Unit-4 System Modeling

System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the **Unified Modeling Language** (UML). System modeling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

Existing and planned system models

- 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.

Models can explain the System from different perspectives:

- 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.

Five types of UML diagrams that are the most useful for system modeling

- **Activity diagrams**, which show the activities involved in a process or in data processing.
- **Use case diagrams,** which show the interactions between a system and its environment.
- **Sequence diagrams**, which show interactions between actors and the system and between system components.
- **Class diagrams**, which show the object classes in the system and the associations between these classes.
- **State diagrams**, which show how the system reacts to internal and external events.

Context models

Context models are simple communication tools used to depict the context of a business, a system, or a process. The context is the environment in which the object of our interest exists. Context models capture how the central object interacts with its environment, be it exchanging data, physical objects, or funds. 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.

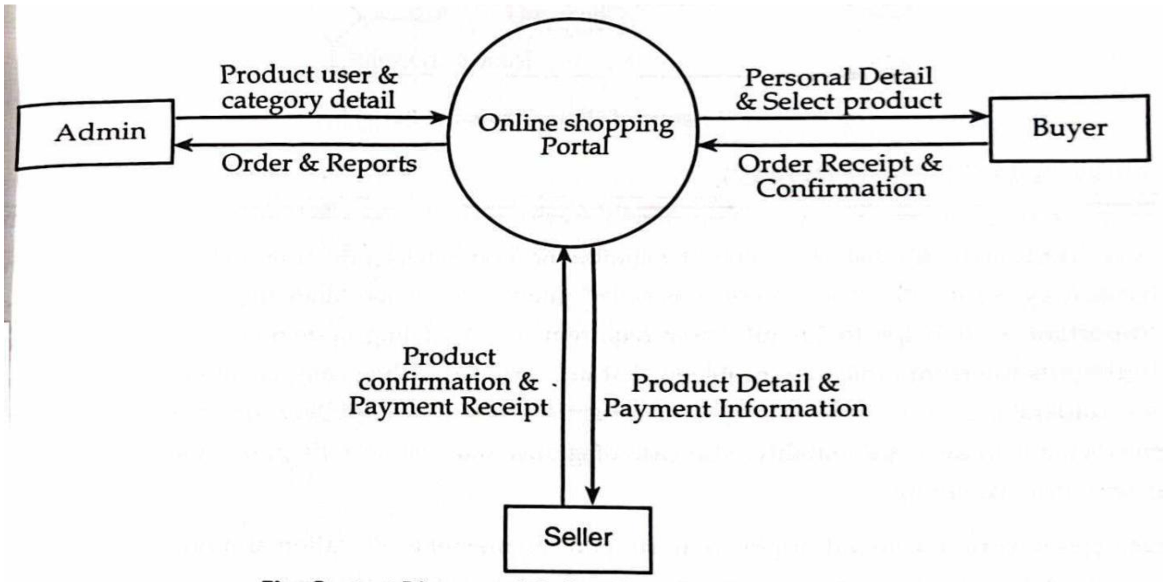
Admin
Product user & category detail
Order & Reports
Online shopping Portal
Personal Detail & Select product
Order Receipt & Confirmation
Buyer
Product confirmation & Payment Receipt
Product Detail & Payment Information
Seller

Fig: Context Diagram of Online Shopping Portal System

Context diagrams are an excellent starting point for requirements planning and analysis. While user journeys are a good approach when we need to take a user perspective, it will not give us the full picture. As each organization interacts with its environment, industry, and value chain elements, we need to see it in the context to have a full picture.

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.

