OBJECT-ORIENTED MODELING AND DESIGN

Domain Analysis

CONTENTS :

1. Overview of Analysis
2. Domain Class Model
3. Domain State Model
4. Domain Interaction Model
5. Iterating the Analysis
6. Chapter Summary

1. Overview of Analysis

- The problem statement should not be taken as immutable, but rather as a basis for refining the requirements.
- Analysis is divided into two sub stages
- Domain analysis
- Application analysis

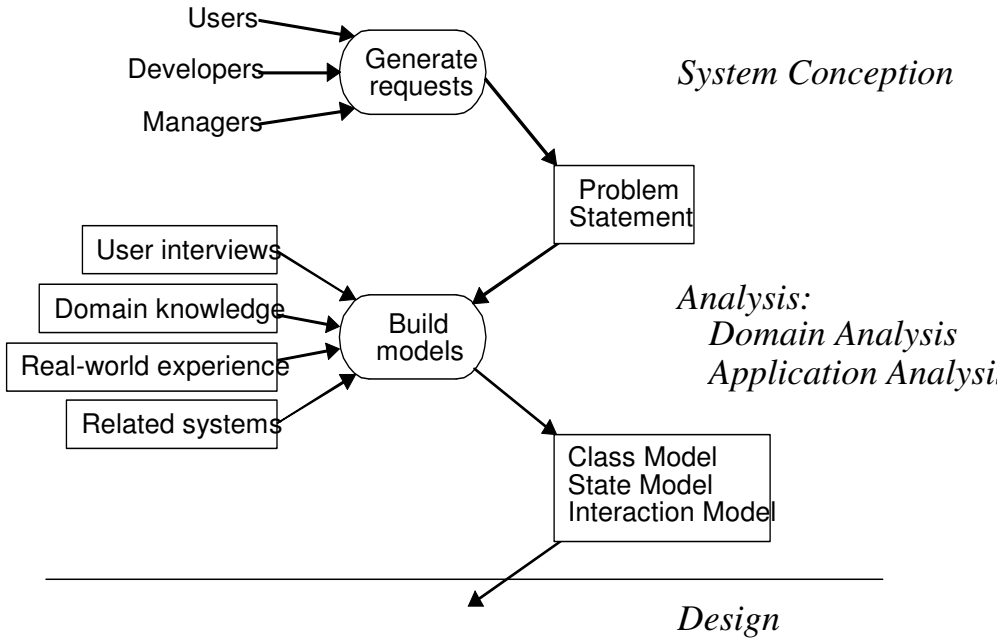


Fig 12.1 Overview of Analysis

2. Domain Class Model

- First step in analyzing the requirements is to construct a domain model.
- Static structure of the real world system is captured.
- The domain model describes real-world classes and their relationships to each other.
- Information for the domain model comes from the problem statement, artifacts from related systems, expert knowledge of the application domain and general knowledge of the real world.

The steps to be performed to construct a domain class model:

- Find Classes.
- Prepare a data dictionary.
- Find associations.
- Find attributes of objects and links.
- Organize and simplify classes using inheritance.
- Verify that access paths exist for likely queries.
- Iterate and refine the model.
- Reconsider the level of abstraction.
- Group classes into packages

- **Finding classes**

  - Classes often correspond to nouns. Ref figure 12.2.
  - Eg- ” a reservation system sell tickets to performances at various theater”- Tentative classes would be Reservation, System, Tickets, Performance and Theaters.
  - Idea is to capture concepts. not all nouns are concepts, and concepts are also expressed in other parts of speech.

