bank customers, Account objects (both checking and saving) that hold information about individual bank accounts, and Debit Card objects that hold information about all the debit cards maintained at the bank. The classes from which these objects are instantiated all appear on the static model of the entity classes depicted in Figure 21.4.

In the Banking Service Subsystem, the entity classes are Customer, the Account superclass, Checking Account and Savings Account subclasses, and Debit Card. There is also the ATM Transaction object, which migrates from the client to the server. The client sends the transaction request to the Banking Service, which sends a response to the client. The transaction is also stored at the server as an entity object in the form of a Transaction Log, so that a transaction history is maintained. The transient data sent as part of the ATM Transaction message might differ from the persistent transaction data; for example, transaction status is known at the end of the transaction but not during it.

Business logic objects are also needed at the server to define the business-specific application logic for processing client requests. In particular, each ATM transaction type needs a transaction manager to specify the business rules for handling the transaction. The business logic objects are the PIN Validation Transaction Manager, the Withdrawal Transaction Manager, the Query Transaction Manager, and the Transfer Transaction Manager. For example, the business rules maintained by the Withdrawal Transaction Manager are that (1) the account must always have a balance greater or equal to zero after each withdrawal, and that (2) there is a maximum amount that can be withdrawn each day, which is given by the attribute limit in the entity class Debit Card.

21.5 DYNAMIC MODELING

The dynamic model depicts the interaction among the objects that participate in each use case. The starting point for developing the dynamic model is the use cases and the objects determined during object structuring. The sequence of interobject message communication to satisfy the needs of a use case is depicted on either a sequence diagram or a communication diagram. Usually one or the other of the diagrams suffices. In this example, both diagrams are developed for the client subsystem to allow a comparison of the two approaches.

Because the Banking System is a client/server system, the decision was made earlier to structure the system into client and service subsystems, as shown in Figure 21.8. The communication diagrams are structured for client and service subsystems.

The communication diagrams depicted in Figures 21.11 and 21.16 are for the realizations of the Validate PIN and Withdraw Funds use cases on the ATM client. Communication diagrams are also needed to realize the Transfer Funds and Query

Account use cases on the ATM client, as well as for the use cases initiated by the operator.

The Validate PIN and Withdraw Funds communication diagrams for the ATM client are state-dependent. The state-dependent parts of the interactions are defined by the ATM Control object, which executes the ATM statechart. The state-dependent dynamic analysis approach is used to determine how the objects interact with each other. Statecharts are shown for the two use cases in Figures 21.13 and 21.18, respectively. The dynamic analysis for these two client-side use cases is described in Sections 21.5.1 and 21.5.3, respectively.

The Banking Service processes transactions from multiple ATMs in the order it receives them. The processing of each transaction is self-contained; thus, the banking service part of the use cases is not state-dependent. Consequently, a stateless dynamic analysis is needed for these use cases. The communication diagrams for the server side Validate PIN and Withdraw Funds use cases are given in Figures 21.14 and 21.19. The dynamic analysis for these two server-side use cases is given in Sections 21.5.2 and 21.5.4, respectively.

Consider how the objects interact with each other. A detailed example is given for the Validate PIN and Withdraw Funds use cases. On the client side, both communication diagram and sequence diagrams are shown. The same message sequence numbering and message sequence description applies to both the sequence diagram and the communication diagram.

21.5.1 Message Sequence Description for Client-Side Validate PIN Interaction Diagram

The client-side Validate PIN interaction diagram starts with the customer inserting the ATM card into the card reader. The message sequence number starts at 1, which is the first external event initiated by the actor. Subsequent numbering in sequence, representing the messages arriving at software objects in the system, is 1.1, 1.2, 1.3 and ends with 1.4, the system's response displayed to the actor. The next input from the actor is the external event numbered 2, followed by the internal events 2.1, 2.2, and so on. The following message sequence description corresponds to the communication diagram shown in Figure 21.11 and the sequence diagram in Figure 21.12.

Because the Validate PIN interaction diagram is state-dependent, it is also necessary to consider the ATM statechart, which is executed by the ATM Control object. In particular, the interaction between the statechart (shown in Figure 21.13) and ATM Control (depicted on the communication diagram) needs to be considered. The following message sequence description also addresses the states and transitions on the statechart that correspond to the events on the communication diagram in Figure 21.11 and the events on the sequence diagram in Figure 21.12. The message sequence description is as follows:

1: The ATM Customer actor inserts the ATM card into the Card Reader. The Card Reader Interface object reads the card input.

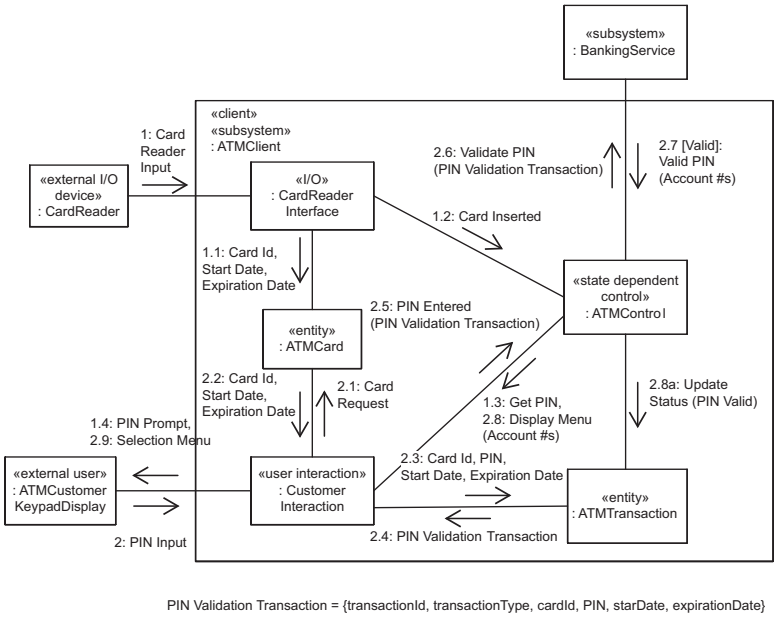


Figure 21.11. Communication diagram: ATM client Validate PIN use case

1.1: The Card Reader Interface object sends the card input data, containing Card Id, Start Date, Expiration Date to the entity object ATM Card.

1.2: Card Reader Interface sends the Card Inserted message to ATM Control. The equivalent Card Inserted event causes the ATM Control statechart to transition from Idle state (the initial state) to Waiting for PIN state. The output event associated with this transition is Get PIN.

1.3: ATM Control sends the Get PIN message to Customer Interaction.

1.4: Customer Interaction displays the PIN Prompt to the ATM Customer actor.

2: ATM Customer inputs the PIN number to the Customer Interaction object.

2.1: Customer Interaction requests card data from ATM Card.

2.2: ATM Card provides the card data to the Customer Interaction.

2.3: Customer Interaction sends Card Id, PIN, Start Date, Expiration Date, to the ATM Transaction entity object.

2.4: ATM Transaction entity object sends the PIN Validation Transaction to Customer Interaction.

2.5: Customer Interaction sends the PIN Entered (PIN Validation Transaction) message to ATM Control. The PIN Entered event causes the ATM Control statechart to transition from Waiting for PIN state to Validating PIN state. The output event associated with this transition is Validate PIN.

2.6: ATM control sends a Validate PIN (PIN Validation Transaction) request to the Banking Service.

2.7: Banking Service validates the PIN and sends a Valid PIN response to ATM Control. As a result of this event, ATM Control transitions to Waiting for

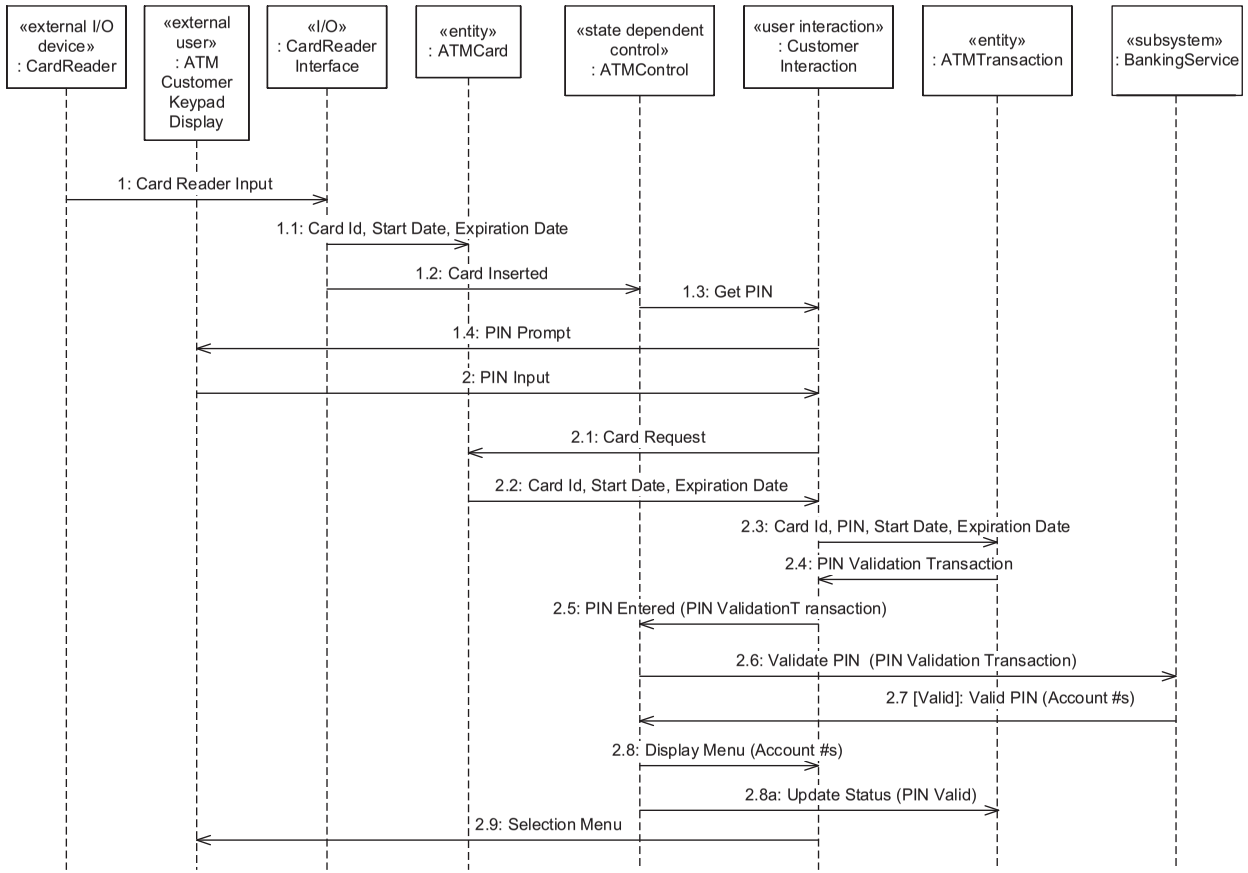


Figure 21.12. Sequence diagram: ATM client Validate PIN use case

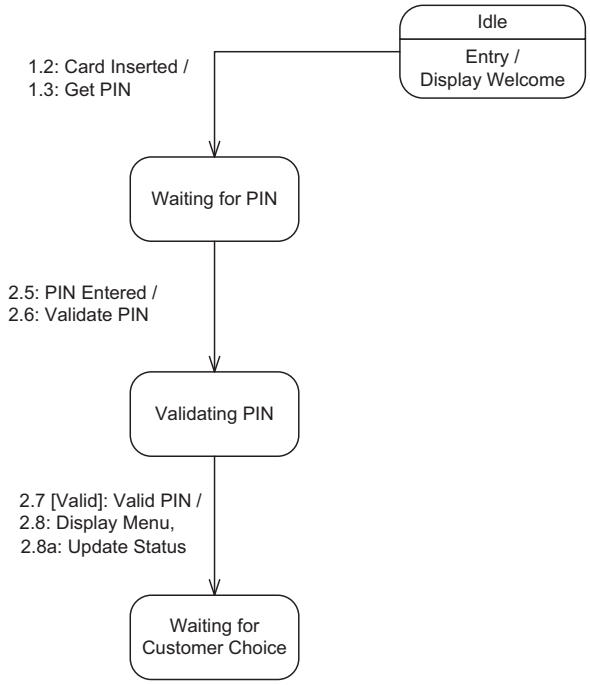


Figure 21.13. Statechart for ATM Control: Validate PIN use case

Customer Choice state. The output events for this transition are Display Menu and Update Status, which correspond to the output messages sent by ATM Control.

2.8: ATM Control sends the Display Menu message to Customer Interaction.

2.8a: ATM Control sends an Update Status message to the ATM Transaction.

2.9: Customer Interaction displays a menu showing the Withdraw, Query, and Transfer options to the ATM Customer.

The dynamic modeling of the alternative scenarios, corresponding to the alternative sequences through the Validate PIN use case, is described in Chapter 11. The alternative scenarios are depicted on interaction diagrams and statecharts.

21.5.2 Message Sequence Description for Server-Side Validate PIN Interaction Diagram

Consider the interaction diagram for the server side Validate PIN inclusion use case. To validate the PIN at the server, the Debit card entity object, which contains all the information pertinent to all debit cards that belong to the bank, needs to be accessed. If PIN validation is successful, the Card Account entity object needs to be accessed to retrieve the account numbers of the accounts that can be accessed by this debit card.

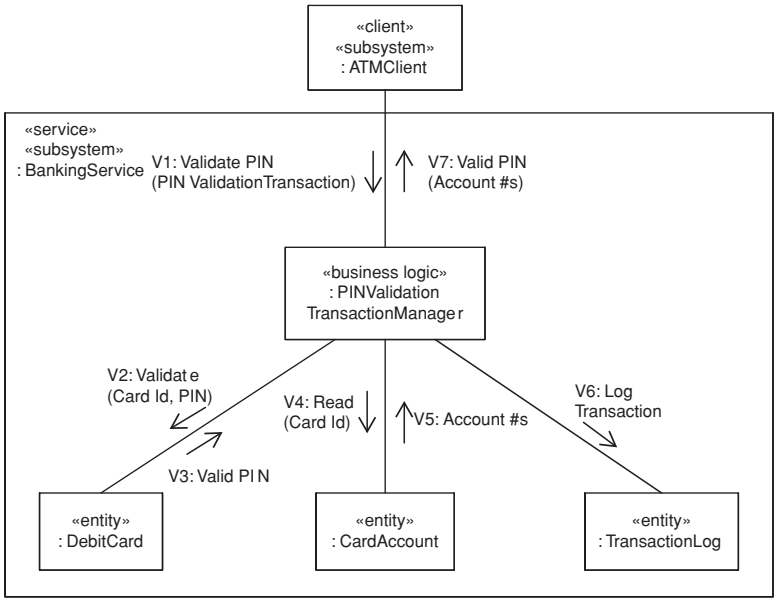


Figure 21.14. Communication diagram: Banking Service Validate PIN use case

In addition, each transaction has a **business logic object** that encapsulates the business application logic to manage the execution of the transaction. The business logic object receives the transaction request from the `ATM Control` object at the client and then interacts with the entity objects to determine what response to return to `ATM Control`. For example, the business logic object for the PIN Validation transaction is the `PIN Validation Transaction Manager`.

