Chapter 5

SYSTEM MODELLING

CHAPTER OUTLINE

After studying this chapter, the reader will be able to understand the

- Introduction to System Modelling
- Context Models
- Interaction Models
- Behavioural Models

INTRODUCTION TO SYSTEM MODELLING

System modelling is approach of developing system's abstract models for representing its different views or perspective. Now a day's system modeling is mainly represented in graphical notation which is based on notations in the Unified Modelling Language (UML). It helps system analyst to validate the system's functionality and helps to communicate clearly with the customers about their needs. Hence, it is mainly used for requirement engineering. System models mainly help in identifying and validating the requirements of the new system by finding scope and limitation of the existing system. Engineers use these models to discuss design proposals and to document the system for implementation. In a model-driven engineering process, it is possible to generate a complete or partial system implementation from the system model. Models are used during the requirements engineering process to help derive the requirements for a system, during the design process to describe the system to engineers implementing the system and after implementation to document the system's structure and operation. You may develop models of both the existing system and the system to be developed:

- Models of the existing system are used during requirements engineering. They help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system.
- Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation. In a model-driven engineering process, it is possible to generate a complete or partial system implementation from the system model.

The most important aspect of a system model is that it leaves out detail. A model is an abstraction of the system being studied rather than an alternative representation of that system. Ideally, a representation of a system should maintain all the information about the entity being represented but unfortunately, the real world (also known as the universe of discourse) is utterly complex so weed to simplify. An abstraction consciously simplifies and picks out the most evident characteristics. You may develop different models to represent the system from different perspectives. For example:

- An external perspective, where you model the context or environment of the system.
- An interaction perspective where you model the interactions between a system and its environment or between the components of a system.
- A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.

CONTEXT MODELS

Context models represents the operational environment of the system, they represent what lies outside the system boundaries. The environment of the system contains social and organisational concerns which directly or indirectly effects on the position of the system boundaries. Hence, architectural models are used to show the system and its relationship with other systems.

System boundaries are established to define what is inside and what is outside the system. They show other systems that are used or depend on the system being developed. The position of the system boundary has a profound effect on the system requirements. Defining a system boundary is a political judgment. There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.

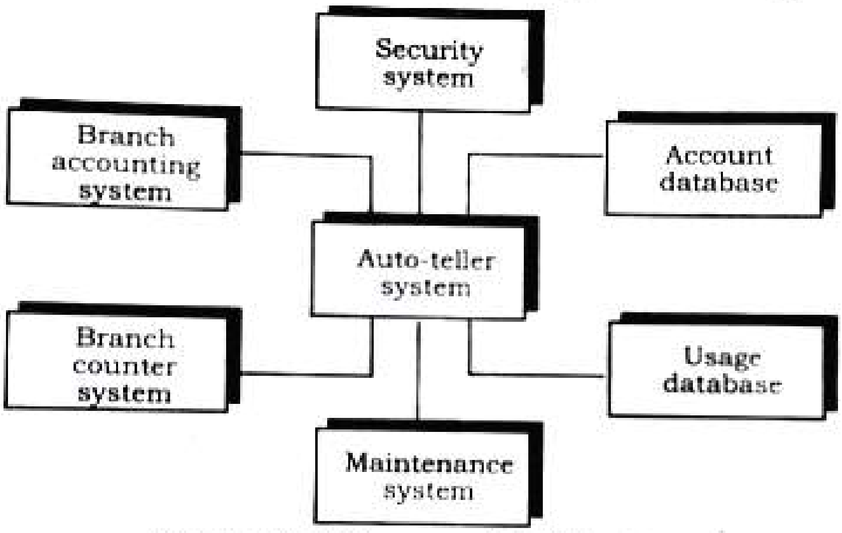

Fig: Context Diagram of ATM System

Context models simply show the other systems in the environment, not how the system being developed is used in that environment. Process models reveal how the system being developed is used in broader business processes. UML activity diagrams may be used to define business process models.

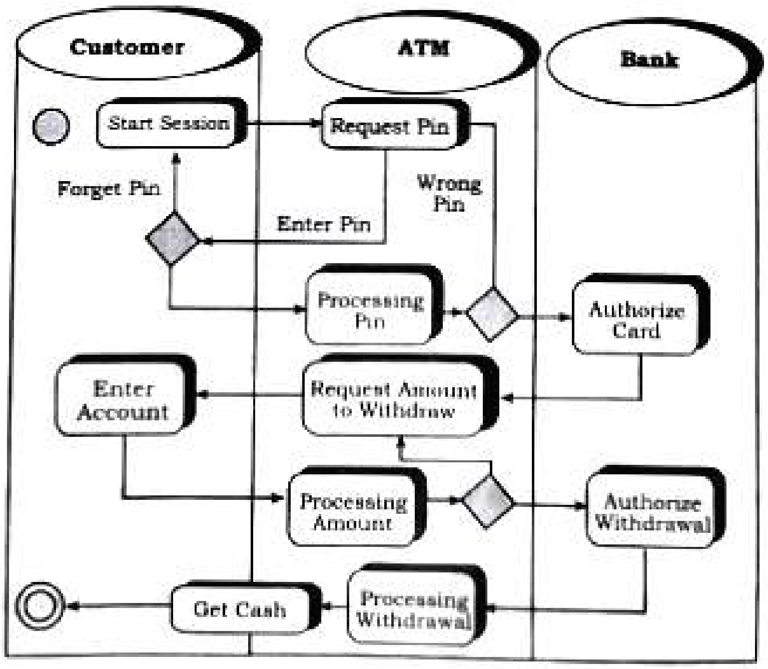

Fig: Activity Diagram of ATM System

INTERACTION MODELS

Modelling user interaction is important as it helps to identify user requirements. Modelling system-to-system interaction highlights the communication problems that may arise. Modelling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability. Use case diagrams and sequence diagrams may be used for interaction modelling.