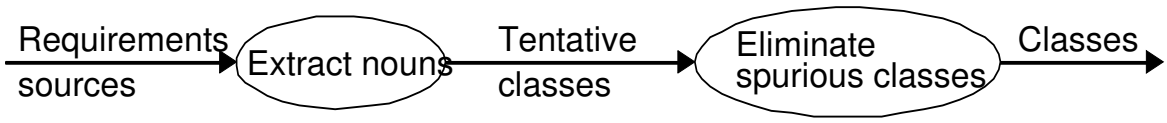


Figure: 12.2 Finding Classes-by considering nouns

For the Case study of the ATM: The following are the classes extracted from problem statement nouns. Figure 12.3

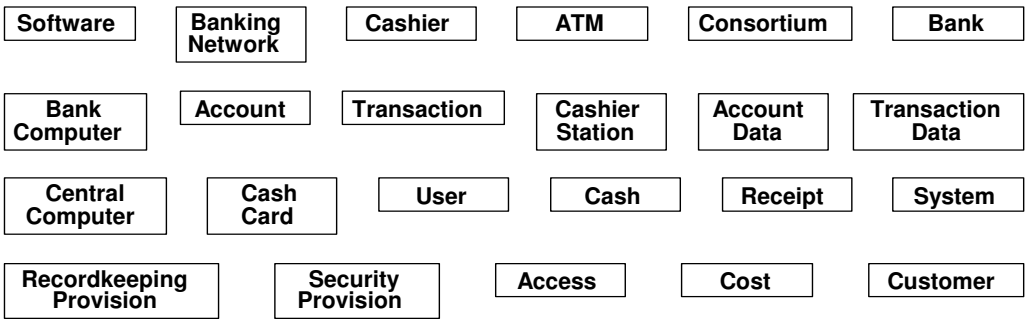


Figure: 12.3 ATM classes extracted from problem statement nouns

- Additional classes that do not appear directly in the statement but can be identified from our knowledge of the problem domain

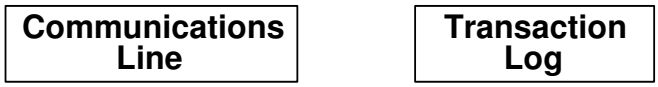


Figure:12.4 ATM classes identified from knowledge of problem domain

- Domain Class Model: Keeping the right classes

  Discard unnecessary and incorrect classes according to the following criteria

- Redundant classes: If two classes express the same concept, you should keep the most descriptive name.

  ATM example. Customer and user are redundant; we retain customer because it is more descriptive.

- Irrelevant classes
  Apportioning cost is outside the scope of the ATM software.

- Vague classes: class should be specific.

- Attributes: Names that primarily describe individual objects should be restated as attributes.
- Operations: If a name describes an operation that is applied to objects and not manipulated in its own right, then it is not a class.
  Eg-if we are simply building telephones, then call is part of the state model and not a class
- Roles: The name of a class should reflect its intrinsic nature and not a role that it plays in an association.
  - Owner of a car..i n a car manufacturing database, not correct as a class. It can be a person(owner, driver, lessee)

- Implementation Constructs:
  Eliminate constructs from the analysis model that are extraneous to the real world. We may need them during design and not now.

- Derived classes:
  As a general rule, omit classes that can be derived from other classes. mark all derived classes with a preceding slash('/')in the class name.

Keeping the right classes:

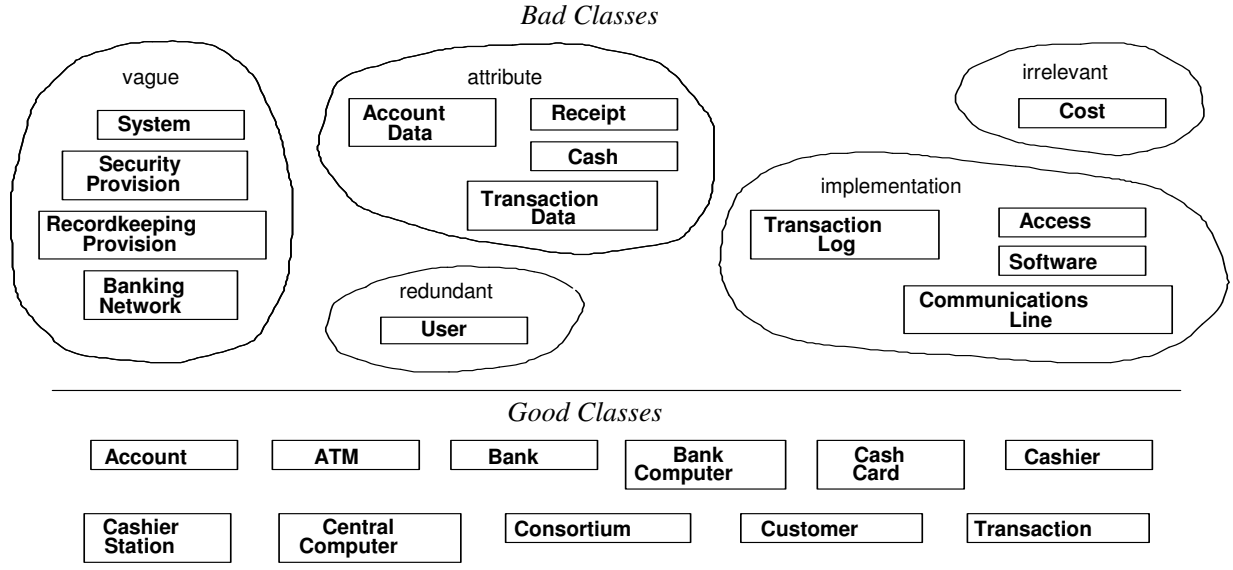


Figure:12.5 Eliminating unnecessary classes from ATM Problem.

Preparing a Data Dictionary

- Prepare a data dictionary for all modeling elements.
- Describe the scope of the class within the current problem, including all assumptions or restrictions on its use.
- DD also describes associations, attributes, operations and enumeration values

Data Dictionary for the ATM classes

Account	ATM
Bank	BankComputer
CashCard	Cashier
cashierStation	CentralComputer
Consortium	Customer
Transaction	

Finding associations

- A structural relationship between two or more classes is an association.
- A reference from one class to another is an association.
- Associations often correspond to verbs or verb phrases.
- Idea here is to capture relationships

Verb phrases

Banking network includes cashier stations and ATMs
Consortium shares ATMs
Bank provides bank computer
Bank computer maintains accounts
Bank computer processes transaction against account
Bank owns cashier station
Cashier station communicates with bank computer
Cashier enters transaction for account
ATMs communicate with central computer about transaction
Central computer clears transaction with bank
ATM accepts cash card
ATM interacts with user
ATM dispenses cash
ATM prints receipts
System handles concurrent access
Banks provide software
Cost apportioned to banks

Implicit verb phrases

Consortium consists of banks
Bank holds account
Consortium owns central computer
System provides recordkeeping
System provides security
Customers have cash cards

Knowledge of problem domain

Cash card accesses accounts
Bank employs cashiers

Keeping the Right Association

Discard unnecessary associations, using the following criteria:

- Associations between eliminated classes.
- Irrelevant or implementation associations.
- Actions: An association should describe a structural property of the application domain not a transient event.
- Ternary associations.
- Derived associations :they may be redundant

Further specify the semantics of associations as follows:

- Misnamed associations.
- Association end names.
- Qualified associations.
- Multiplicity
- Missing associations
- Aggregation.
- Fig 12.9 shows class diagram with remaining associations