The following message sequence description for the server side `Validate PIN` interaction diagram corresponds to the communication diagram shown in Figure 21.14 and the sequence diagram shown in Figure 21.15.

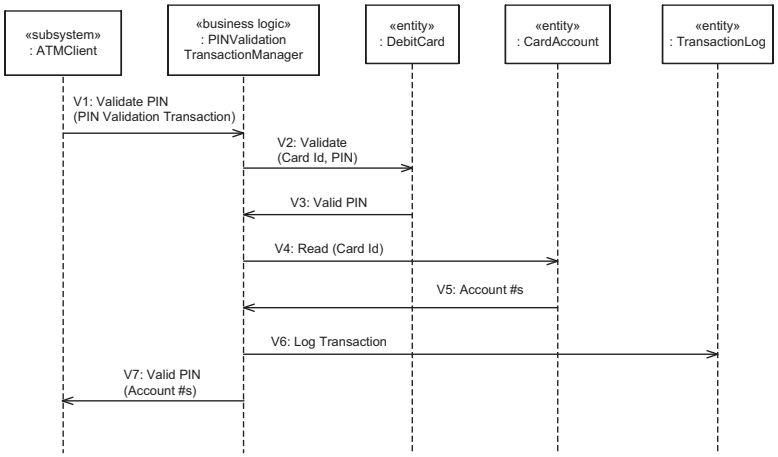


Figure 21.15. Sequence diagram: Banking Service Validate PIN use case

V1: ATM Client sends the incoming Validate PIN request to the PIN Validation Transaction Manager. The PIN Validation Transaction Manager contains the business logic to determine whether the customer-entered PIN matches the PIN stored in the Banking Service database.

V2: PIN Validation Transaction Manager sends a Validate (Card Id, PIN) message to the Debit Card entity object, requesting it to validate this customer's debit card, given the card Id and customer-entered PIN.

V3: Debit Card checks that customer-entered PIN matches the Debit Card record PIN, that card Status is okay (not reported missing or stolen), and that Expiration Date has not passed. If card passes all checks, Debit Card sends PIN Validation Transaction Manager a Valid PIN response.

V4: If validation is positive, PIN Validation Transaction Manager sends a message to the Card Account entity object requesting it to return the account numbers that may be accessed for this card Id.

V5: Card Account responds with the valid account numbers.

V6: PIN Validation Transaction Manager logs the transaction with the Transaction Log.

V7: PIN Validation Transaction Manager sends a Valid PIN response to the ATM Client. If the PIN validation checks are satisfactory, the account numbers are also sent.

21.5.3 Message Sequence Description for Client-Side Withdraw Funds Interaction Diagram

The message sequence description for the client-side Withdraw Funds interaction diagram addresses the messages on the communication diagram (Figure 21.16) and the sequence diagram (Figure 21.17). It also describes the relevant states and transitions on the ATM statechart (Figure 21.18). The message numbering is a continuation of that described for the client-side Validate PIN interaction diagram in Section 21.5.1.

3: ATM Customer actor inputs Withdrawal selection to Customer Interaction, together with the account number for checking or savings account and withdrawal amount.

3.1: Customer Interaction sends the customer selection to ATM Transaction.

3.2: ATM Transaction responds to Customer Interaction with the Withdrawal Transaction details. Withdrawal Transaction contains transaction Id, transaction Type, card Id, PIN, account number, and amount.

3.3: Customer Interaction sends the Withdrawal Selected (Withdrawal Transaction) request to ATM Control. ATM Control transitions to Processing Withdrawal state. Two output events are associated with this transition, Request Withdrawal and Display Wait.

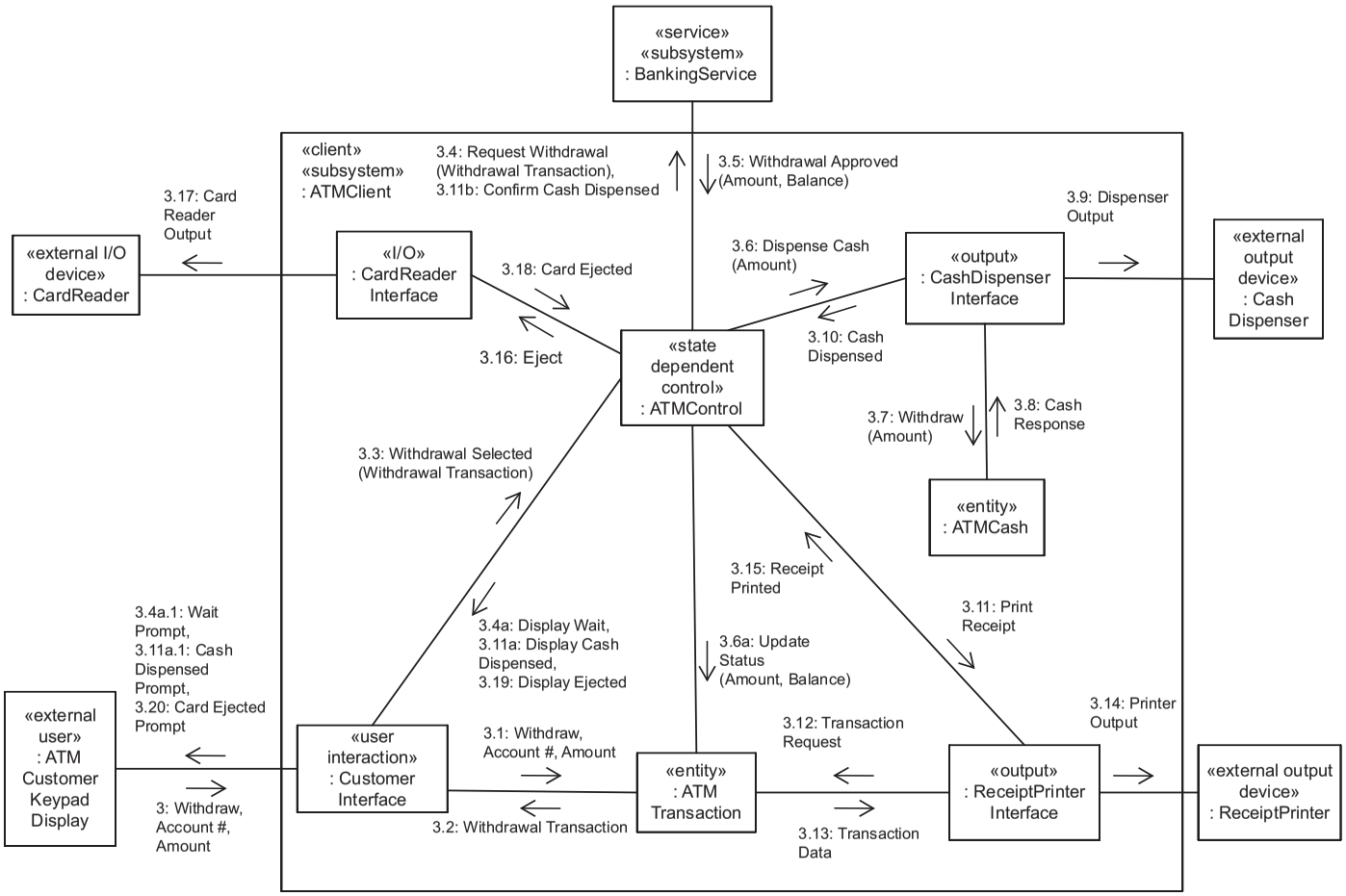


Figure 21.16. Communication diagram: ATM client Withdraw Funds use case

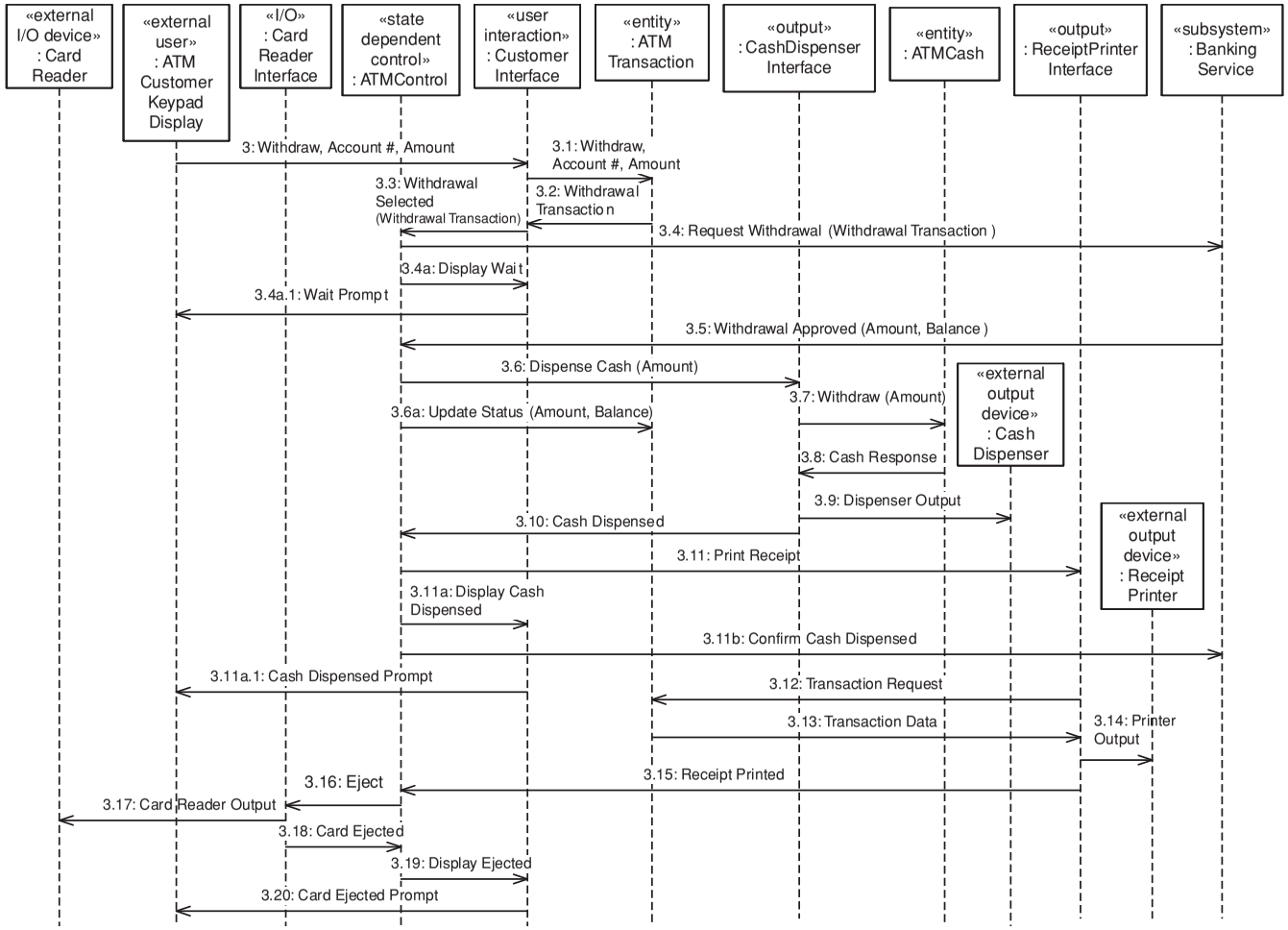


Figure 21.17. Sequence diagram: ATM client Withdraw Funds use case

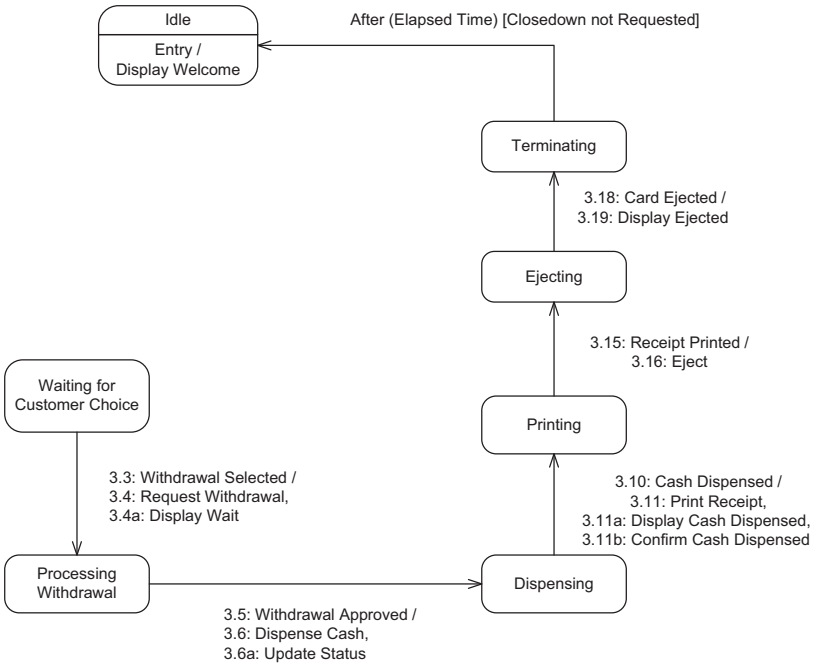


Figure 21.18. Statechart for ATM Control: Withdraw Funds use case

3.4: ATM Control sends a Request Withdrawal transaction containing the Withdrawal Transaction to the Banking Service.

3.4a: ATM Control sends a Display Wait message to Customer Interaction.

3.4a.1: Customer Interaction displays the Wait Prompt to the ATM Customer.

3.5: Banking Service sends a Withdrawal Approved (Amount, Balance) response to ATM Control. This event causes ATM Control to transition to Dispensing state. The output events are Dispense Cash and Update Status.