Use cases were developed originally to support requirements elicitation and now incorporated into the UML. Each use case represents a discrete task that involves external interaction with a system. Actors in a use case may be people or other systems. Represented diagrammatically to provide an overview of the use case and in a more detailed textual form.

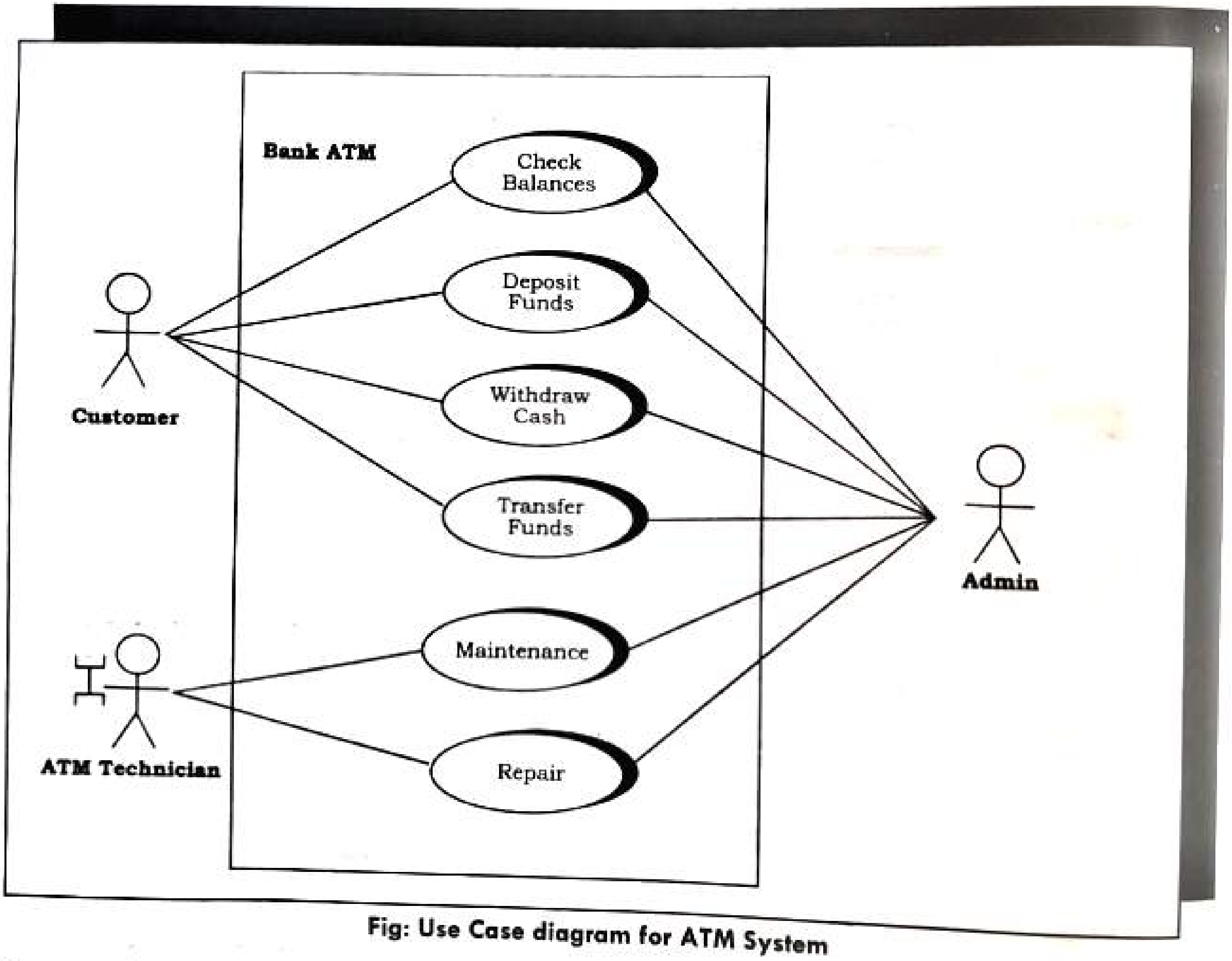

Fig: Use Case diagram for ATM System

Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system. A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance. The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these. Interactions between objects are indicated by annotated arrows.