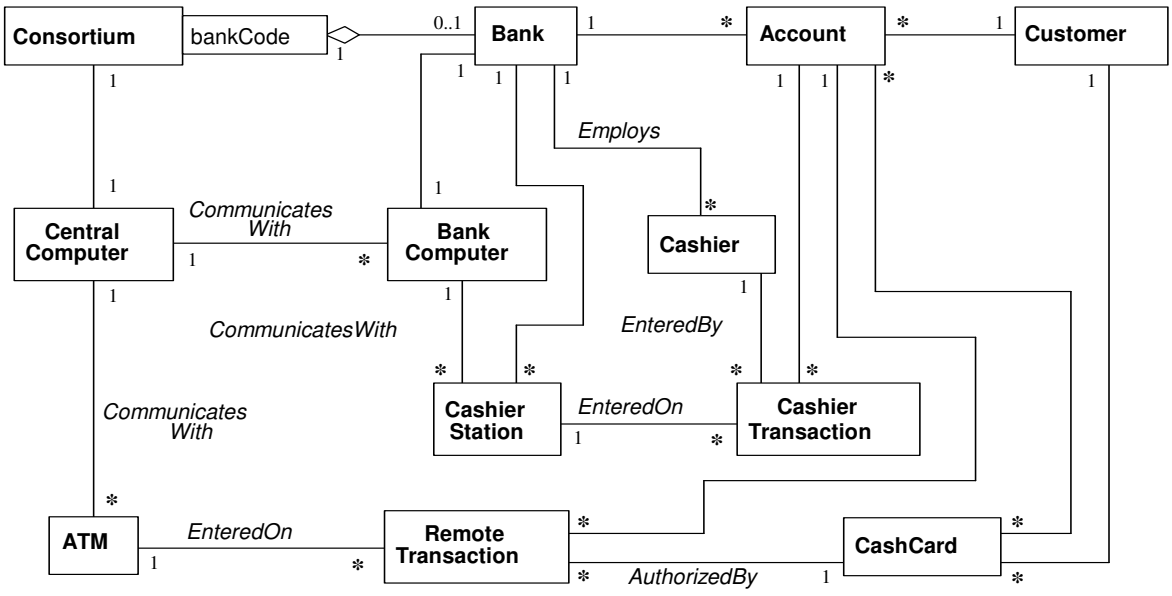


Figure:12.9 Initial class diagram for ATM system

Find attributes of objects and links.

- Attributes are data properties of objects
- Attribute values should not be objects; use an association to show any relationship between objects.
- Attributes usually correspond to nouns followed by possessive phrases, such as "the color of the car

Keeping the right attributes:
Eliminate unnecessary and incorrect attributed with the following criteria:

- Objects . Discordant attributes
- Qualifiers . Boolean attributes.
- Names
- Identifiers
- Attributes on associations
- Internal values
- Fine detail