3.6: ATM Control sends a Dispense Cash (Amount) message to Cash Dispenser Interface.

3.6a: ATM Control sends an Update Status (Amount, Balance) message to ATM Transaction.

3.7: Cash Dispenser Interface sends the Withdraw (Amount) to ATM Cash.

3.8: ATM Cash sends a positive Cash Response to the Cash Dispenser Interface, identifying the number of bills of each denomination to be dispensed.

3.9: Cash Dispenser Interface sends the Dispenser Output command to the Cash Dispenser external output device to dispense cash to the customer.

3.10: Cash Dispenser Interface sends the Cash Dispensed message to ATM Control. The equivalent Cash Dispensed event causes ATM Control to transition to Printing state. The three output events associated with this transition are Print Receipt, Display Cash Dispensed, and Confirm Cash Dispensed.

3.11: ATM Control sends Print Receipt message to Receipt Printer Interface.

3.11a: ATM Control sends Customer Interaction the Display Cash Dispensed message.

3.11a.1: Customer Interaction displays Cash Dispensed prompt to ATM Customer.
3.11b: ATM Control sends a Confirm Cash Dispensed message to the Banking Service.
3.12: Receipt Printer Interface requests transaction data from ATM Transaction.
3.13: ATM Transaction sends the transaction data to the Receipt Printer Interface.
3.14: Receipt Printer Interface sends the Printer Output to the Receipt Printer external output device.
3.15: Receipt Printer Interface sends the Receipt Printed message to ATM Control. As a result, ATM Control transitions to Ejecting state. The output event is Eject.
3.16: ATM Control sends the Eject message to Card Reader Interface.
3.17: Card Reader Interface sends the Card Reader Output to the Card Reader external I/O device.
3.18: Card Reader Interface sends the Card Ejected message to ATM Control. ATM Control transitions to Terminated state. The output event is Display Ejected.
3.19: ATM Control sends the Display Ejected message to the Customer Interaction.
3.20: Customer Interaction displays the Card Ejected prompt to the ATM Customer.

21.5.4 Message Sequence Description for Server-Side Withdraw Funds Interaction Diagram

The **business logic object** that participates in the server-side Withdraw Funds use case is the Withdrawal Transaction Manager, which encapsulates the logic for determining whether the customer is allowed to withdraw funds from the selected account. The other business logic objects that participate in the server use cases are the Transfer Transaction Manager, which encapsulates the logic for determining whether the customer can transfer funds from one account to another, and the Query Transaction Manager. The latter is sufficiently simple that a separate business logic object is not strictly necessary; the functionality could be handled by the read operation of the Account object. However, to be consistent with the other business logic objects, it is kept as a separate object.

A detailed analysis is given for the server-side Withdraw Funds use case. A similar approach is needed for the server-side Transfer Funds and server-side Query Account use cases. The following message sequence description corresponds to the communication diagram shown in Figure 21.19 for the server-side Withdraw Funds use case and sequence shown in Figure 21.20.

W1: ATM Client sends the Request Withdrawal request to the Withdrawal Transaction Manager, which contains the business logic for determining whether a withdrawal can be allowed. The incoming withdrawal

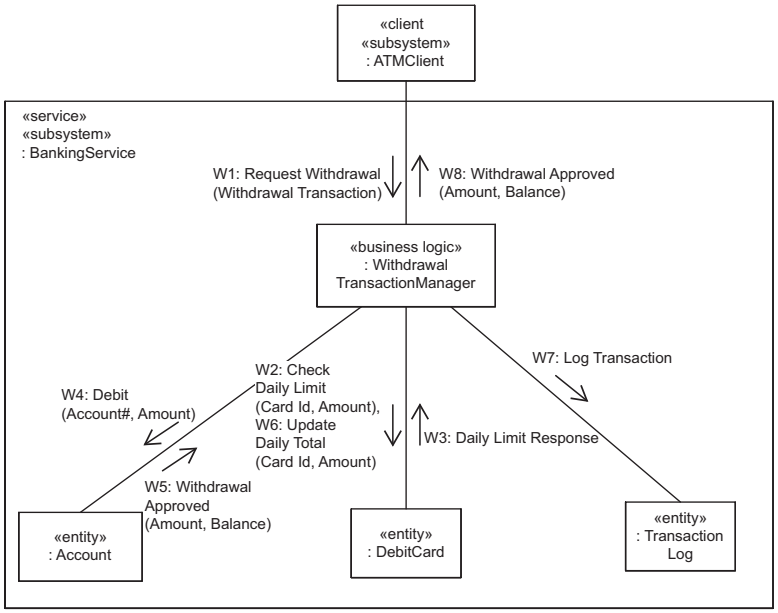


Figure 21.19. Communication diagram: Banking Service Withdraw Funds use case

transaction consists of transaction Id, transaction Type, card Id, PIN, account Number, and amount.

W2: Withdrawal Transaction Manager sends a Check Daily Limit (Card Id, Amount) message to Debit Card, with the card Id and amount requested. Debit Card checks whether the daily limit for cash withdrawal has been

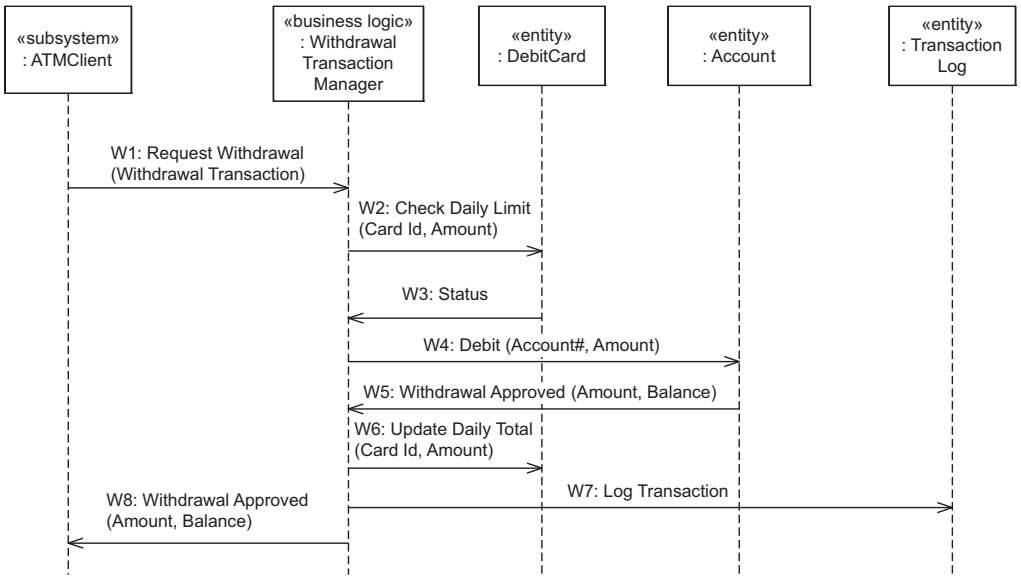


Figure 21.20. Sequence diagram: Banking Service Withdraw Funds use case

exceeded for this card Id. Debit Card determines if: Total Withdrawn Today + Amount Requested ≤ Daily Limit

W3: Debit Card responds to Withdrawal Transaction Manager with a positive or negative Daily Limit Response.

W4: If the response is positive, Withdrawal Transaction Manager sends a message to Account (which is an instance of either Checking Account or Savings Account), requesting it to debit the customer's account if there are sufficient funds in the account. Account determines whether there are sufficient funds in the account:

$$\text{Account Balance} - \text{Amount Requested} \geq 0$$

If there are sufficient funds, Account decrements the balance by the Amount Requested.

W5: Account responds to Withdrawal Transaction Manager with either Withdrawal Approved (Amount, Balance) or Withdrawal Denied.

W6: If the account was debited satisfactorily, the Withdrawal Transaction Manager sends an Update Daily Total (Card Id, Amount) to Debit Card so it increments the total withdrawn today by the amount requested.

W7: Withdrawal Transaction Manager logs the transaction with the Transaction Log.

W8: Withdrawal Transaction Manager returns Withdrawal Approved (Amount, Balance) or Withdrawal Denied to the ATM Client.

21.6 ATM STATECHART

Because there is one control object, ATM Control, a statechart needs to be defined for it. Partial statecharts are shown corresponding to the Validate PIN and Withdraw Funds use cases in Figures 21.14 and 21.18, respectively. It is necessary to develop similar statecharts for the other use cases, and to develop states and transitions for the alternative paths of the use cases, which in this application address error situations. Flat statecharts are used initially for the use cases. Integration of the statecharts for the individual use cases and design of the hierarchical ATM Control statechart are described in Chapter 10. One of the advantages of a hierarchical statechart is that it can be presented in stages, as is shown for the ATM statechart in Figures 21.21 through 21.24. The event sequence numbers shown on these figures correspond to the object interactions previously described.

Five states are shown on the top-level statechart in Figure 21.21: Closed Down (which is the initial state), Idle, and three composite states, Processing Customer Input, Processing Transaction, and Terminating Transaction. Each composite state is decomposed into its own statechart, as shown on Figures 21.22, 21.23, and 21.24, respectively.

At system initialization time, given by the event Startup, the ATM transitions from the initial Closed Down state to Idle state. The event Display Welcome is triggered on entry into Idle state. In Idle state, the ATM is waiting for a customer-initiated event.

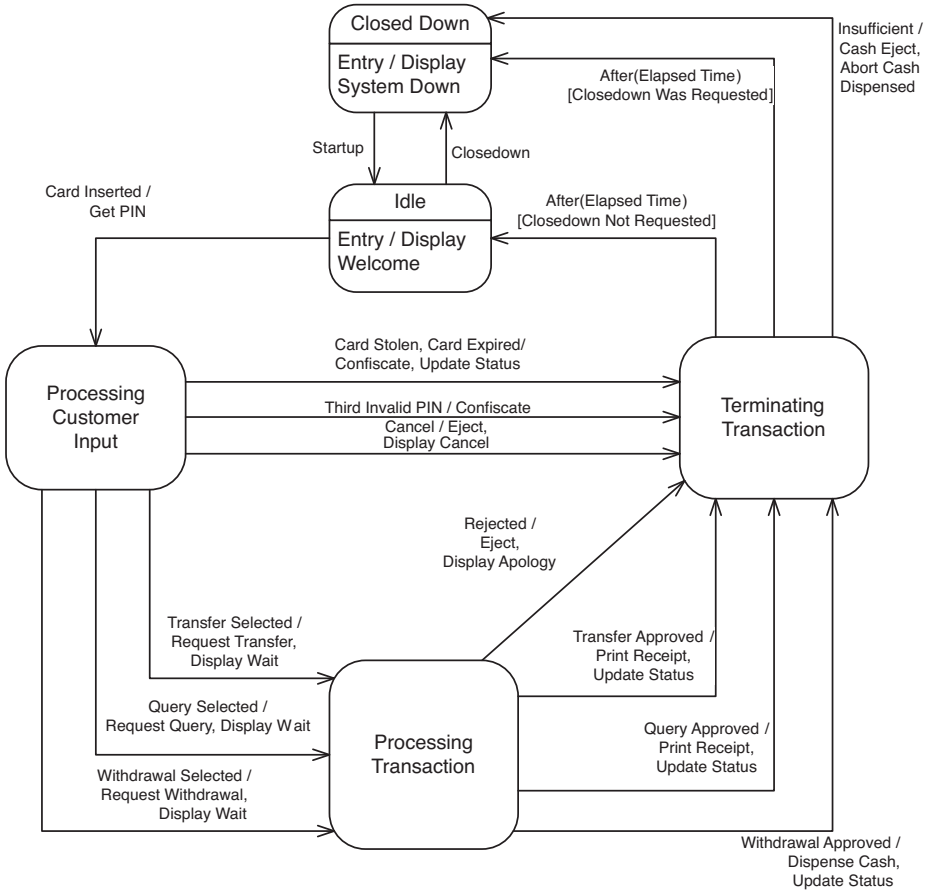


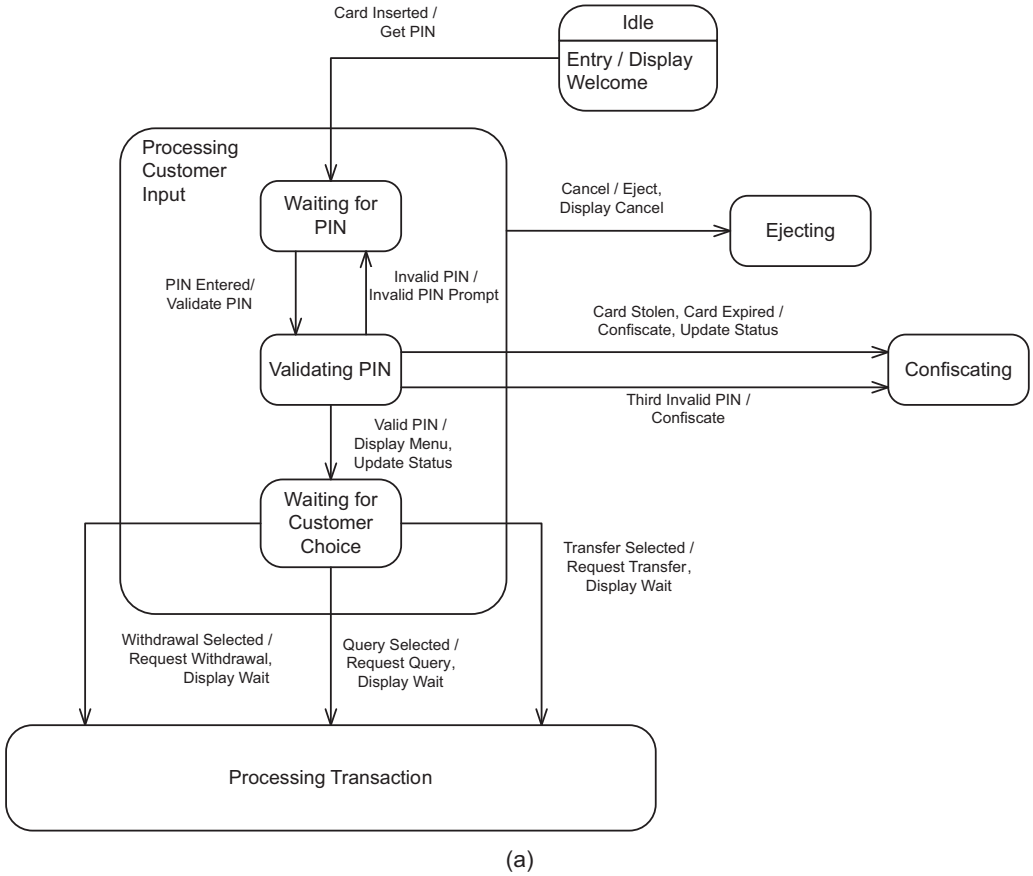
Figure 21.21. Top-level statechart for ATM Control