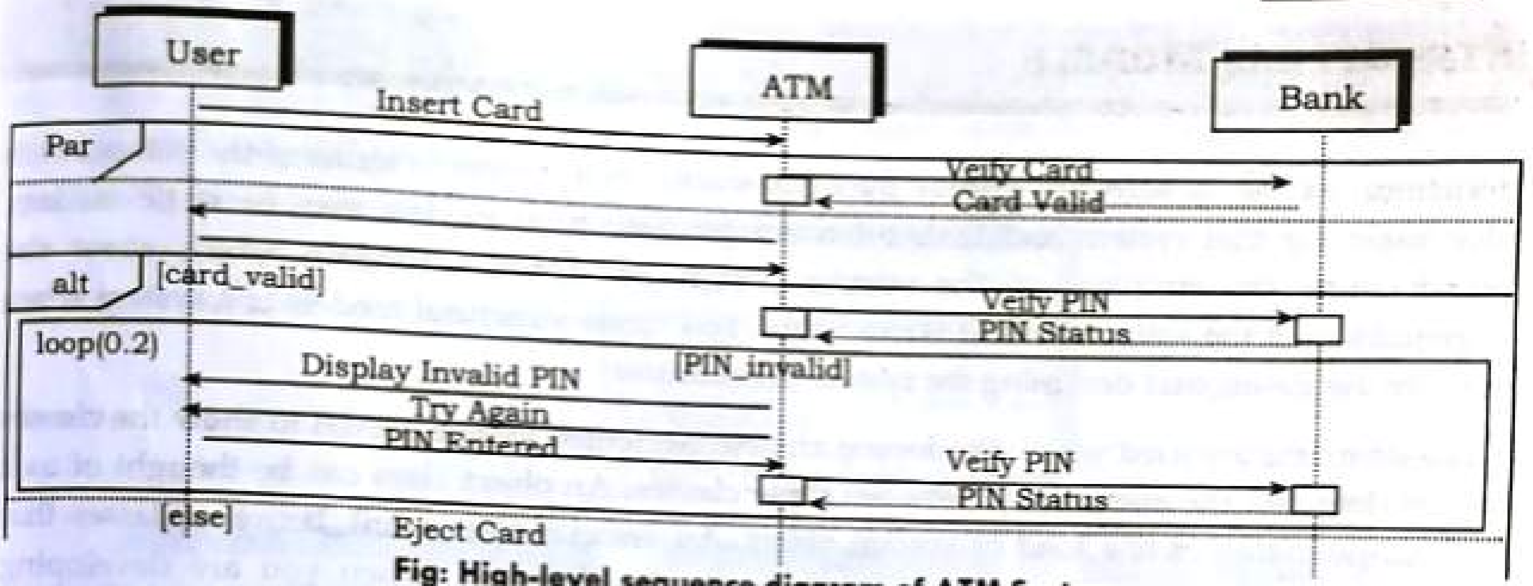

Fig: High-level sequence diagram of ATM System

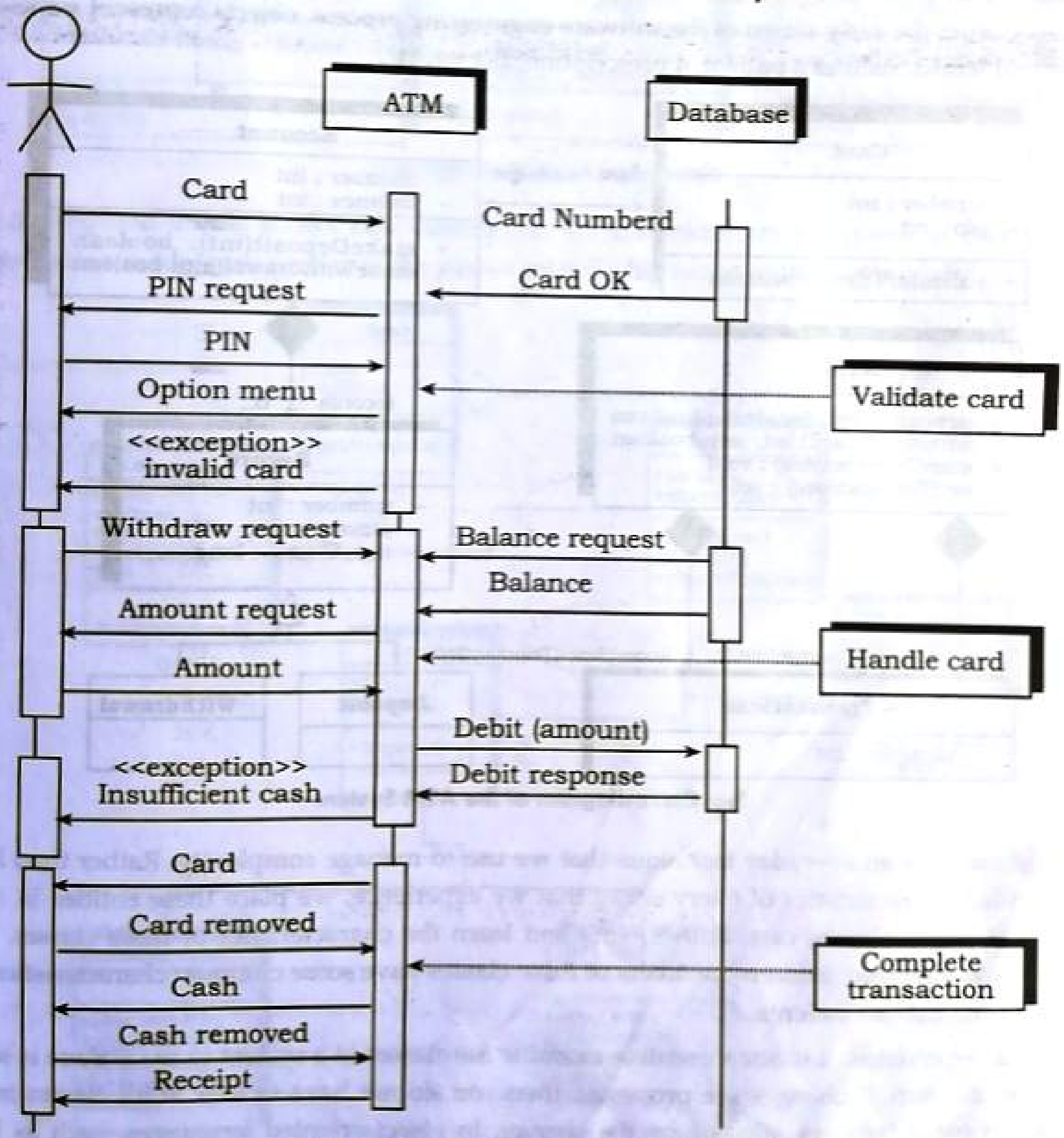

Fig: Sequence Diagram of ATM withdrawal

STRUCTURAL MODELS

Structural models of software display the organization of a system in terms of the components that make up that system and their relationships. Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing. You create structural models of a system when you are discussing and designing the system architecture.

Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes. An object class can be thought of as a general definition of