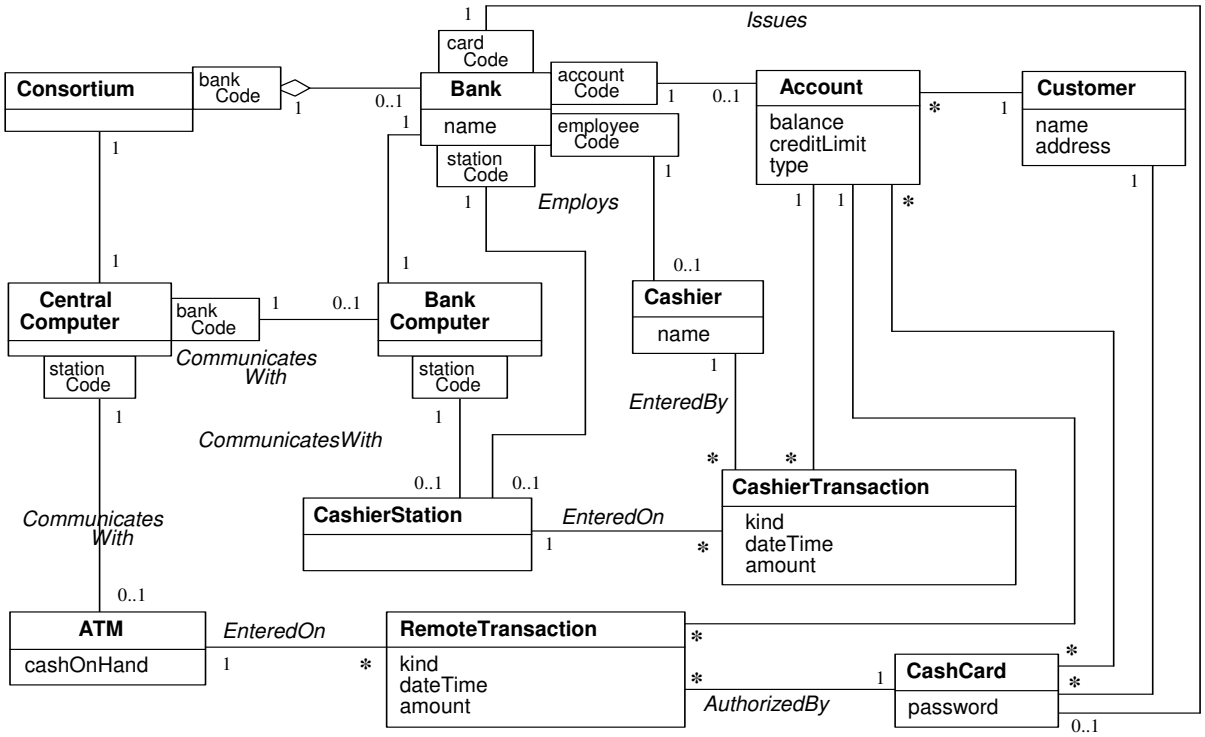


Figure:12.10 ATM class model with attributes

Refining With Inheritance

Organize classes by using inheritance to share common features:

- Bottom up generalization.
- Top-down generalization
- Generalization vs. enumeration.
- Multiple inheritance
- Similar associations
- Adjusting the inheritance level.
- Fig 12.11 shows the ATM class model after adding inheritance