21.6.1 Processing Customer Input Composite State

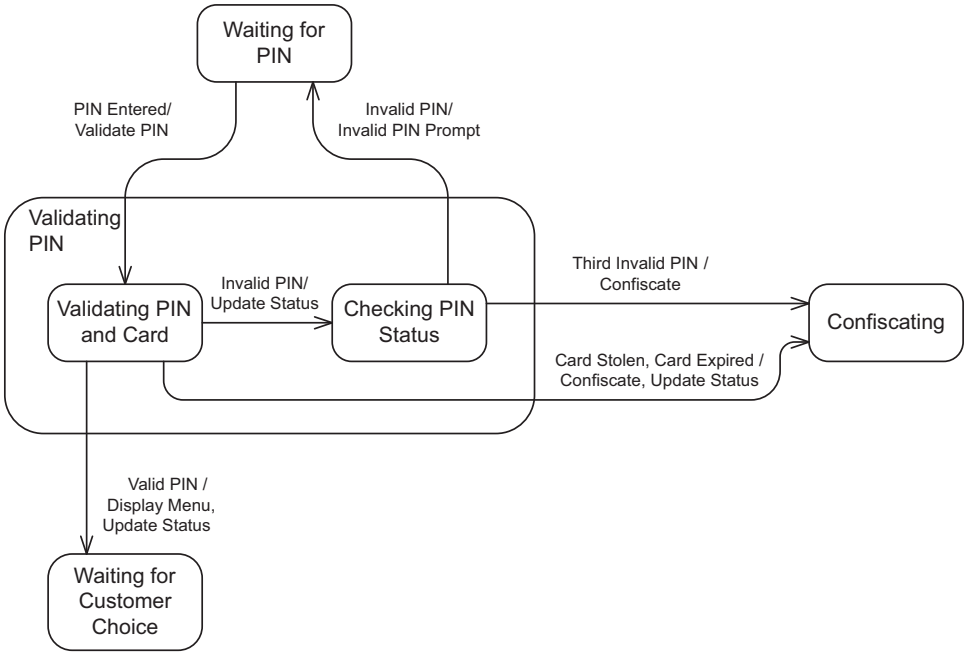
The Processing Customer Input composite state (Figure 21.22) is decomposed into three substates – Waiting for PIN, Validating PIN, and Waiting for Customer Choice:

1. **Waiting for PIN.** This substate is entered from Idle state when the customer inserts the card in the ATM, resulting in the Card Inserted event. In this state, the ATM waits for the customer to enter the PIN.
2. **Validating PIN.** This substate is entered when the customer enters the PIN. In this substate, the Banking Service validates the PIN.
3. **Waiting for Customer Choice.** This substate is entered as a result of a Valid PIN event, indicating a valid PIN was entered. In this state, the customer enters a selection: Withdraw, Transfer, or Query.

The statechart is developed by considering the different states of the ATM as the customer actor proceeds through each of the use cases, starting with the Validate PIN use case. When a customer inserts an ATM card, the event Card Inserted causes the ATM to transition to the Waiting for PIN substate of the Processing Customer Input composite state (see Figure 21.22*a*). During this time, the ATM is waiting for the customer to input the PIN. The output event, Get PIN, results in a display prompt



(a)



(b)

Figure 21.22. Statechart for ATM Control: Processing Customer Input composite state

to the customer. When the customer enters the PIN number, the PIN Entered event causes a transition to the Validating PIN substate, during which the Banking Service determines whether the customer-entered PIN matches the PIN stored by the Banking System for this particular card. There are three possible state transitions out of the Validating PIN state. If the two PIN numbers match, the Valid PIN transition is taken to the Waiting for Customer Choice state. If the PIN numbers do not match, the Invalid PIN transition is taken to re-enter the Waiting for PIN state and allow the customer to enter a different PIN number. If the customer-entered PIN is still invalid after the third attempt, the Third Invalid transition is taken to the Confiscating substate of the Terminating Transaction composite state.

The Validating PIN substate is itself a composite state consisting of two substates: Validating PIN and Card as well as Checking PIN Status (see Figure 21.22*b*). In the first substate, the card Id (read off card) and PIN (entered by customer) combination are validated by comparing them with the card Id/PIN combination stored in the Card Account entity object. In addition, the card Id is checked to ensure that the card is not lost or stolen. If the validation is successful, the ATM transitions to Waiting for Customer Choice. If the card is lost or stolen, the ATM transitions to Confiscating state. However, if the PIN is invalid, an additional check needs to be made to determine whether this is the third time that the PIN is incorrect. It is better to store the Invalid PIN count at the client rather than the server, because this is a local ATM concern. An invalid PIN count is therefore stored in ATM Transaction. This count is updated and checked after each invalid PIN response from the server – if the count is less than three, then the ATM transitions back to Waiting for PIN. If the count is Third Invalid PIN, then the ATM transitions to Confiscating state.

The customer can also press the Cancel button on the ATM machine in any of the three Processing Customer Input substates. The Cancel event transitions the ATM to the Ejecting substate of the Terminating Transaction composite state. Because the Cancel event can occur in any of the three substates of the Processing Customer Input composite state, it is more concise to show the Cancel transition leaving the composite state.

21.6.2 Processing Transaction Composite State

The Processing Transaction composite state (Figure 21.23) is also decomposed into three substates, one for each transaction: Processing Withdrawal, Processing Transfer,

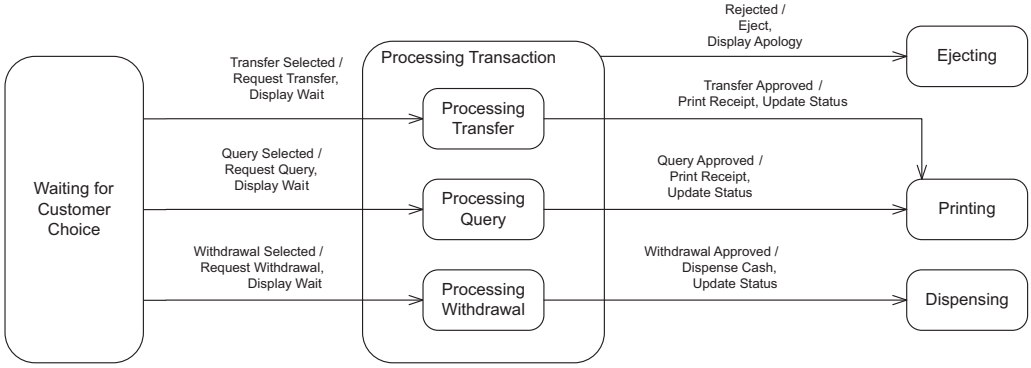


Figure 21.23. Statechart for ATM: Processing Transaction composite state

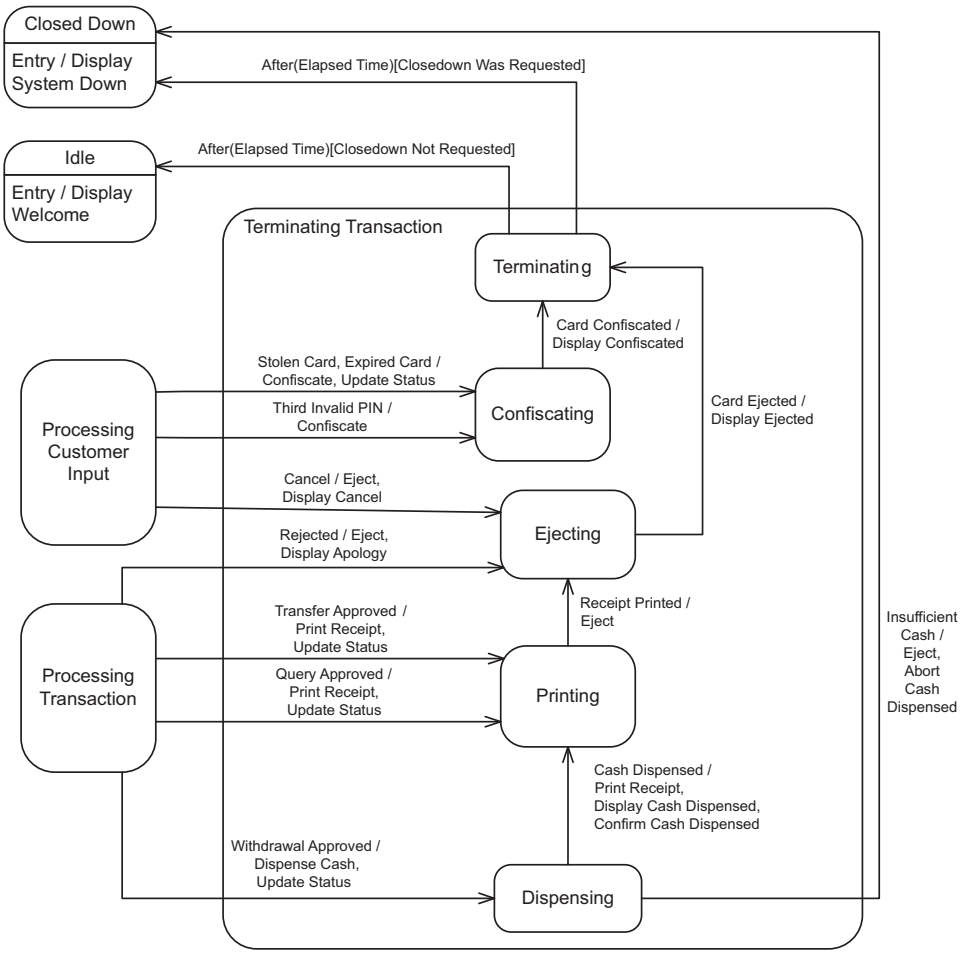


Figure 21.24. Statechart for ATM Control: Terminating Transaction composite state

and Processing Query. Depending on the customer's selection – for example, withdrawal – the appropriate substate within Processing Transaction – for example, Processing Withdrawal – is entered, during which the customer's request is processed.

From Waiting for Customer Choice state, the customer may select Withdraw, Query, or Transfer and enter the appropriate substate within the Processing Transaction composite state (see Figure 21.23) – for example, Processing Withdrawal. When a Withdrawal transaction is completed, the event Withdrawal Approved is issued if the customer has enough funds, and the Dispensing substate of the Terminating Transaction composite state is entered (Figure 21.24). Alternatively, if the customer has insufficient funds or has exceeded the daily withdrawal limit, a Rejected event is issued.

21.6.3 Terminating Transaction Composite State

The Terminating Transaction composite state (see Figure 21.24) has substates for Dispensing, Printing, Ejecting, Confiscating, and Terminating.

The actions associated with the transition to Dispensing state are to Dispense Cash and Update Status. After the Cash Dispensed event has taken place, the ATM transitions to Printing state to print the receipt. The action Print Receipt is executed at the transition. When the receipt is printed, the state Ejecting is entered and the Eject action is executed. When the card has been ejected (event Card Ejected), the Terminating state is entered.

For the Query and Transfer transactions, the sequence of states following approval of the transaction is similar, except that no cash is dispensed, as can be seen on the ATM statecharts.

21.7 DESIGN OF BANKING SYSTEM

Next, the analysis model of the Banking System is mapped to a design model. The steps in this process are as follows:

1. Integrate the communication model. Develop integrated communication diagrams.
2. Structure the Banking System into subsystems. Define the interfaces of the subsystems.
3. For each subsystem, structure the system into concurrent tasks.
4. For each subsystem, design the information hiding classes.
5. Develop the detailed software design.

21.8 INTEGRATING THE COMMUNICATION MODEL

Because the Banking System is a client/server system (Section 21.4), a decision was made earlier to structure the system into client and service subsystems, as shown in Figure 21.8. The communication diagrams are also structured for client and service subsystems.

The communication diagrams for the client-side Validate PIN and Withdraw Funds use cases are depicted in Figures 21.11 and 21.16. Communication diagrams are also needed for the client-side Transfer Funds and Query Account use cases, as well as for the use cases initiated by the operator. The integrated communication diagram for the ATM Client Subsystem (Figure 21.25) is the result of the merger of all these use case–based communication diagrams, as described in Chapter 13. To be complete, the integration must consist of communication scenarios for the main and alternative sequences through each use case.

Some objects participate in all the client-side communications, such as ATM Control, but others participate in as few as one, such as the Cash Dispenser Interface. Some of the messages depicted on the integrated communication diagram are aggregate messages, such as Customer Events and Display Prompts. The integrated diagram must also include messages from all the alternative sequences, as described in Chapter 13. Thus, the Confiscate and Card Confiscated messages originate from alternative sequences in which the customer transaction is unsuccessful. Similarly, the aggregate Display Prompts messages include messages dealing with incorrect PIN entry, insufficient cash in the customer account, and so on.