one kind of system object. An association is a link between classes that indicates that there is some relationship between these classes. When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

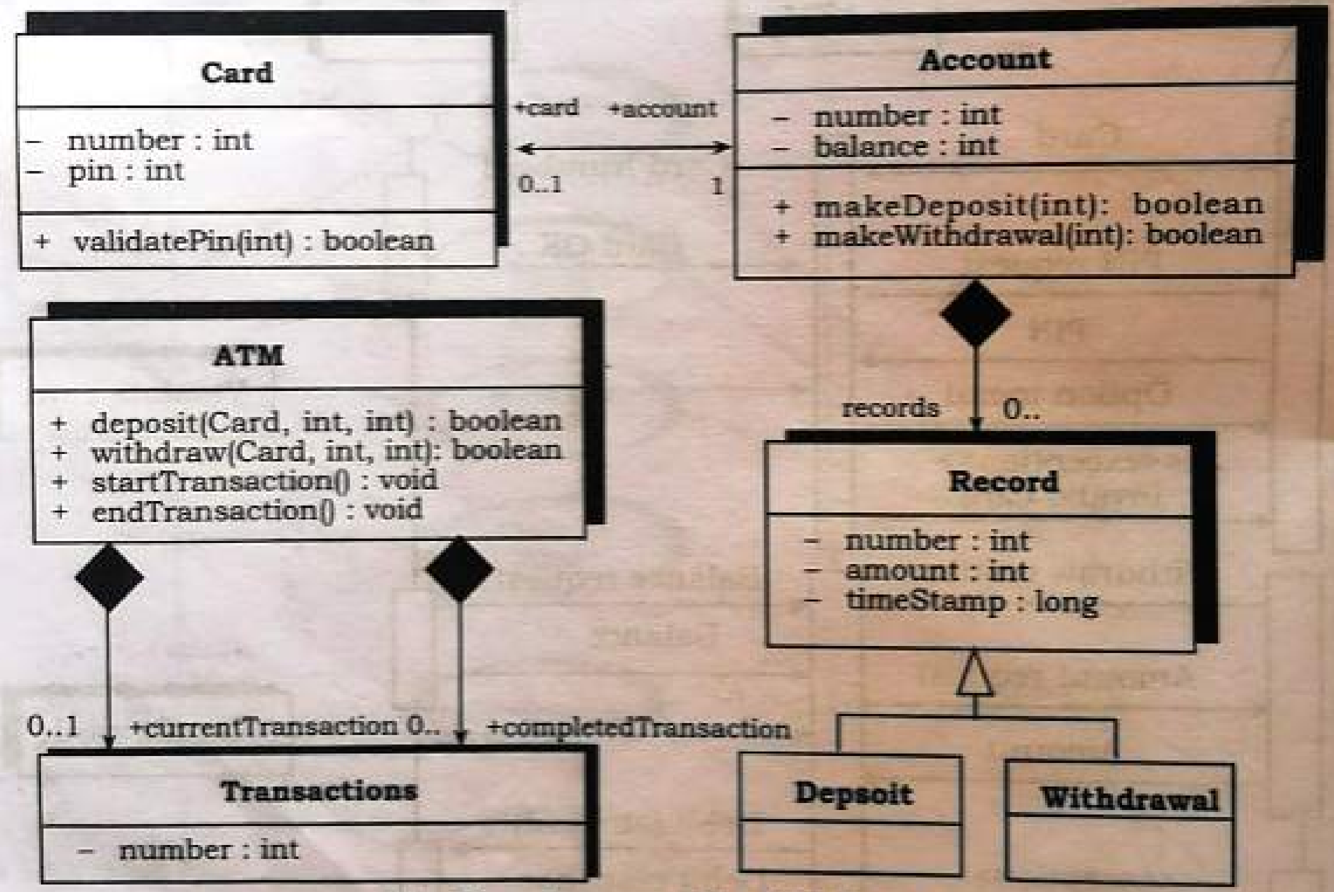

Fig: Class diagram of the ATM System

Generalization is an everyday technique that we use to manage complexity. Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes (animals, cars, houses, etc.) and learn the characteristics of these classes. This allows us to infer that different members of these classes have some common characteristics e.g. squirrels and rats are rodents.