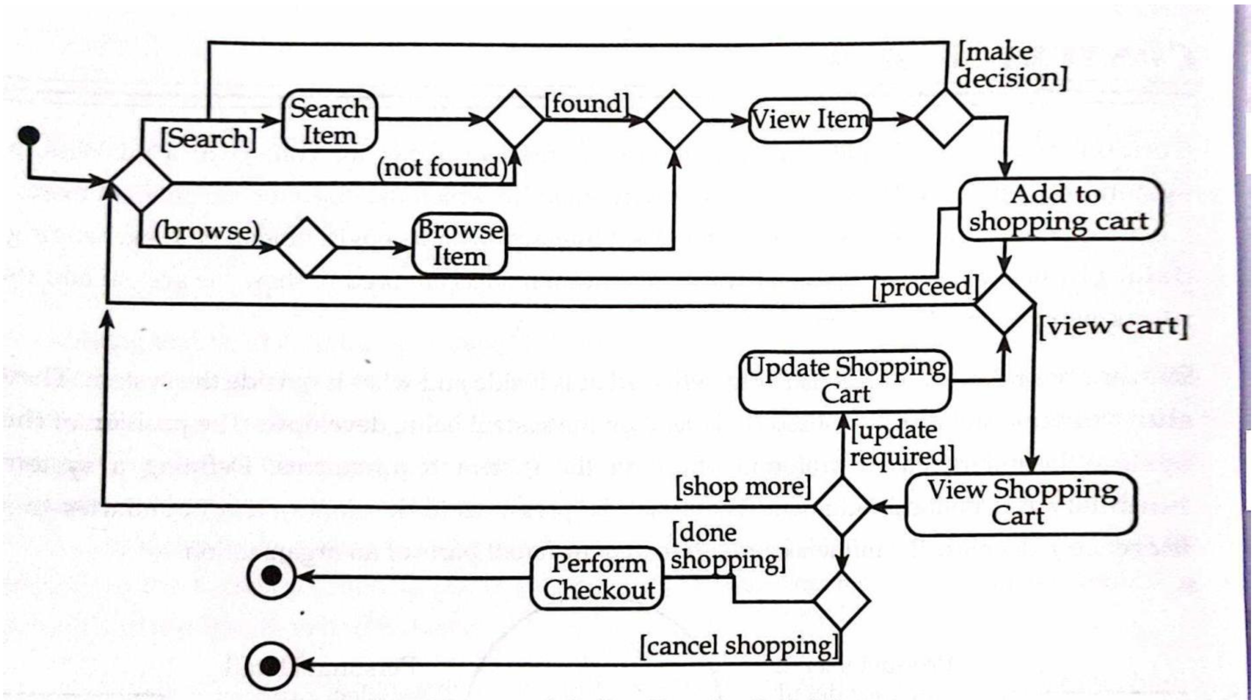

Fig: Activity Diagram of Online Shopping Portal System

Interaction Models

A conceptual model that represents the communication between the user and the Information System by means of a user interface is called interaction model. Modeling user interaction important as it helps to identify user requirements. Modeling system-to-system interaction highlights the communication problems that may arise. Modeling 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 Modeling.

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.

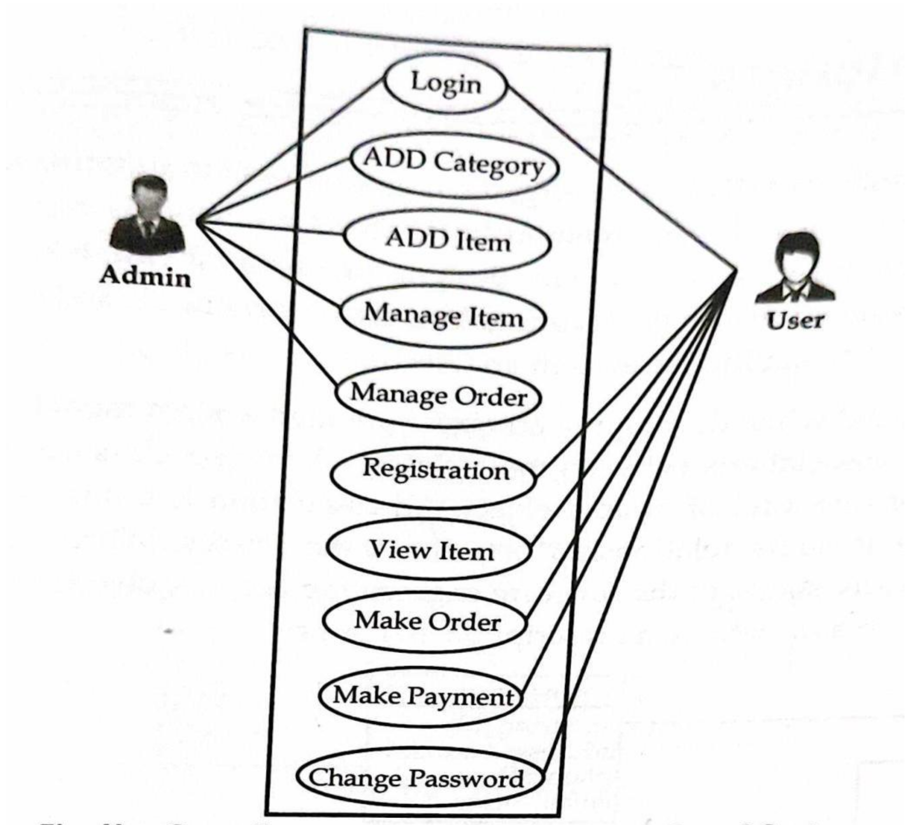

Fig: Use Case diagram for Online Shopping Portal System

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.