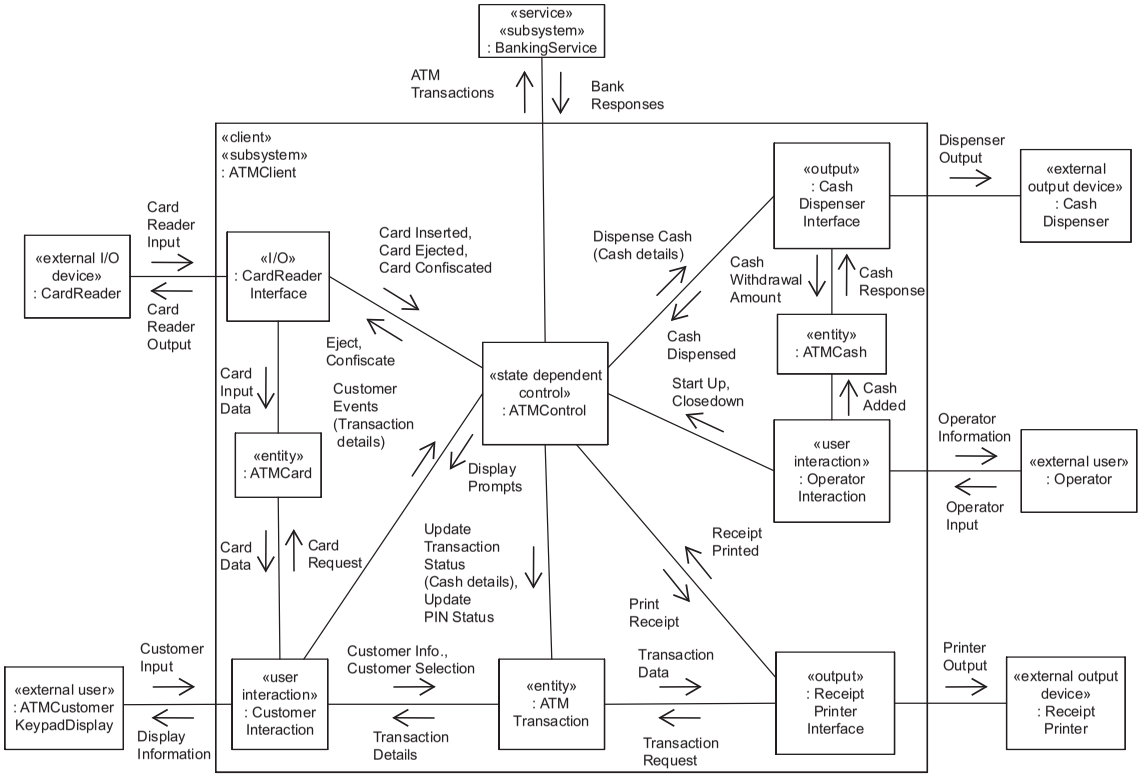


Figure 21.25. Integrated communication diagram for ATM client subsystem

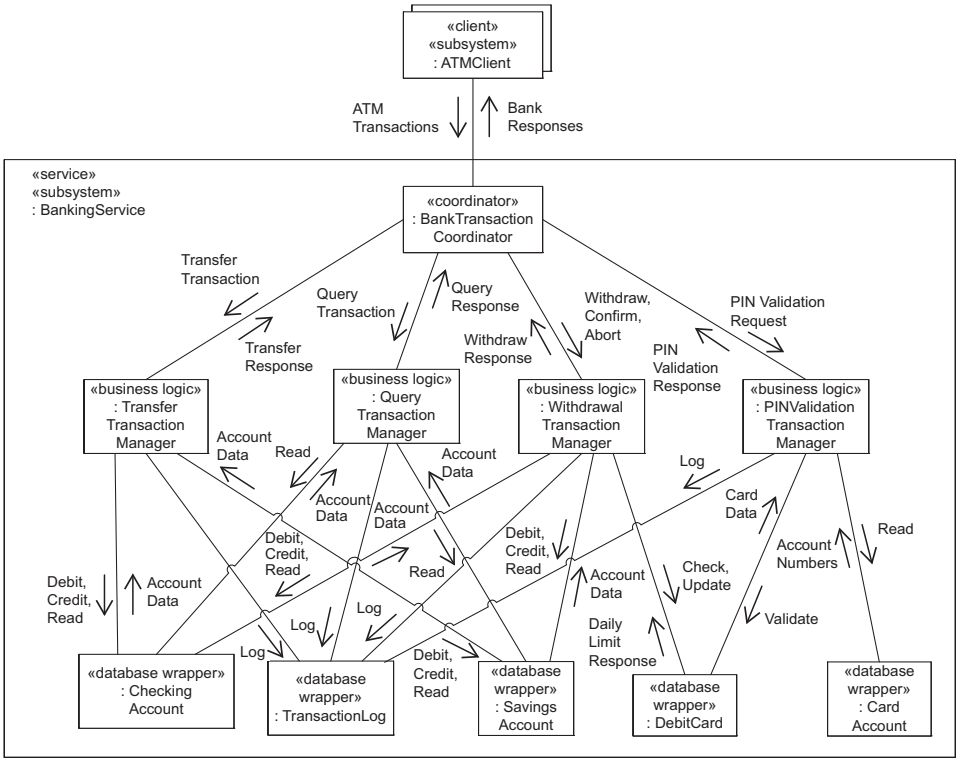


Figure 21.26. Integrated communication diagram for Banking Service subsystem

Now consider the Banking Service Subsystem. Figures 21.14 and 21.19 are the communication diagrams for the server-side Validate PIN and Withdraw Funds use cases. Additional communication diagrams are needed for the server-side Transfer Funds and Query Account use cases. The integrated communication diagram for the Banking Service Subsystem is shown in Figure 21.26. For each transaction, there is a transaction manager object that encapsulates the business logic for the transaction. These are the PIN Validation Transaction Manager, Withdrawal Transaction Manager, Query Transaction Manager, and Transfer Transaction Manager objects. In addition, it is decided at design time that there is a need for a coordinator object, the Bank Transaction Coordinator, which receives client requests and delegates them to the appropriate transaction manager, as described in Chapter 15.

21.9 STRUCTURING THE SYSTEM INTO SUBSYSTEMS

In the case of the Banking System, the step of structuring the system into subsystems is straightforward. The Banking System is a classic client/server architecture that is based around the multiple client/single service architectural pattern. There are two subsystems, the multiple instances of the ATM Client Subsystem and the Banking Service Subsystem, as initially depicted in Figure 21.8. The two subsystems might also be depicted on a high-level communication diagram, as shown in Figure 21.27.

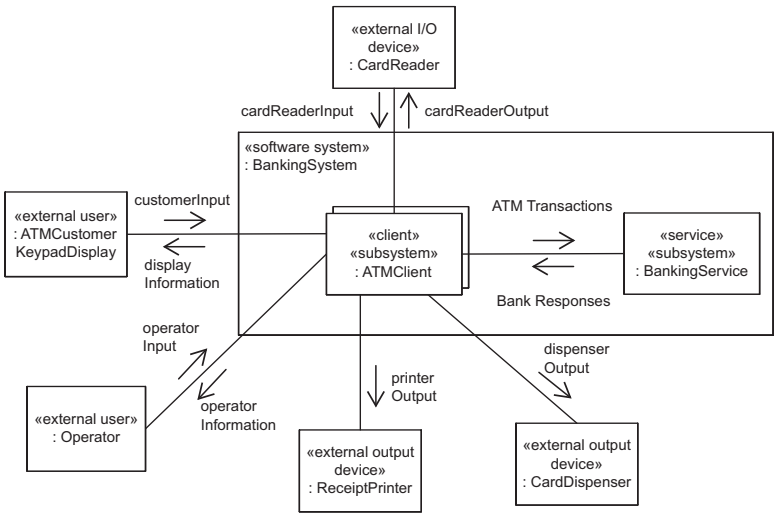


Figure 21.27. Subsystem design: high-level communication diagram for Banking System

Figure 21.27 is an analysis-level communication diagram showing the two subsystems and simple messages passed between them. The ATM Client Subsystem sends ATM Transactions to the Banking Service Subsystem, which responds with Bank Responses. ATM Transactions is an aggregate message consisting of the PIN Validation, Withdraw, Query, Transfer, Confirm, and Abort messages. The Bank Responses are responses to these messages.

The next step is to consider the distributed nature of the application and define the distributed message interfaces. Because this is a client/server subsystem, there are multiple instances of the client subsystem and one instance of the service subsystem. Each subsystem instance executes on its own node. In the design model, each of these subsystems is a concurrent subsystem, consisting of at least one task. The message interface is **synchronous message communication with reply**. Each ATM client sends a message to the Banking Service and then waits for a response. Because the Banking Service can receive messages from several ATM clients, a message queue can build up at the Banking Service, which processes incoming messages on a FIFO basis. The design model communication diagram is depicted in Figure 21.28.

The next step is to structure each subsystem into concurrent tasks. In the following sections, the design of the ATM Client Subsystem and then the design of the Banking Service Subsystem are considered.

21.10 DESIGN OF ATM CLIENT SUBSYSTEM

To determine the tasks in a system, it is necessary to understand how the objects in the application interact with each other. This is best depicted on the analysis model communication diagram, which shows the sequence of messages passed between objects in support of a given use case. For the ATM Client Subsystem, consider the communication diagrams for the Client Validate PIN and Client Withdraw Funds use cases in addition to the integrated communication diagram for this subsystem. The

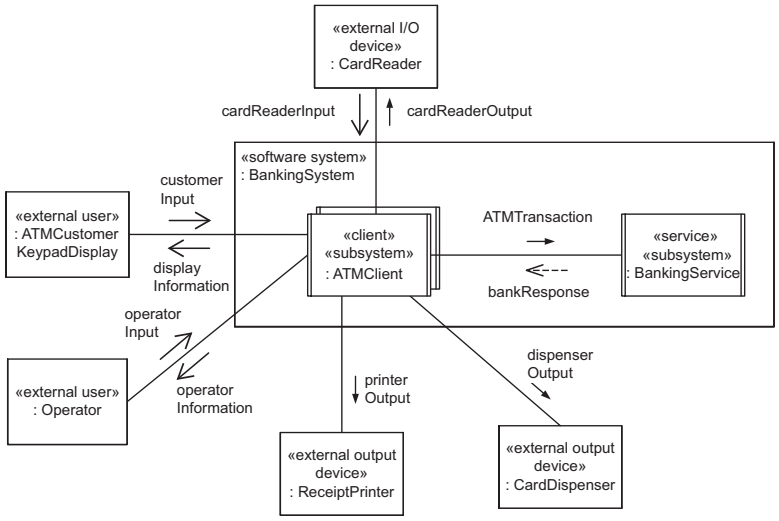


Figure 21.28. Subsystem interfaces: high-level concurrent communication diagram for Banking System

task design described in this section leads to the concurrent communication diagram shown in Figure 21.29.

21.10.1 Design the ATM Subsystem Concurrent Task Architecture

Consider the communication diagram supporting the Validate PIN use case (see Figure 21.11). The first object to participate in the communication is the Card Reader Interface object, which is an I/O object that interfaces to the real-world card reader. The characteristics of the Card Reader external I/O device are that it is an event driven I/O device that generates an interrupt when some input is available. The Card Reader Interface object is structured as an event driven I/O task, as shown in Figure 21.29. Initially, the task is dormant. It is activated by an interrupt, reads the card reader input, and converts it into an internal format. It then writes the contents of the card to the ATM Card entity object. ATM Card is a passive object and thus does not need a separate thread of control. It is further categorized as a data abstraction object.

The Card Reader Interface task then sends a Card Inserted message to ATM Control, which is a state-dependent control object that executes the ATM Control statechart. ATM Control is structured as a demand driven state-dependent control task because it needs to have a separate thread of control to allow it to react to incoming messages from a variety of sources. Initially, it is idle until it is activated on demand by the arrival of a control request message. On receiving the Card Inserted message, ATM Control executes the statechart and transitions to Waiting for PIN substate (see Figures 21.21 and 21.22). The action associated with the state transition is to send a Get PIN message to Customer Interaction, which is a user interaction object that interacts with the user, providing outputs to the display and receiving inputs from the keypad. Customer Interaction is structured as an event driven user interaction task with its own separate thread of control. It prompts the customer for the PIN,

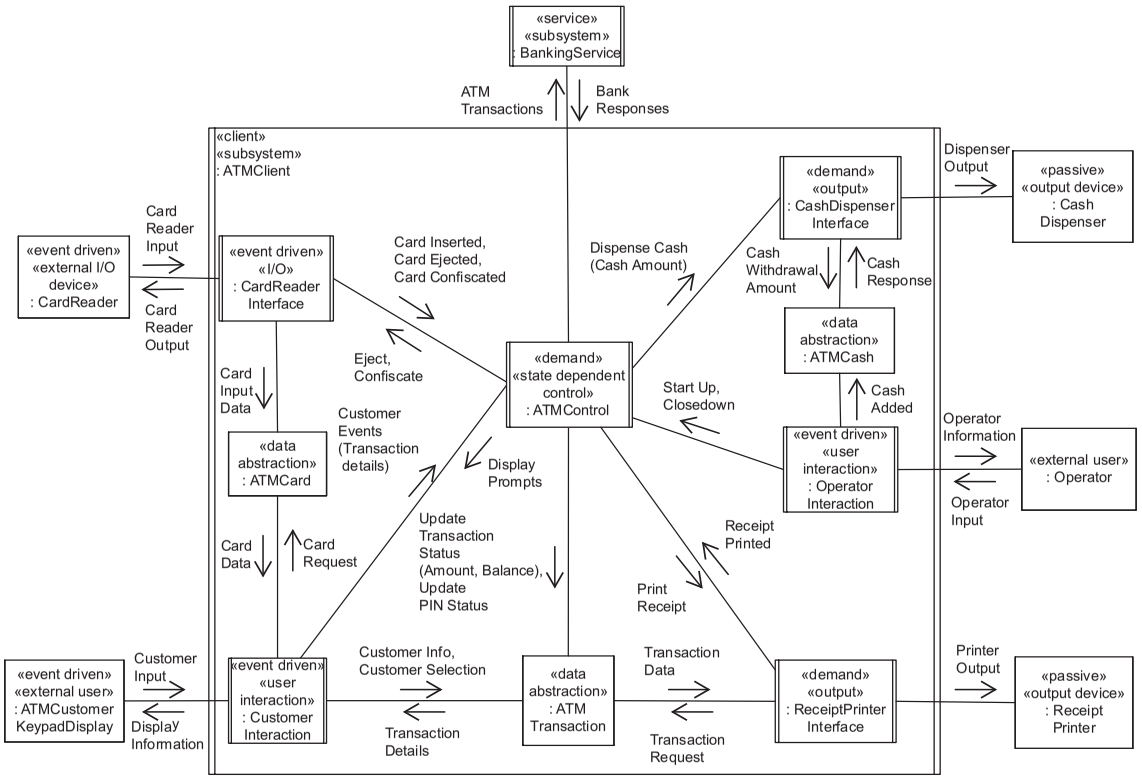


Figure 21.29. Task architecture: initial concurrent communication diagram for ATM client subsystem

receives the PIN, reads the card information from ATM Card, and then writes the card and PIN information to ATM Transaction, which is also a passive data abstraction object. Because the ATM Card and ATM Transaction data abstraction objects are each accessed by more than one task, they are both placed outside any task.

Next, consider the communication diagram supporting the Withdraw Funds use case, which has many of the same objects as the Validate PIN communication diagram. The additional objects are Receipt Printer Interface, Cash Dispenser Interface, and ATM Cash.