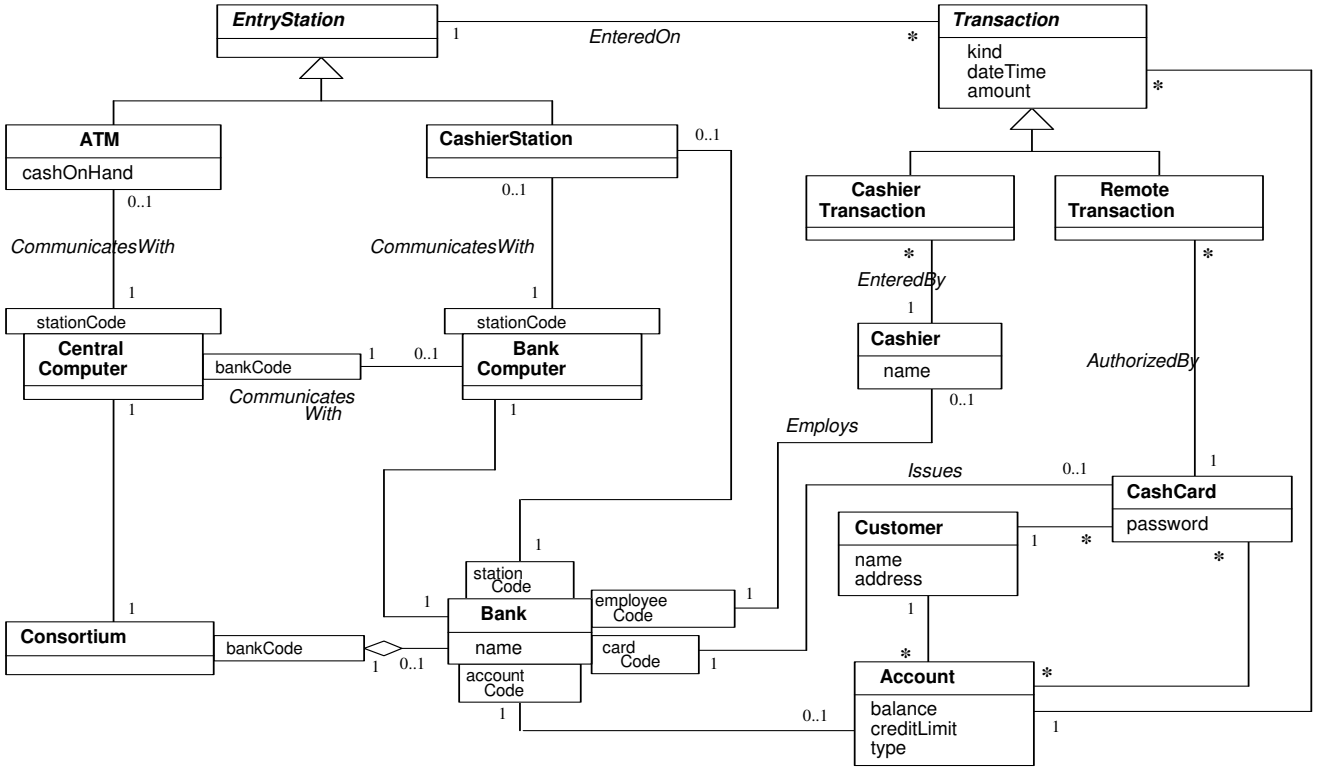


Figure:12.11 ATM class model with attributes and inheritance

- Verify that access paths exist for likely queries.
  Trace access paths through the class model to see if they yield sensible results.

- Iterate and refine the model.
  A class model is rarely correct after a single pass, so iterate and refine the model.
- Fig 12.12 shows a revised class diagram that is simpler and cleaner.

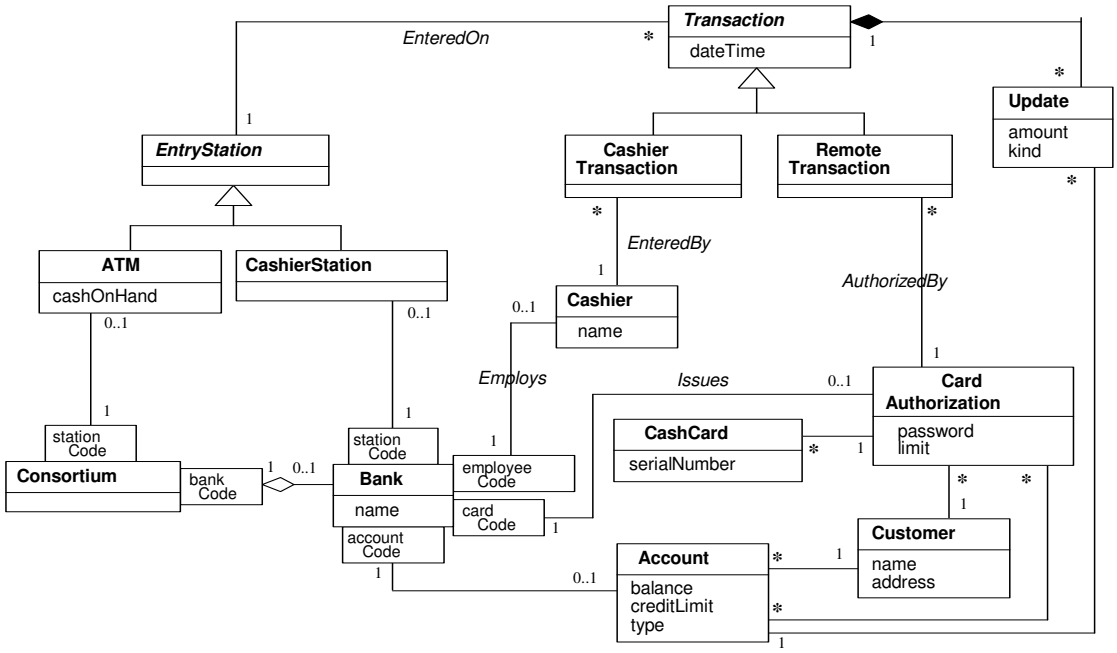
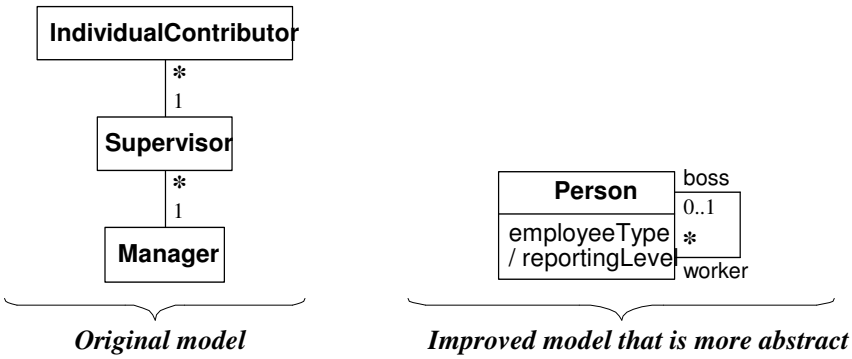