In modelling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change. In object-oriented languages, such as Java, generalization is implemented using the class inheritance mechanisms built into the language.

In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes. The lower-level classes are subclasses inherit the attributes and operations from their superclasses. These lower-level classes then add more specific attributes and operations.

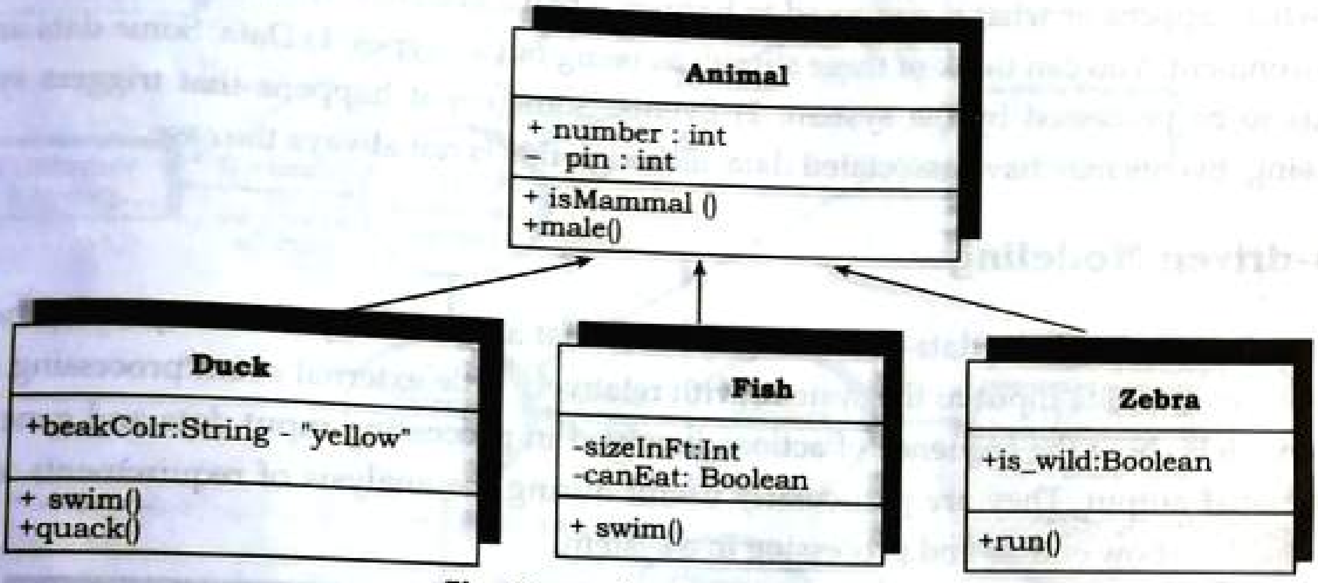

Fig: Generalization in Animals

An aggregation model shows how classes that are collections are composed of other classes. Aggregation models are similar to the part-of relationship in semantic data models.