The external Cash Dispenser is a passive output device, so it does not need an event driven output task. Instead, the Cash Dispenser Interface object is structured as a demand driven output task, which is activated on demand by message arrival from ATM Control. Similarly, the Receipt Printer Interface object is structured as a demand driven output task, which is activated by message arrival from the ATM Control task.

The Operator Interaction user interaction object (see Figure 21.24), which participates in the three operator-initiated use cases, is also mapped to an event driven user interaction task (see Figure 21.29). The ATM Cash entity object is a passive data abstraction object and thus does not need a separate thread of control, which is accessed by both the Cash Dispenser Interface and Operator Interaction tasks.

To summarize, there is one event driven I/O task, Card Reader Interface, one demand driven state-dependent control task, ATM Control, two demand driven output tasks, Cash Dispenser Interface and Receipt Printer Interface, and two event driven user interaction tasks, Customer Interaction and Operator Interaction. There are three passive entity objects, ATM Card, ATM Transaction, and ATM Cash, which are all categorized further as data abstraction objects.

21.10.2 Define the ATM Subsystem Task Interfaces

To determine the task interfaces, it is necessary to analyze the way the objects (active or passive) interact with each other. First, consider the interaction of the tasks just determined with the passive data abstraction objects. In each case, the task calls an operation provided by the passive object. This has to be a synchronous call, because the operation executes in the thread of control of the task. Similarly, all other operations of the data abstraction objects are invoked as synchronous calls. Because each of these passive objects is invoked by more than one task, it is necessary for the operations to synchronize the access to the data. The operations provided by these passive objects are described in the next section.

Next consider the message interaction between the tasks. Consider the interface between the Card Reader Interface and ATM Control tasks. It is desirable for Card Reader Interface task to be able to send a message to ATM Control and not have to wait for it to be accepted. For this to be the case, an asynchronous message interface is needed, as shown in Figure 21.30. This means that there is also a message interface in the opposite direction because ATM Control sends Eject and Confiscate messages to the Card Reader Interface task. This is designed as a synchronous message interface without reply because, after sending a message to ATM Control, the Card Reader Interface waits for an Eject or Confiscate return message. This means that ATM Control can send a synchronous message and not have to wait for Card Reader Interface to accept the message. The latter task's responses are asynchronous, providing the greatest flexibility in the interface between the Card Reader Interface and ATM Control tasks.

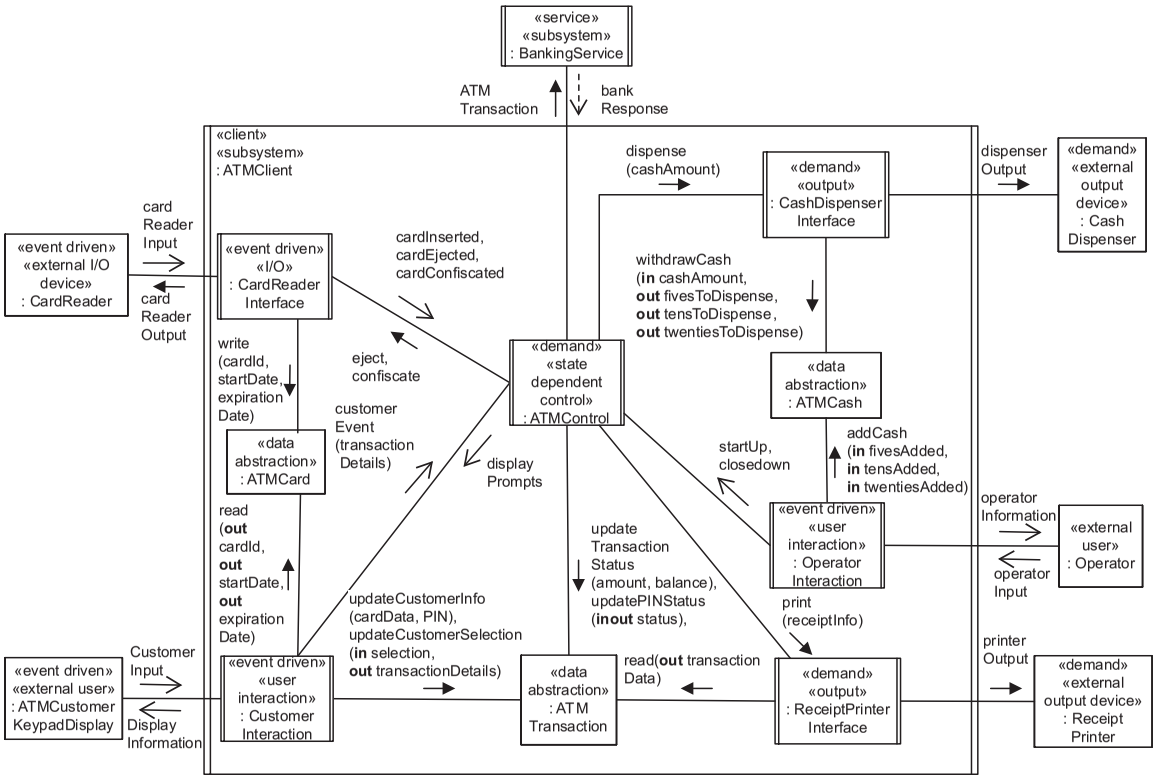


Figure 21.30. Task architecture: revised concurrent communication diagram for ATM client subsystem

Consider the interface between Customer Interaction and ATM Control. Should it be asynchronous or synchronous? First, consider a synchronous with response scenario. Customer Interaction sends a Withdrawal request to ATM Control, which then sends the transaction to the Banking Service. After receiving the Server's response, ATM Control sends a display prompt to Customer Interaction. In the meantime, it is not possible for Customer Interaction to have any interaction with the customer, because it is suspended, waiting for the response from the ATM Control. This is undesirable from the customer's viewpoint. Consider, instead, an asynchronous interface, as shown in Figure 21.30. With this approach, Customer Interaction sends the Withdrawal request to ATM Control and does not wait for a response. In this case, Customer Interaction can respond to customer inputs such as a Cancel request before a response is received from the server. Customer Interaction receives responses from ATM Control as a separate asynchronous message interface. Customer Interaction is designed to be capable of receiving inputs from either the customer or ATM Control. It processes whichever input comes first.

The Operator Interaction task's interface is also asynchronous. The operator actor's requests are independent of the customer's requests, so messages from the customer and the operator could arrive in any order at ATM Control. To allow for this, ATM Control receives all incoming messages on a message queue and processes them on a FIFO basis.

The two output tasks, Cash Dispenser Interface and Receipt Printer Interface, are activated by messages arriving from ATM Control on demand. In each case, the output task is idle prior to the arrival of the message, so a synchronous interface is acceptable because it will not hold up ATM Control. In Figure 21.30, the concurrent communication diagram is updated to show the task interfaces.

21.10.3 Design the ATM Client Information Hiding Classes

The objects and classes for the Banking System are initially determined in the analysis model. Further categorization of passive classes is possible during design; for example, entity classes are categorized further as data abstraction classes or database wrapper classes. During class design, the class interfaces are designed, as described in Chapter 14. To determine the class interfaces, it is necessary to consider how the objects on the communication diagrams interact with each other.

First, consider the design of the entity classes in the ATM Client Subsystem. Because there is no database in the ATM Client Subsystem, all the entity classes encapsulate their own data and are therefore categorized further as data abstraction classes. The ATM Client Subsystem has three data abstraction classes: ATM Card, ATM Transaction, and ATM Cash. The attributes of data abstraction classes are determined during the conceptual static modeling of the entity classes, as described in Section 21.3. The operations of these classes are determined by analyzing the way they are used on the communication diagrams.

The designs of the ATM Cash and ATM Card classes are described in Chapter 14. For the ATM Transaction class, the attributes are also determined from the static model, but its operations are determined from the way it is accessed by other objects, as given on the communication diagrams. The operations are update Customer Information, update Customer Selection, update PIN Status, update Transaction Status, and

read. The first two operations are invoked by the Customer Interaction task. The next two operations are invoked by the ATM Control task. The read operation is invoked by the Receipt Printer Interface task prior to printing the receipt.

There is one state-machine class, namely, ATM State Machine, which is internal to the ATM Control task and encapsulates the ATM statechart, which is implemented as a state transition table. The operations are process Event and current State, which are standard operations for a state-machine class.

The design of the classes is shown in more detail in Figure 21.31, which shows the attributes and operations of the classes.

21.11 DESIGN OF BANKING SERVICE SUBSYSTEM

Because the bank server holds the centralized database for the Banking System, we start the design of the Banking Service Subsystem by considering some important design decisions concerning the static model. The conceptual static model of the entity classes (see Figures 21.4–21.7) contains several entity classes that actually reside at the bank server. A design decision is made that the entity classes at the server, which were originally depicted in the static model of the problem domain (see Figure 21.4), are to be stored as relational tables in a relational database. Thus, during design we determine that the entity classes at the server do not actually encapsulate any data but rather encapsulate the interface to the relational database and are actually database wrapper classes. The design of the database wrapper classes and the mapping of the entity class model to the relational database are described later in this section.

21.11.1 Design the Banking Service Subsystem Concurrent Task Architecture

Now consider the Banking Service Subsystem design. A decision is made to use a sequential service. As long as the throughput of the server is fast enough, this is not a problem. In a sequential service, the service is designed as one task; thus, it is designed as one program with one thread of control. Each transaction is received on a FIFO message queue and is processed to completion before the next transaction is started.

The Banking Service Subsystem is designed as one sequential service task, which is activated on demand. Inside the task are the coordinator object (the Bank Transaction Coordinator), the business logic objects (PIN Validation Transaction Manager, Withdrawal Transaction Manager, Query Transaction Manager, and Transfer Transaction Manager), and the entity classes, now categorized further as database wrapper classes. The initial task design for the service subsystem, consisting of one task, is shown in Figure 21.32.

The Bank Transaction Coordinator task receives the incoming transaction messages and replies with the bank responses. It delegates the transaction processing to the transaction managers, which in turn access the database wrapper objects. All communication internal to the Banking Service Subsystem is synchronous, corresponding to operation calls, as described next.

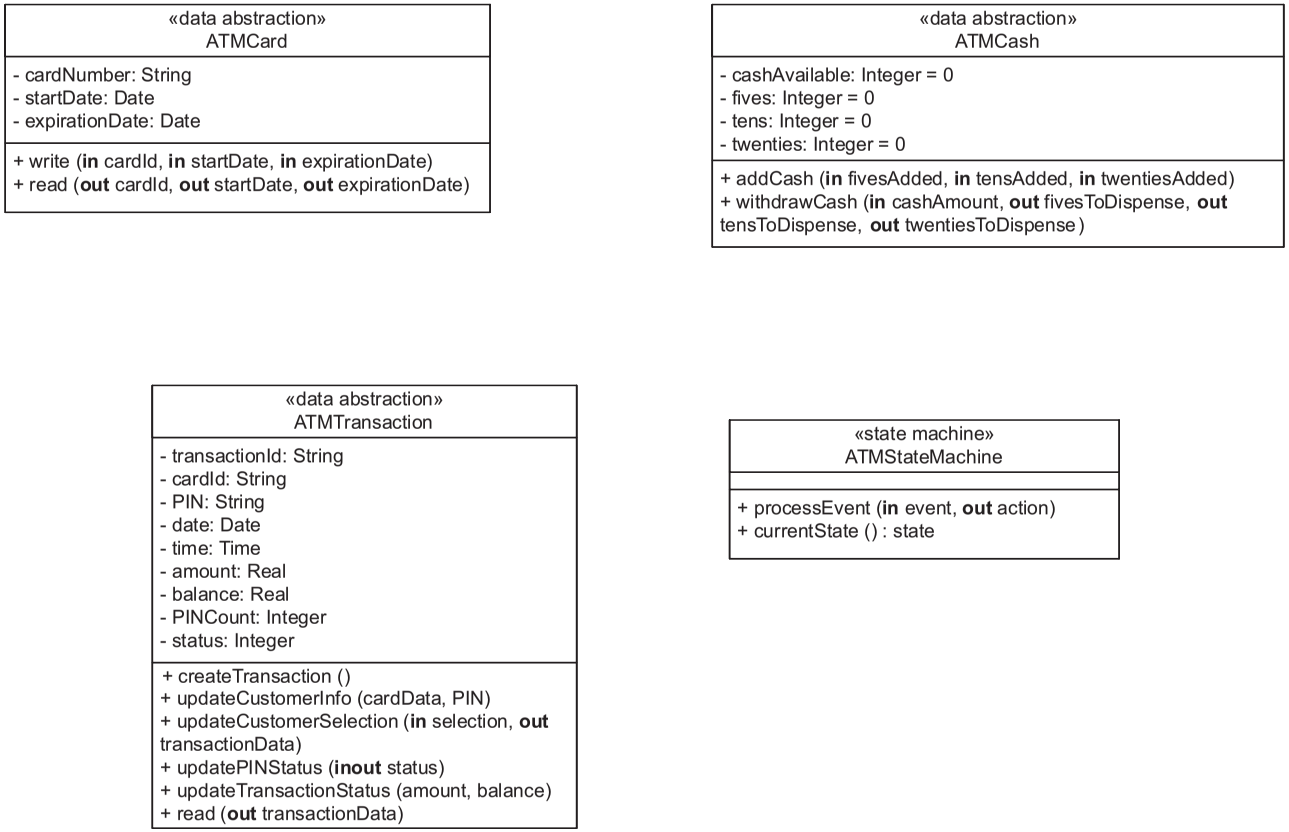


Figure 21.31. ATM client information hiding classes

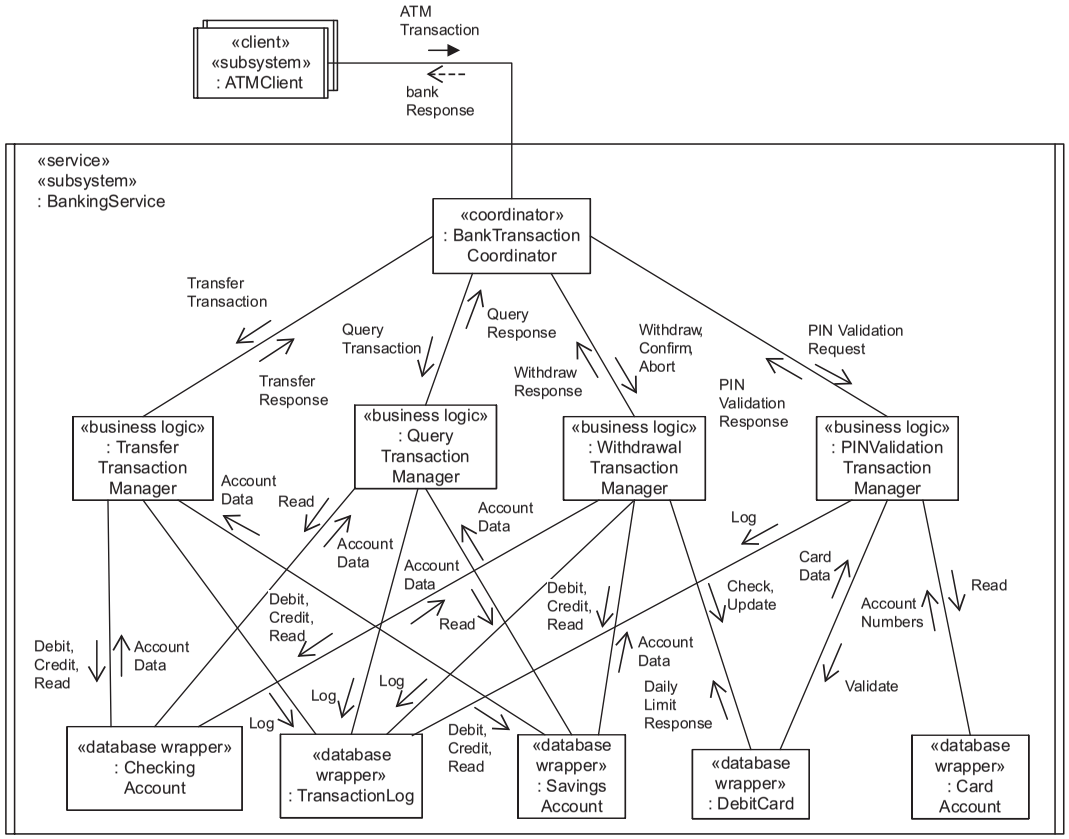


Figure 21.32. Initial concurrent communication diagram for Banking Service subsystem