Figure:12.12 ATM class model after further revision.

- Reconsider the level of abstraction.
  Abstraction makes a model more complex but can increase flexibility and reduce the number of classes.



- Group classes into packages.
    - The last step of class modeling is to group classes into packages.
    - A package is a group of elements(classes, association, generalizations and lesser packages) with common theme.

3. Domain State Model

The Following steps are performed in constructing a domain state model

- Identifying classes with states
- Finding states
- Finding Events
- Building state diagrams
- Evaluating state diagrams

After running thro the above steps, the domain state model obtained is a shown in Figure 12.14

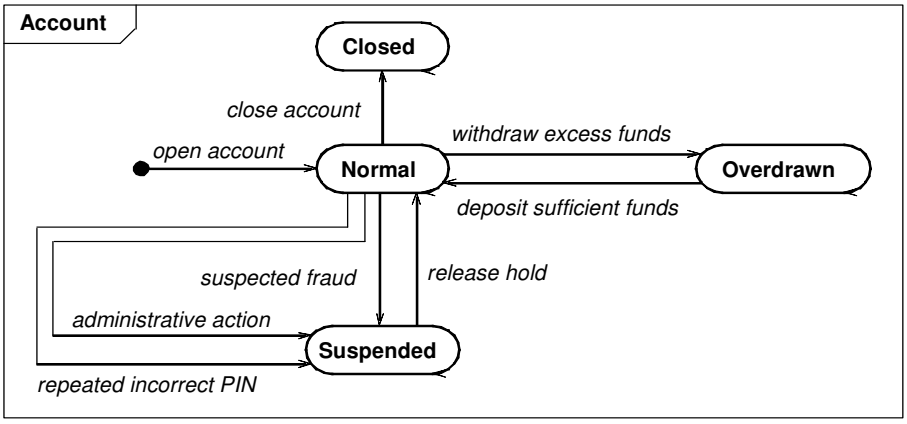


Figure: 12.14 Domain state model.

4. Domain Interaction Model

- The Interaction model is seldom important for domain analysis.
- The Interaction model is an important aspect of application modeling

5. Iterating the Analysis

The iteration to the analysis should ne done as follows:

- Refining the Analysis Model
- Restating the Requirements
- Analysis and Design.

6. Chapter Summary.

- The domain model captures general knowledge about the application-concepts and relationships know to experts in the domain.
- The domain model has class model and state model but seldom has an interaction model.
- A good analysis captures the essential features without introducing the implementation artifacts that prematurely restrict the design decisions.
- The result of analysis replaces the original problem statement and serves as the basis for design

Exercise

1. For each of the following systems, identify the relative importance of the three aspects of modeling
   1) Class modeling 2) state modeling 3) interaction modeling

   a. **bridge player**
   b. **change making machine**
   c. **car cruise control**
   d. **electronic typewriter**
   e. **spelling checker**
   f. **telephone answering machine.**

Answer:

a. bridge player. Interaction modeling, class modeling, and state modeling, in that order, are important for a bridge playing program because good algorithms are needed to yield intelligent play. The game involves a great deal of strategy. Close attention to inheritance and method design can result in significant code reuse. The interface is not complicated, so the state model is simple and could be omitted.