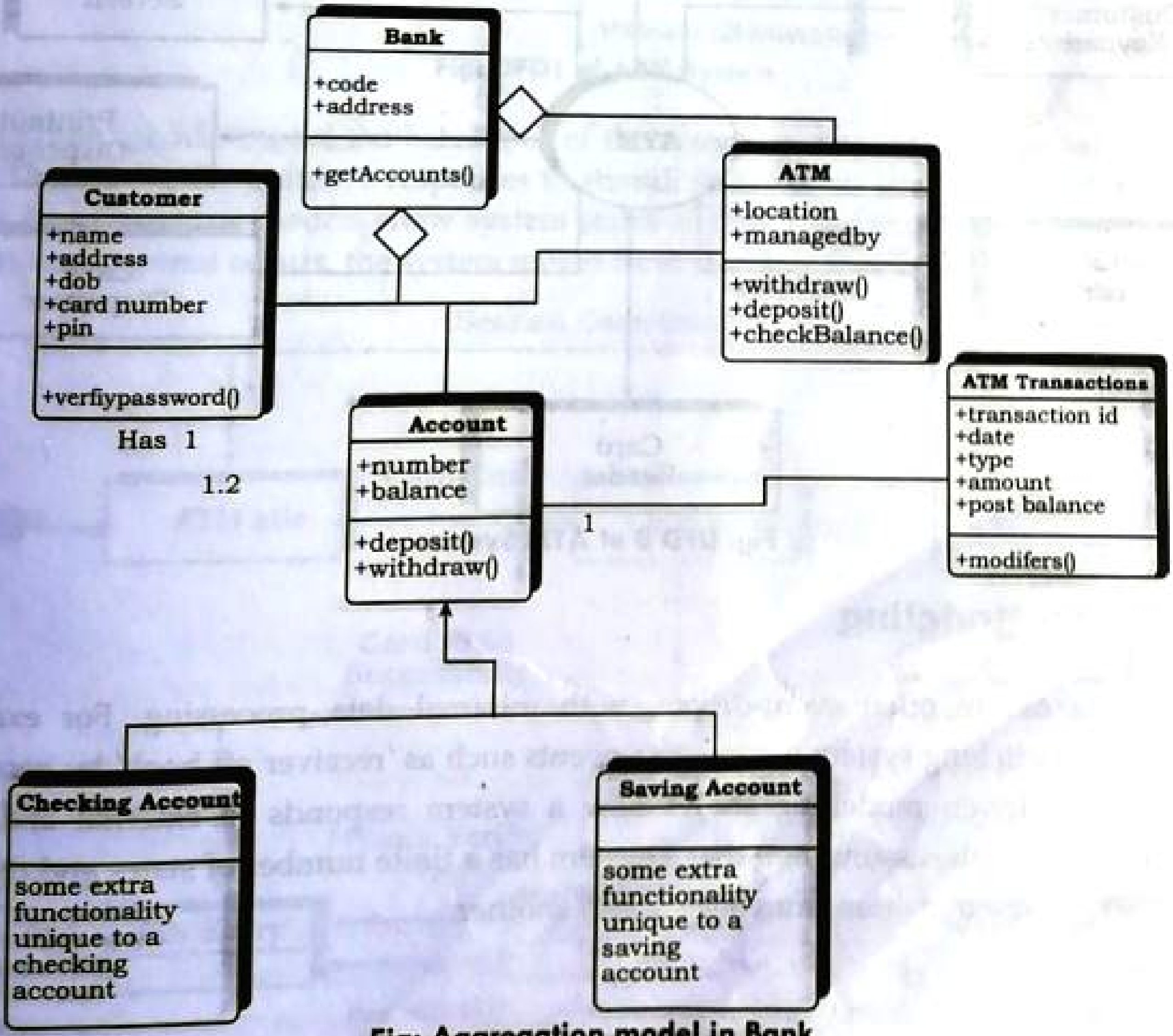

Fig: Aggregation model in Bank

BEHAVIOURAL MODELS

Behavioural models are models of the dynamic behaviour of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment. You can think of these stimuli as being of two types: 1) Data: Some data arrives that has to be processed by the system. 2) Events: Some event happens that triggers system processing. Events may have associated data, although this is not always the case.

Data-driven Modeling

Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing. Data-driven models show the sequence of actions involved in processing input data and generating an associated output. They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system.