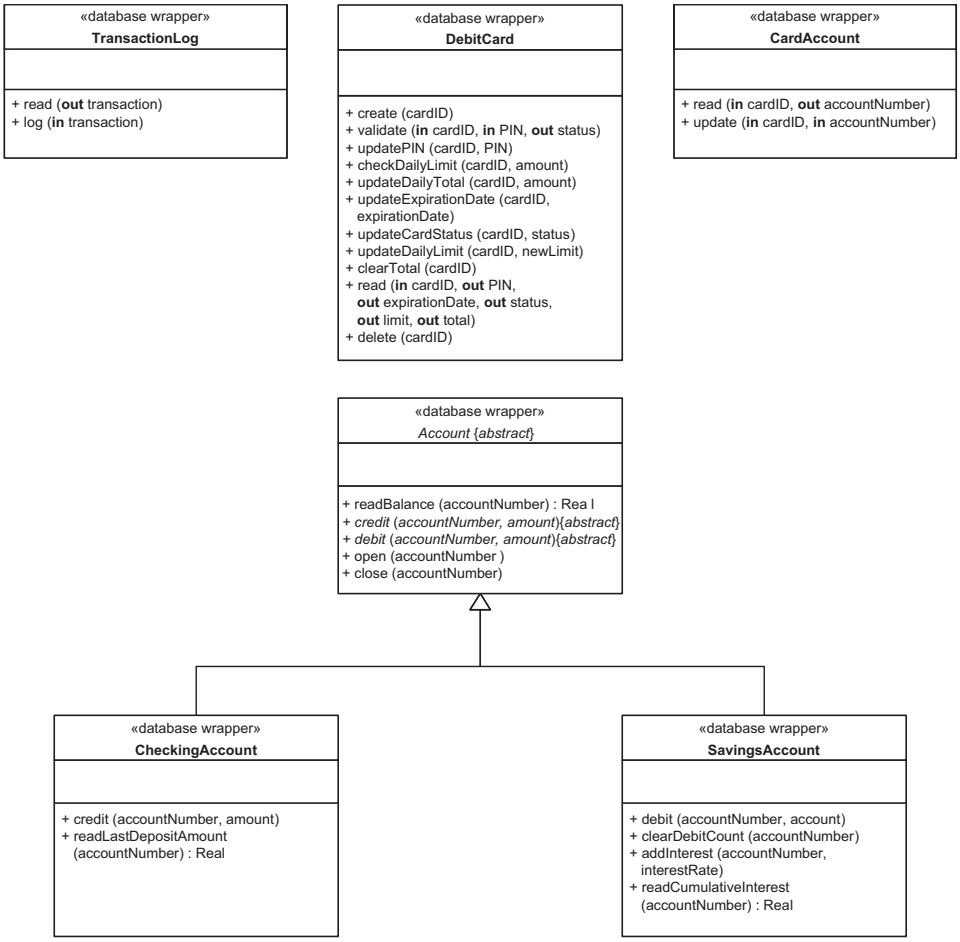


Figure 21.33. Banking Service database wrapper classes

21.11.2 Design the Banking Service Information Hiding Classes

Chapter 15 describes the design of database wrapper classes as well as the mapping of analysis model entity classes to design model database wrapper classes and relational tables (flat files) for a relational database. At the Banking Service, the database wrapper classes are Account, Checking Account, Savings Account, Debit Card, Card Account, and Transaction Log, as shown in Figure 21.33. Each of these classes encapsulates an interface to a database relation. Because a relational database consists of flat files and does not support class hierarchies, from a database perspective, the Account generalization/specialization hierarchy is flattened so that the attributes of the Account superclass are assigned to the Checking Account and Savings Account relations (as described in Chapter 15). However, in the Banking Service class design of the database wrappers, the Account generalization/specialization hierarchy is preserved so that the Checking Account and Savings Account database wrapper classes inherit generalized operations from the abstract Account superclass.

There are also four business logic classes whose interfaces need to be designed. These are the PIN Validation Transaction Manager, the Withdrawal Transaction

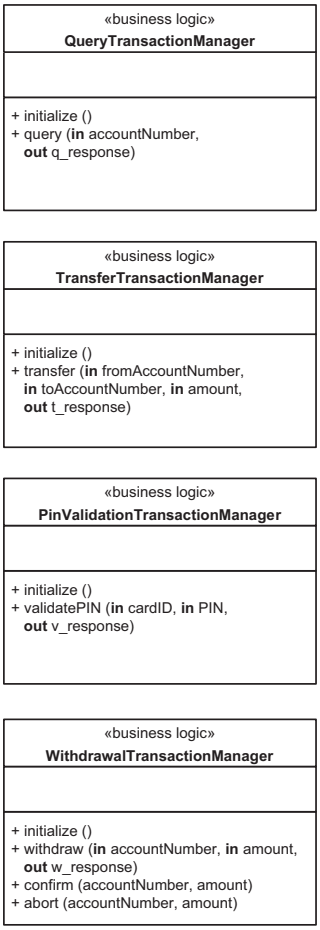


Figure 21.34. Banking Service business logic classes

Manager, the Query Transaction Manager, and the Transfer Transaction Manager, as shown in Figure 21.34. Each transaction manager handles an atomic transaction. For example, the Withdrawal Transaction Manager provides a withdraw operation, which is called to handle a customer request to withdraw funds, as well as two other operations. The confirm operation is called when an ATM Client confirms that the cash was dispensed to the client. The abort operation is called when an ATM Client aborts the transaction, for example, because the cash dispenser failed to dispense the cash or the customer cancelled the transaction.

21.11.3 Design the Banking Service Interfaces

The Banking Service is a sequential service subsystem with one thread of control. In particular, the design of the Banking Service task needs to be considered at this stage. The task is a composite task composed of passive objects. The Bank Transaction Coordinator receives incoming transactions and delegates them to the business logic objects, namely, the PIN Validation Transaction Manager, the Withdrawal Transaction Manager, the Query Transaction Manager, and the Transfer Transaction Manager.

The Bank Transaction Coordinator actually receives the messages FIFO from the ATM Clients. For each message, it determines the type of the transaction and then delegates the transaction processing to the appropriate transaction manager. Each transaction is processed to completion, with the response returned to the Bank Transaction Coordinator, which in turn sends the response to the appropriate ATM Client. The Bank Transaction Coordinator then processes the next transaction message.

Figure 21.32 shows the initial design of the Banking Service Subsystem In the initial concurrent communication diagram for the Banking Service, all interfaces are simple messages. Figure 21.35 shows the final version of the Banking Service Subsystem concurrent communication diagram. Communication between the multiple instances of the ATM Client and the Banking Service is synchronous with reply. All internal interaction within the Banking Service is between passive objects; hence, all internal interfaces are defined in terms of operation calls (depicted by using the synchronous message notation).

21.12 RELATIONAL DATABASE DESIGN

This section describes the logical design of the bank's relational database, starting from the conceptual entity class model described in Section 21.3.3 and depicted in Figures 21.4 through 21.7. All the entity classes depicted on the class diagram (Figure 21.4) reside on the bank server. The data held by these entity classes need to be persistent and therefore need to be stored in a database. As described in Section 21.12, the entity classes are designed as database wrapper classes, whereas the contents of the entity classes (as defined by the attributes of the entity classes) need to be stored in relational tables in the database. In the following description, primary keys are underlined and foreign keys are shown in italics: (underline = primary key, italic = *foreign key*).

The guidelines for designing a relational database from a static model are described in Section 15.5. Consider the entity classes in Figure 21.4. Each of the Bank, ATM Info, Customer, and Debit Card entity classes is mapped to a relational table. In each case, an attribute that uniquely locates a row of the respective table is made the primary key, such as the primary key customerId for the Customer table. Foreign keys are chosen to allow navigation between the tables.

For the Account generalization/specialization hierarchy, the decision is made to flatten the hierarchy by replicating the attributes of the superclass in the subclass tables Checking Account and Savings Account. Although account type (savings or checking) is an attribute of the Account classes, it is assumed that the account type can be determined from the Account Number; therefore, the primary key for both Checking Account and Savings Account tables is accountNumber. The association class Card Account (explicitly depicted in Figure 21.4) is designed as an association table, which represents the many-many relationship between Card and Account. Customer Account is also designed as an association table, representing the many-many relationship between Customer and Account. Even though a Customer Account association class is not explicitly modeled in the static mode (it is not needed by the ATM transactions), it is necessary in the relational database.

For the ATM Transaction generalization/specialization hierarchy, the same decision is made to flatten the hierarchy and only provide relational tables for the transaction subclasses. The primary key for an ATM transaction is the transaction

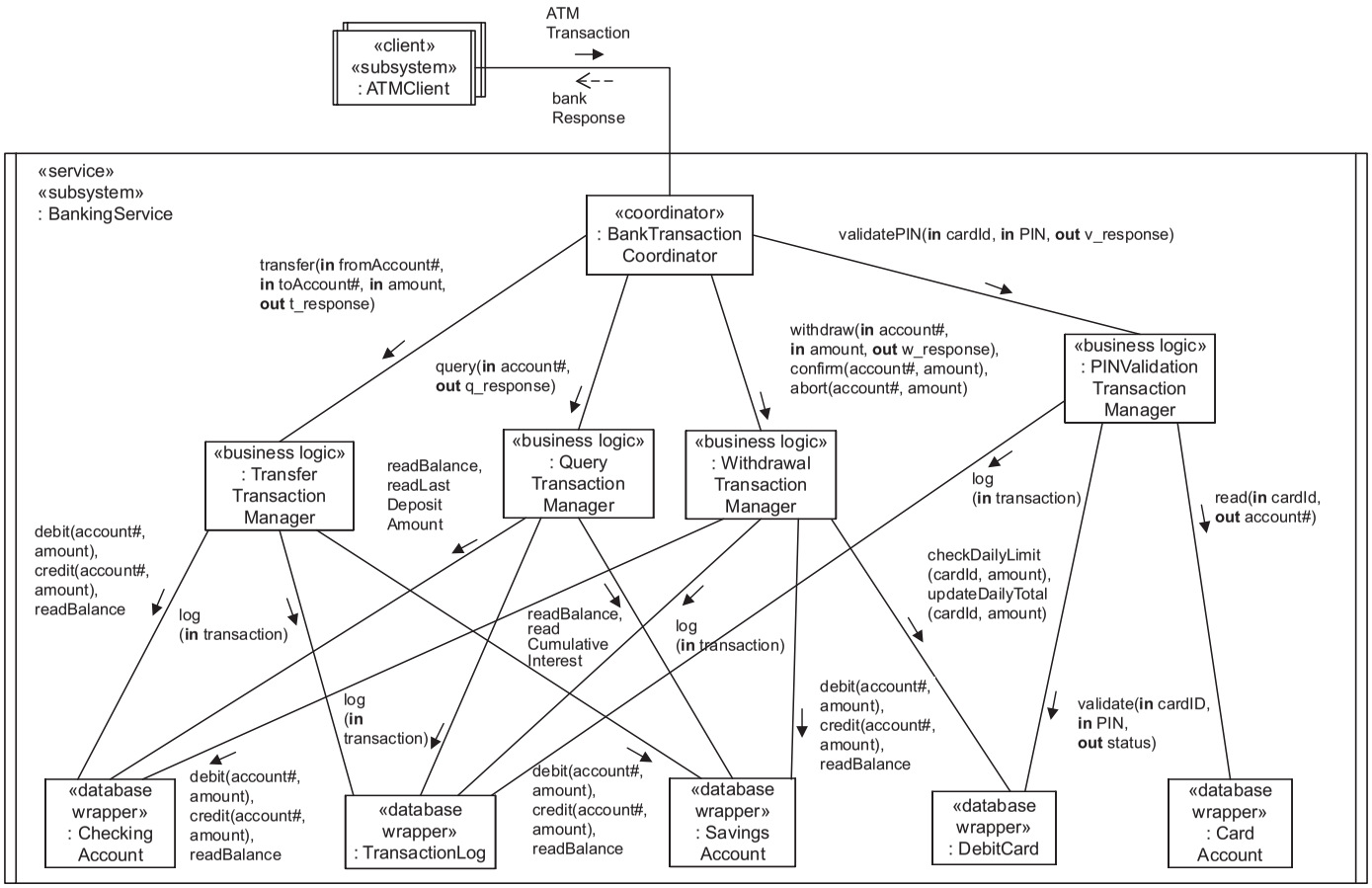


Figure 21.35. Revised concurrent communication diagram for Banking Service subsystem

Id, which consists of a concatenated key: bankId, ATMId, date, time. *bankId* and *ATMId* are also foreign keys because they allow navigation to the Bank and ATMInfo tables. ATMInfo has a concatenated primary key consisting of bankId, ATMId, with *bankId* also a foreign key. The attributes date and time provide a time stamp to uniquely identify a transaction.