b. change-making machine. Interaction modeling is the most important because the machine must perform correctly. A change making machine must not make mistakes; users will be angry if they are cheated and owners of the machine do not want to lose money. The machine must reject counterfeit and foreign money, but should not reject genuine money. The state model is least important since user interaction is simple.

c. car cruise control. The order of importance is state modeling, class modeling, and interaction modeling. Because this is a control application you can expect the state model to be important. The interaction model is simple because there are not many classes that interact.

d. electronic typewriter. The class model is the most important, since there are many parts that must be carefully assembled. The interaction between the parts also must be thoroughly understood. The state model is least important.

e. spelling checker. The order of importance is class modeling, interaction modeling, and state modeling. Class modeling is important because of the need to store a great deal of data and to be able to access it quickly. Interaction modeling is important because an efficient algorithm is needed to check spelling quickly. The state model is simple because the user interface is simple: provide a chance to correct each misspelled word that is found.

f. telephone answering machine. The order of importance is state modeling, class modeling, and interaction modeling. The state diagram is non-trivial and important to the behavior of the system. The class model shows relationships between components that complement the state model. There is little computation or state diagrams to interact, so the interaction model is less important.

2. The following is the class model for the scheduler software:

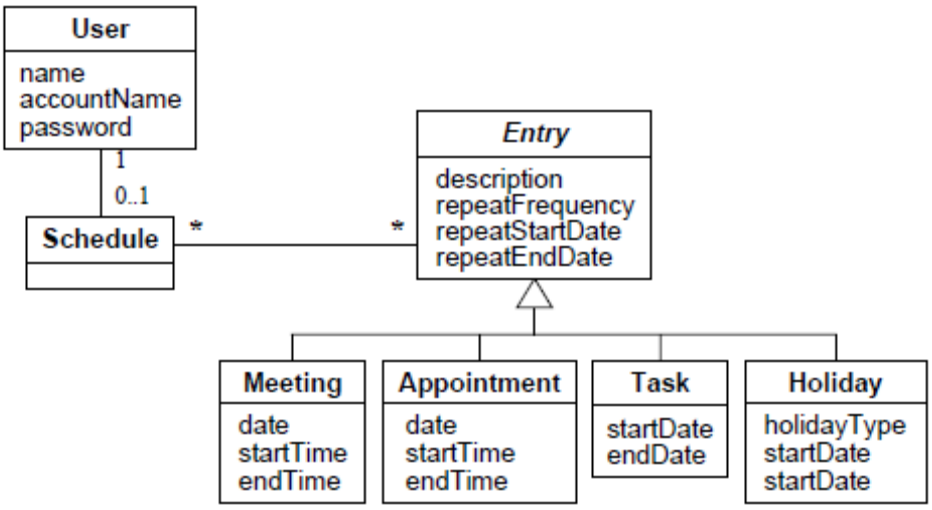


Figure A12.12 Class model for scheduler software

3. Ref to the Question 2. The following is a list of candidate classes. Prepare a list of classes that should be eliminated for any of the reasons listed in the chapter. Give reasons for each elimination. If there is more than one reason, give the main one.

   Scheduling software, meeting user, chairperson, software, meeting entry, schedule, attendee, scheduler, room, time, everyone, attendance, acceptance status, meeting notice, invitation, meeting information, invitee, notice.

Answer:

The following tentative classes should be eliminated.

Redundant classes. *Invitee* is synonymous with *user* and could be a role, depending on the realization of the class model. *Everyone* refers to *user* and is redundant. *Meeting notice* is the same as *notice*; we keep *notice*. *Attendance* is also extraneous and merely refers to an attendee participating in a meeting. *Meeting* and *meeting entry* are synonymous; we keep *meeting*.

Irrelevant or vague classes. We discard *scheduling software, scheduler, software,* and *meeting information* because they are irrelevant.

Attributes. We model *time* and *acceptance status* as attributes. An *invitation* is a type of notice; we need the enumeration attribute *noticeType* to record whether an invitation is an invitation, reschedule, cancellation, refusal, or confirmation.

Roles. *Chair person* and *attendee* are association ends for user.