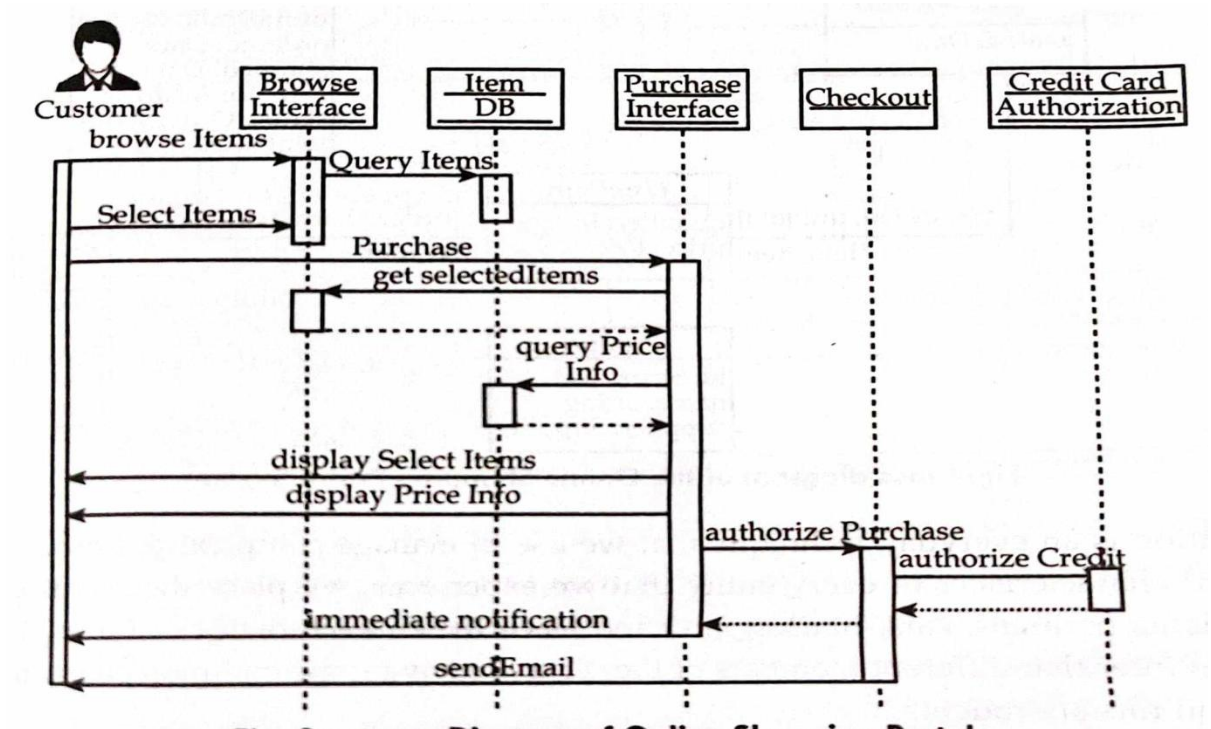

Fig: Sequence Diagram of Online Shopping Portal

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.

A Class diagram is used to model the static design view of the system. A class diagram shows the static aspect of a system: classes, their internal structure and the relationship in which they participate. In UML, a class is represented by a rectangle with **three compartments** separated by horizontal lines. The class name appears in the top compartment, the list of attributes in the middle compartment, and the list of operations in the bottom compartment of the box.

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.