Bank (bankName, bankAddress, bankId)
Customer (customerName, customer Id, customerAddress)
Debit Card (cardId, PIN, startDate, expirationDate, status, limit, total, *customerId*)
Checking Account (accountNumber, accountType, balance, lastDepositAmount)
Savings Account (accountNumber, accountType, balance, interest)
Card Account (cardId, accountNumber)
Customer Account (customerId, accountNumber)
ATM Info (*bankId*, ATMId, ATMLocation, ATMAddress)
Withdrawal Transaction (*bankId*, *ATMId*, date, time, transactionType, cardId, PIN, accountNumber, amount, balance)
Query Transaction (*bankId*, *ATMId*, date, time, transactionType, cardId, PIN, accountNumber, balance)
Transfer Transaction (*bankId*, *ATMId*, date, time, transactionType, cardId, PIN, fromAccountNumber, toAccountNumber, amount)
PIN Validation Transaction (*bankId*, *ATMId*, date, time, transactionType, cardId, PIN, startDate, expirationDate)

21.13 DEPLOYMENT OF BANKING SYSTEM

Because this is a client/server system, there are multiple instances of the client subsystem and one instance of the service subsystem. Each subsystem instance executes on its own node, as depicted in the deployment diagram in Figure 21.36. Thus, each instance of the ATM Client executes on an ATM node, and the one instance of the Banking Service executes on the server node.

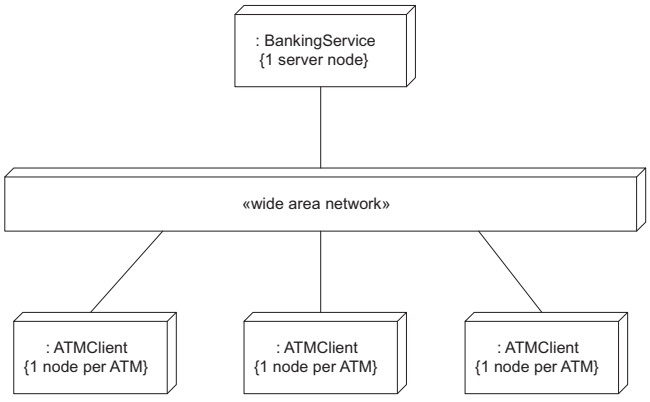


Figure 21.36. Deployment diagram for Banking System

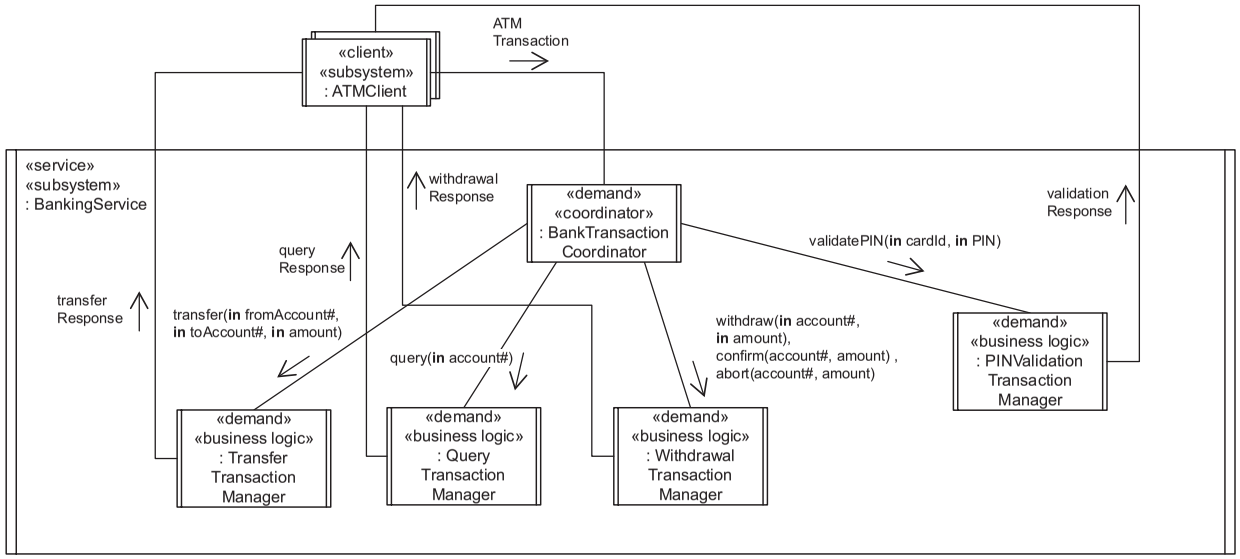


Figure 21.37. Alternative design for Banking Service: concurrent service design for Banking System

21.14 ALTERNATIVE DESIGN CONSIDERATIONS

An alternative design decision is to design the Banking Service as a concurrent service, in which the Bank Transaction Coordinator and each of the business logic objects are designed as separate demand driven tasks that are activated on demand. With this concurrent service design, the Bank Transaction Coordinator delegates a transaction to a business logic object and then immediately accepts the next transaction; thus, multiple transactions would be processed concurrently at the server. This solution should be adopted if the sequential service design is inadequate for handling the transaction load. For more information on the design of concurrent services, refer to Chapter 15.

21.15 DETAILED DESIGN

The detailed design of the Banking System is described in terms of the task event sequencing logic. Examples of task behavior specifications for the Card Reader Interface and ATM Control tasks in the ATM Client Subsystem and for the Banking Service task in the Banking Service Subsystem are given in Chapter 18. This section describes the event sequencing logic for these tasks.

21.15.1 Example of Event Sequencing Logic for Card Reader Interface Task

The Card Reader Interface task (see Figure 21.30) is awakened by a card reader external event, reads the ATM card input, writes the card contents to the ATM Card object, sends a cardInserted message to the ATM Control, and then waits for a message. If the message sent by the ATM Controller is eject, the card is ejected, and if it is confiscate, the card is confiscated. The passive data abstraction object, ATM Card, is outside the task and is used to store the contents of the card.

All message communication in the Banking System is through calls to the operating system. Thus, the message queue, ATMControlMessageQ, between the Card Reader Interface task (producer) and ATM Control (consumer) is provided by the operating system, as is the synchronous communication between ATM Control and the Card Reader Interface task (see Figure 21.32). A synchronous message from ATM Control is received in a message buffer called cardReaderMessageBuffer.

```
Initialize card reader;
loop
-- Wait for external interrupt from card reader
wait (cardReaderEvent);
Read card data held on card's magnetic strip;
if card is recognized
then -- Write card data to ATM Card object;
    ATMCard.write (cardID, startDate, expirationDate);
-- send card Inserted message to ATM Control;
send (ATMControlMessageQ, cardInserted);
-- Wait for message from ATM Control;
```

```
receive (cardReaderMessageBuffer, message);
if message = eject
then
    Eject card;
    -- Send card Ejected message to ATM Control;
    send (ATMControlMessageQ, cardEjected);
 elseif message = confiscate
    then
     Confiscate card;
     -- Send card Confiscated message to ATM Control;
     send (ATMControlMessageQ, cardConfiscated);
    else error condition;
 end if;
else -- card was not recognized so eject;
  Eject card;
end if;
end loop;
```

21.15.2 Example of Event Sequencing Logic for ATM Control Task

The ATM Control task is at the heart of the ATM Client subsystem (see Figure 21.30) and interacts with several tasks. ATM Control has an input message queue called ATM-ControlMessageQ, from which it receives messages from its three producers – Card Reader Interface, Customer Interaction, and Operator Interaction. ATM Control sends messages to several tasks. It sends synchronous messages without reply to the Card Reader Interface. It sends synchronous messages with reply to the Cash Dispenser Interface and Receipt Printer Interface tasks. It sends asynchronous messages to the Customer Interaction task on the promptMessageQueue message queue. It sends synchronous messages with reply to the Banking Service.

Because it is state-dependent, the ATM Control task does not process incoming events but rather the state-dependent actions as given by the statechart. The implementation of the statechart is encapsulated in the ATM State Machine state-machine object, which is nested inside ATM Control. Given the new event, the process Event operation returns the action(s) to be performed. Most events are received on the ATM Control input message queue, although there are three exceptions to this. Because the communication with the Banking Service is synchronous, the response is received as the output parameter of the send message. Because of the synchronous communication with the Cash Dispenser Interface and the Receipt Printer Interface tasks, the dispense Cash and print Receipt actions are synchronous messages with reply, which return whether the respective dispensing and printing actions were successful.

When an event is generated internally as a result of a response to a synchronous message, the variable newEvent is set to the value of this event and the Boolean variable outstandingEvent is set to True. Examples of such internal events are withdrawalResponse (several synchronous bank responses are possible, as described in the next section in the event sequencing logic for the Banking Service) or cash

Dispensed. The event sequencing logic is given below, which describes most of the actions executed by ATM Control. After the execution of each action case, the next execution step is a transfer to the end of the pseudocode case block (for brevity, the transfer is not explicitly shown below). The pseudocode for the ATM Control task follows:

```
loop
    -- Messages from all senders are received on Message Queue
    Receive (ATMControlMessageQ, message);
    -- Extract the event name and any message parameters
    -- Given the incoming event, lookup state transition table;
    -- change state if required; return action to be performed;
    newEvent = message.event
    outstandingEvent = true;
while outstandingEvent do
    ATMStateMachine.processEvent (in newEvent, out action);
    outstandingEvent = false;
    -- Execute action(s) as given on ATM Control statechart
case action of
    Get PIN: -- Prompt for PIN;
        send (promptMessageQueue, displayPINPrompt);
    Validate PIN: --Validate customer entered PIN at Banking Service;
        send (Banking Service, in validatePIN, out
          validationResponse);
        newEvent = validationResponse; outstandingEvent = true;
    Display Menu: -- Display selection menu to customer;
        send (promptMessageQueue,displayMenu);
        ATMTransaction.updatePINStatus (valid);
    Invalid PIN Action: -- Display Invalid PIN prompt;
        send (promptMessageQueue, displayInvalidPINPrompt);
        ATMTransaction.updatePINStatus (invalid);
    Request Withdrawal: -- Send withdraw request to Banking
            Service;
        send (promptMessageQueue, displayWait);
        send (Banking Service, in withdrawalRequest, out
          withdrawalResponse);
        newEvent = withdrawalResponse; outstandingEvent = true;
    Request Query: -- Send query request to Banking Service;
        send (promptMessageQueue, displayWait);
        send (Banking Service, in queryRequest, out queryResponse);
        newEvent = queryResponse; outstandingEvent = true;
    Request Transfer: -- Send transfer request to Banking Service;
        send (promptMessageQueue, displayWait);
        send (Banking Service, in transferRequest, out
            transferResponse);
        newEvent = transferResponse; outstandingEvent = true;
```

```
    Dispense: -- Dispense cash and update transaction status;
        ATMTransaction.updateTransactionStatus (withdrawalOK);
        send (cashDispenserInterface, in cashAmount, out dispenseStatus);
    newEvent = cashDispensed; outstandingEvent = true;
    Print: -- Print receipt and send confirmation to Banking
            Service;
        send (promptMessageQueue, displayCashDispensed);
        send (Banking Service, in confirmRequest);
        send (receiptPrinterInterface, in receiptInfo, out
           printStatus);
        newEvent = receiptPrinted; outstandingEvent = true;
    Eject: -- Eject ATM card;
        send (cardReaderInterface, eject);
    Confiscate: -- Confiscate ATM card;
        send (cardReaderMessageBuffer, confiscate);
        ATMTransaction.updatePINStatus (status);
    Display Ejected: -- Display Card Ejected prompt;
        send (promptMessageQueue, displayEjected);
    Display Confiscated: -- Display Card Confiscated prompt;
        send (promptMessageQueue, displayConfiscated);
    . . .
    end case;
end while;
end loop;
```

21.15.3 Example of Event Sequencing Logic for Banking Service Task

The Banking Service receives messages from all the ATM Clients (see Figure 21.36). Although the communication is synchronous with reply, a message queue can build up at the Banking Service as it receives messages from multiple ATM clients. In this sequential solution, the Banking Service is a sequential service task, which processes each request to completion before starting the next.

```
loop
receive (ATMClientMessageQ, message) from Banking Service Message Queue;
Extract message name and message parameters from message;
case Message of
 Validate PIN:
     -- Check that ATM Card is valid and that PIN entered by
     -- customer matches PIN maintained by Server;
    PINValidationTransactionManager.ValidatePIN
        (in CardId, in PIN, out validationResponse);
    -- If successful, validation Response is valid and return
    -- Account Numbers accessible by this debit card;
```

```
    -- otherwise validation Response is invalid,
    -- third Invalid, or stolen;
    reply (ATMClient, validationResponse);
Withdrawal:
    -- Check that daily limit has not been exceeded and that
    -- customer has enough funds in account to satisfy request.
    -- If all checks are successful, then debit account.
    WithdrawalTransactionManager.withdraw
      (in AccountNumber, in Amount, out withdrawalResponse);
    -- If approved, then withdrawal Response is
    -- {successful, amount, currentBalance};
    -- otherwise withdrawalResponse is {unsuccessful};
    reply (client, withdrawalResponse);
Query:
    -- Read account balance
    queryTransactionManager.query
    (in accountNumber, out queryresponse);
    -- Query Response = Current Balance and either Last Deposit
    -- Amount (checking account) or Interest (savings acount);
    reply (client, queryResponse);
Transfer:
    -- Check that customer has enough funds in From Account to
    -- satisfy request. If approved, then debit From Account
    -- and credit To Account;
    transferTransactionManager.transfer (in fromAccount#,
      in toAccount#, in amount, out transferResponse);
    -- If approved, then transfer Response is
    -- {successful, amount, Current Balance of From Account};
    -- otherwise Transfer Response is {unsuccessful};
    reply (client, transferResponse);
Confirm:
    -- Confirm withdrawal transaction was completed successfully
    withdrawalTransactionManager.confirm (in accountNumber, in amount);
Abort:
    -- Abort withdrawal transaction
    withdrawalTransactionManager.abort (in accountNumber, in amount);
end case;
end loop;
```