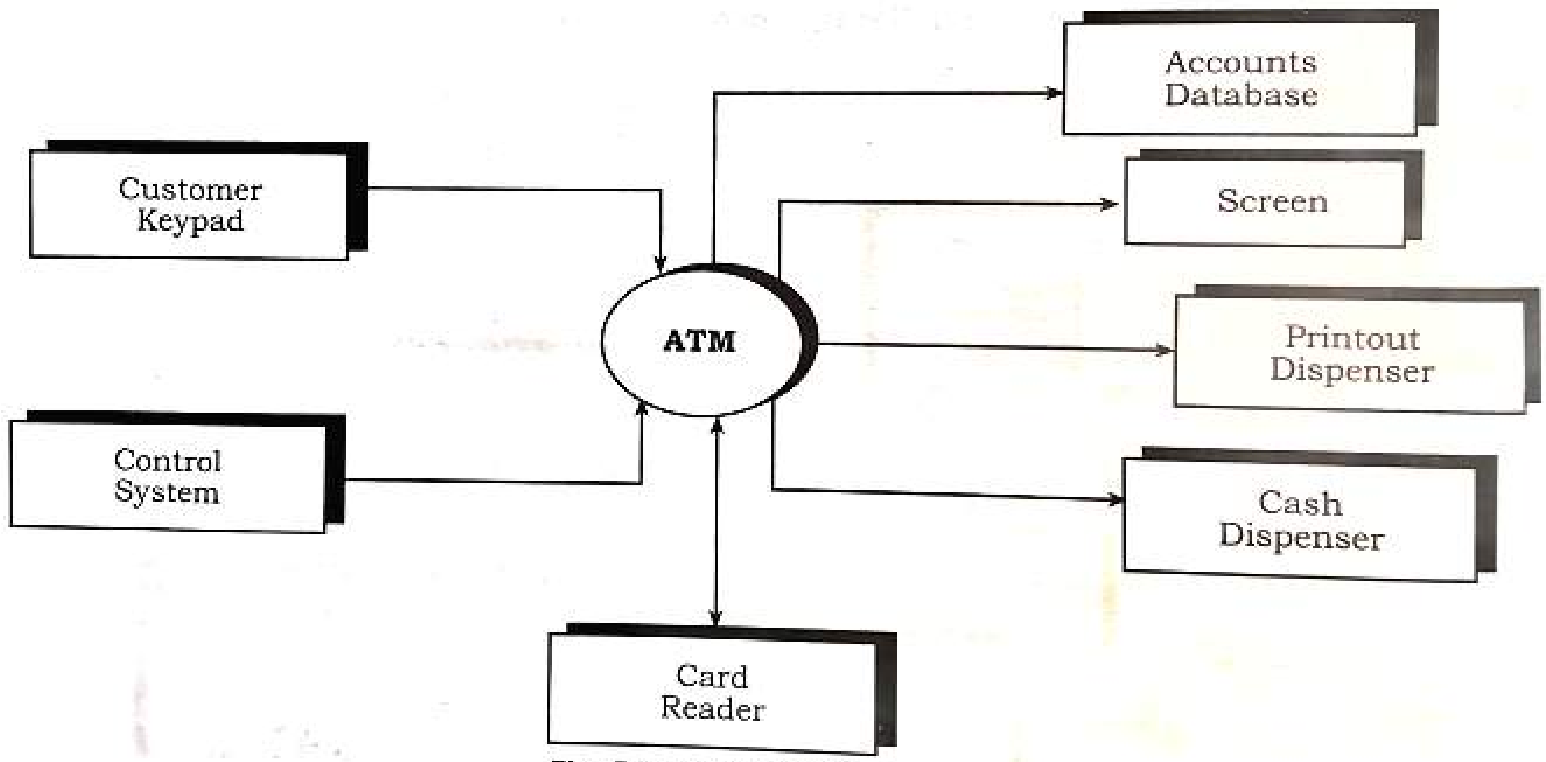

Fig: DFD 0 of ATM System

Event-driven Modeling

Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as 'receiver off hook' by generating a dial tone. Event-driven modelling shows how a system responds to external and internal events. It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another.

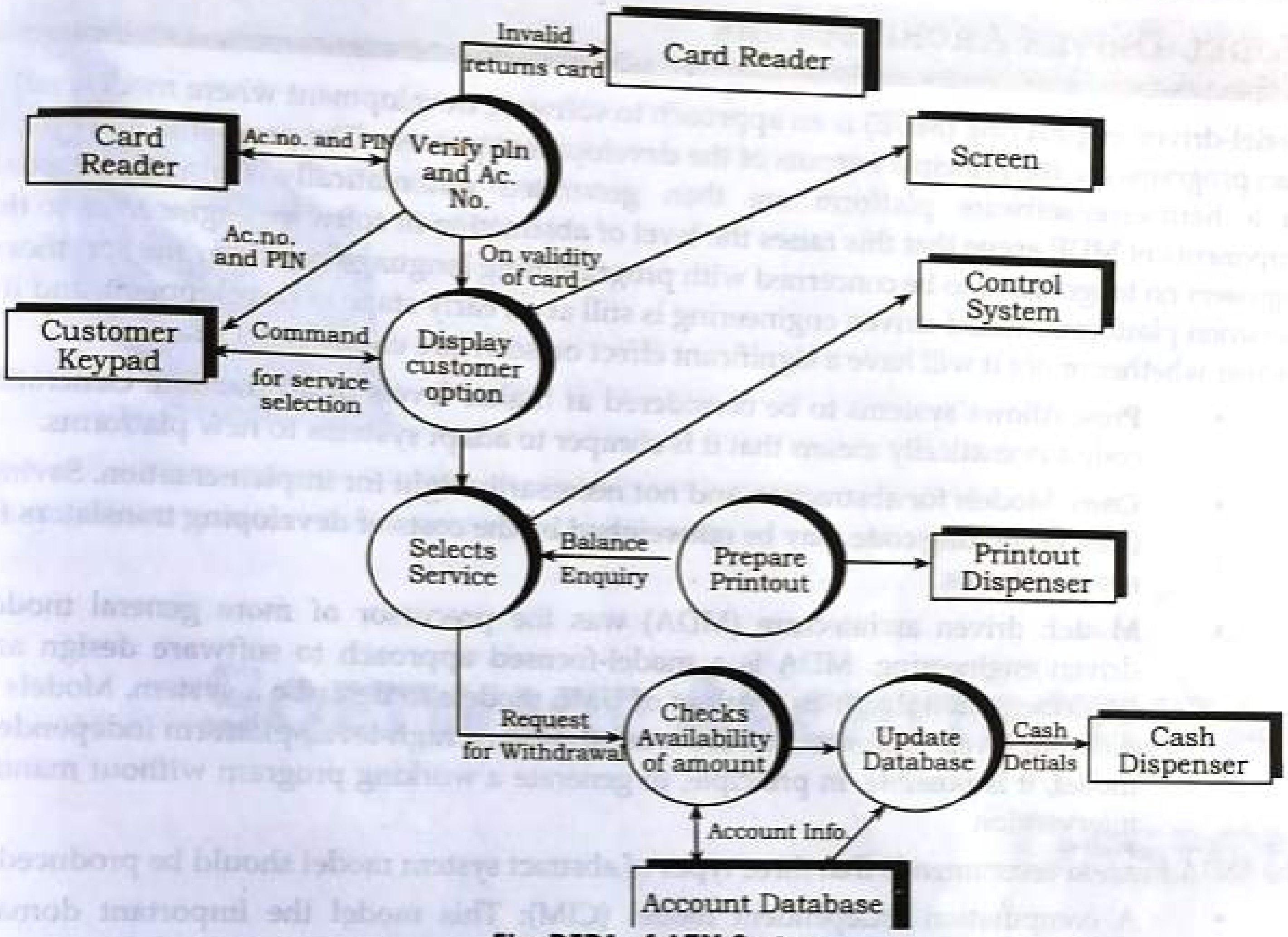

Fig: DFD1 of ATM System

State machine models model the behaviour of the system in response to external and internal events. They show the system's responses to stimuli so are often used for modelling real-time systems. State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.