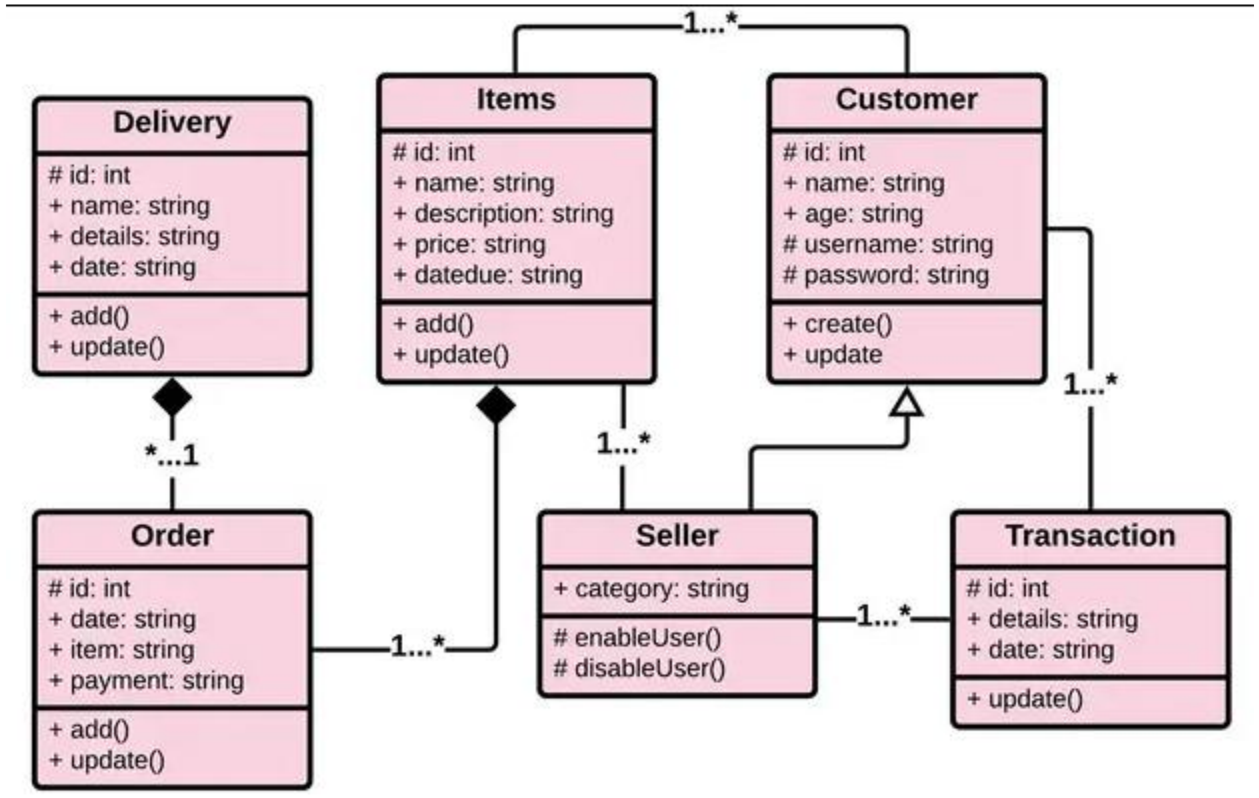

Fig: Class diagram of Online Shopping System

Behavioral Models

Behavioral models are models of the dynamic behavior 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:

▪ **Data:** Some data arrives that has to be processed by the system.

▪ **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.

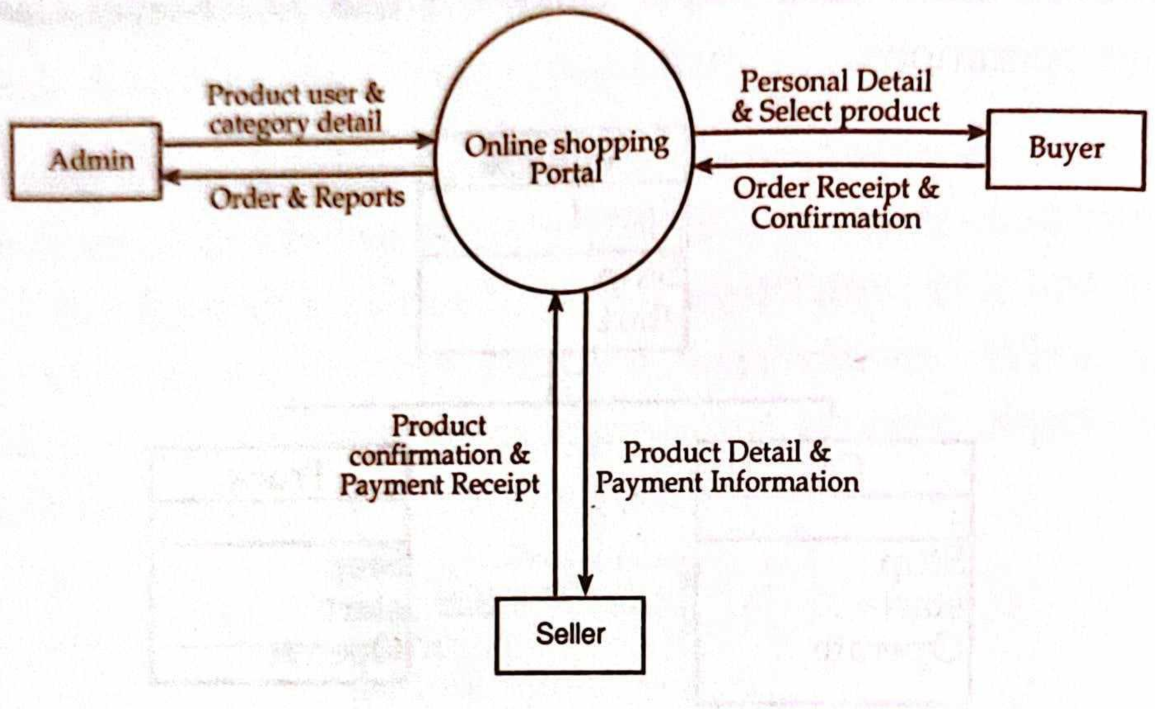
Admin
Product user & category detail
Order & Reports
Online shopping Portal
Personal Detail & Select product
Buyer
Order Receipt & Confirmation
Product confirmation & Payment Receipt
Product Detail & Payment Information
Seller

Fig: DFD 0 of Online Shopping Portal 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 Modeling 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.

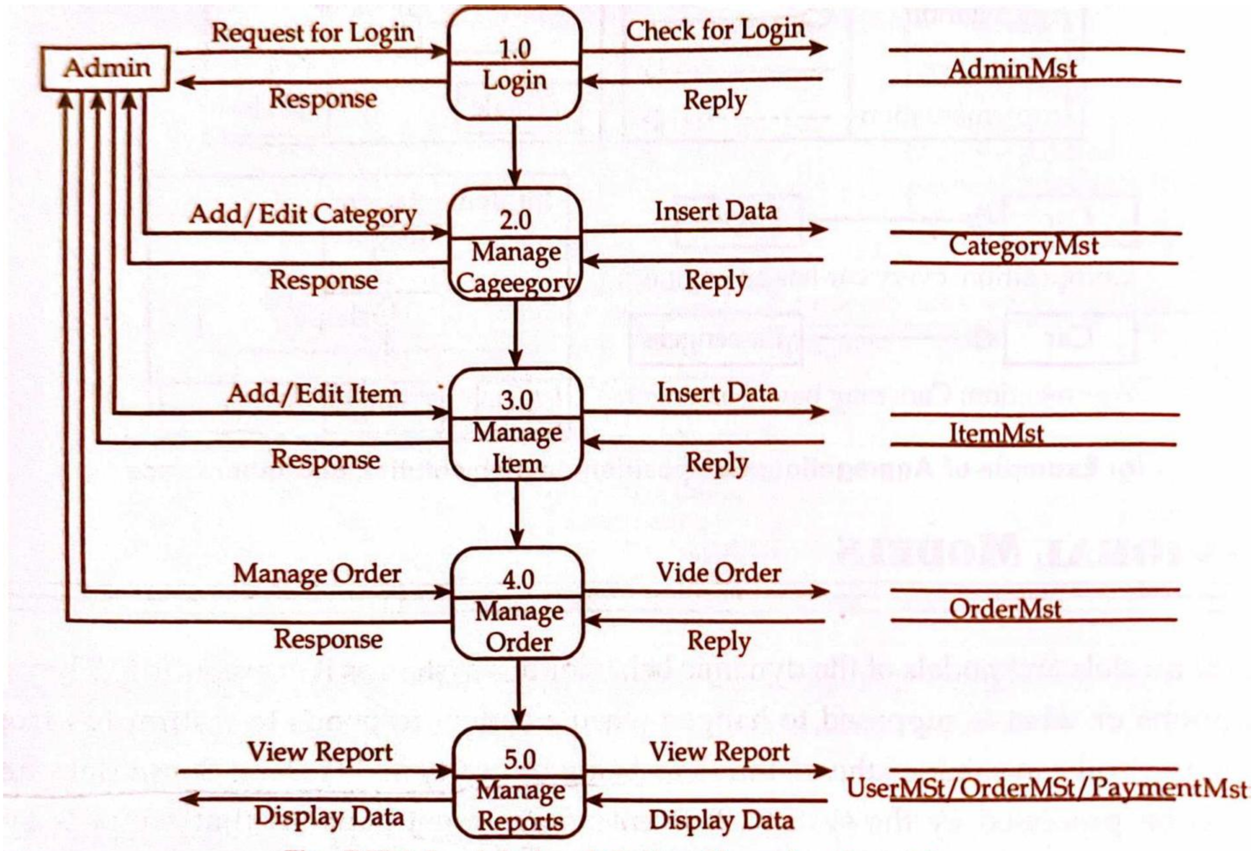

Fig: DFD1 for Admin of Online Shopping Portal System

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:

1. **A computation independent model (CIM):** This model the important domain abstractions used in a system. CIMS are sometimes called domain models.
2. **A platform independent model (PIM):** This 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.
3. **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.