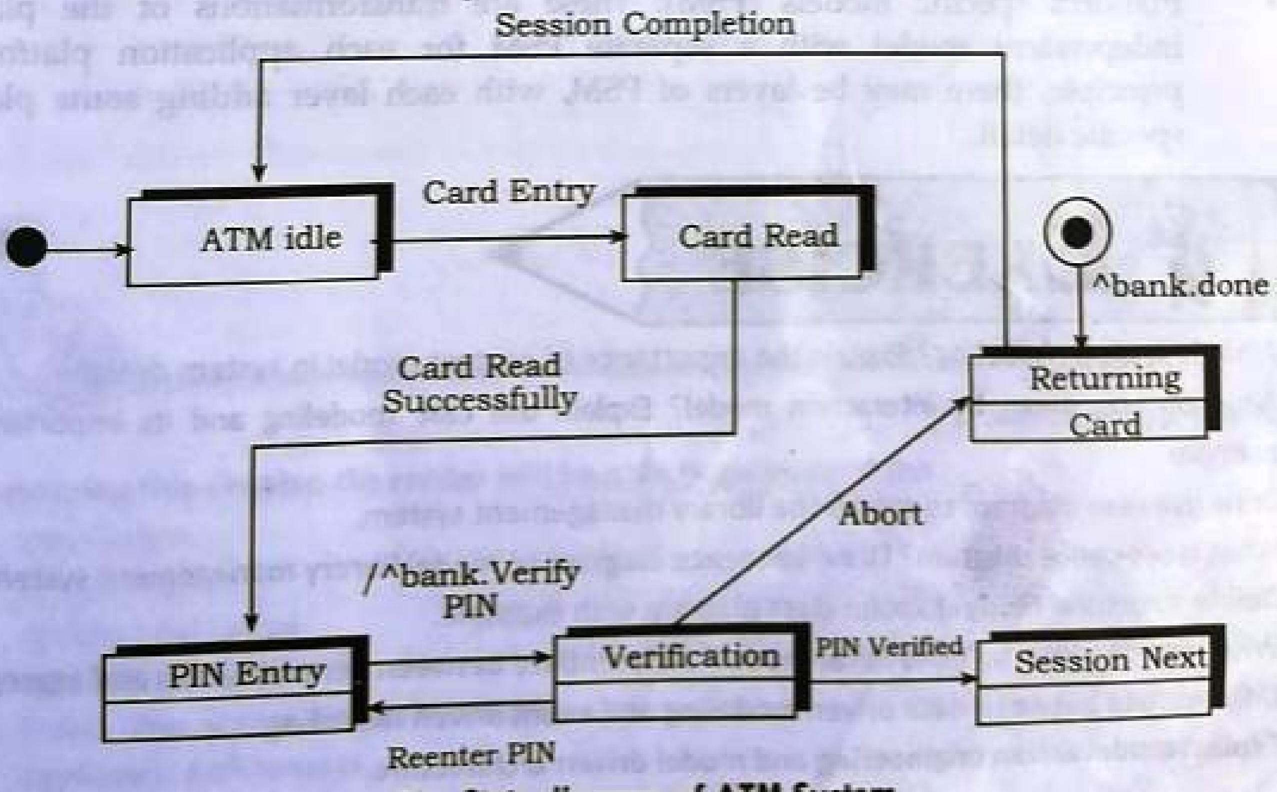

Fig: State diagram of ATM System

MODEL-DRIVEN ARCHITECTURE

Model-driven engineering (MDE) is an approach to software development where models rather than programs are the principal outputs of the development process. The programs that execute on a hardware/software platform are then generated automatically from the models. Proponents of MDE argue that this raises the level of abstraction in software engineering so that engineers no longer have to be concerned with programming language details or the specifics of execution platforms. Model-driven engineering is still at an early stage of development, and it is unclear whether or not it will have a significant effect on software engineering practice.

- **Pros:** Allows systems to be considered at higher levels of abstraction. Generating code automatically means that it is cheaper to adapt systems to new platforms.
- **Cons:** Models for abstraction and not necessarily right for implementation. Savings from generating code may be outweighed by the costs of developing translators for new platforms.
- **Model:** driven architecture (MDA) was the precursor of more general model-driven engineering. MDA is a model-focused approach to software design and implementation that uses a subset of UML models to describe a system. Models at different levels of abstraction are created. From a high-level, platform independent model, it is possible, in principle, to generate a working program without manual intervention.

The MDA method recommends that three types of abstract system model should be produced:

- A computation independent model (CIM): This model the important domain abstractions used in a system. CIMs are sometimes called domain models.
- A platform independent model (PIM): These model the operation of the system without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events.
- Platform specific models (PSM): These are transformations of the platform-independent model with a separate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail.

EXERCISE

1. What is system modeling? Explain the importance of context model in system design.
2. What do you mean by interaction model? Explain use case modeling and its importance with example.
3. Draw use case diagram to model the library management system.
4. What is sequence diagram? Draw sequence diagram to model library management system.
5. Define structural mode. Explain class diagram with example.
6. What do you mean by behavioral model? Differentiate between generalization and aggregation.
7. Differentiate between data driven modeling and event driven modeling.
8. Explain model driven engineering and model driven architecture.