System Analysis and Design

Eighth Edition

Alan Dennis, Barbara Wixom, Roberta M. Roth

Chapter 6

Moving into Design

Objectives

- Explain the initial transition from analysis to design.
- Create a system specification. ✓
  - Already in the process of doing this as part of analysis.
- Describe three ways to acquire a system: custom, packaged, and outsourced alternatives.
- Create an alternative matrix.

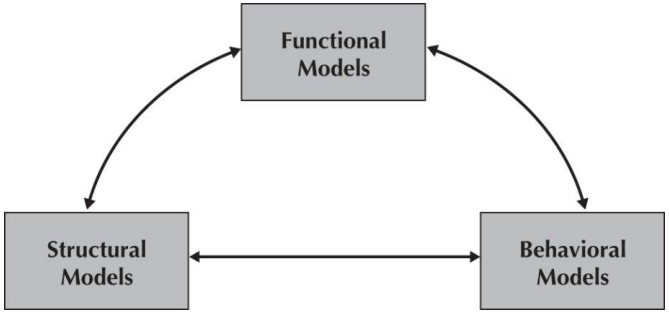
Content Outline

- Design phase
  - Decide *how* to build the system
  - Create ***system requirements*** that describe all technical details for building the system
    - This process was already started as part of the efforts we have done to date.
    - We created a draft Software Requirements Specification\
- Verify Analysis Models
- Design Artifacts
  - Update SRS, as agreed to with Stakeholders.
  - Final Deliverables are the Design Model and the Final SRS
  - Conveys exactly what system the development team will implement during the implementation phase

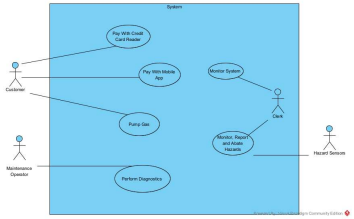
Transition from Requirements to Design

- In ***systems analysis*** we figure out…
  - What the business needs
- In ***system design*** we figure out…
  - How to build the system that fulfills those needs
- All the “logical” work from systems analysis is converted to the “physical”

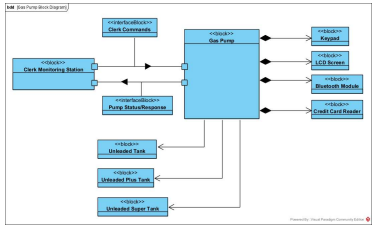
Models Thus Far



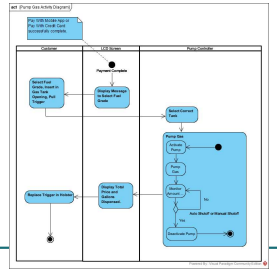
- **Functional Model** – Describe System to be built.
  - User Requirements Gathering Process
  - Use Case Cases
  - Use Case Diagram
  - Use Case Template Description



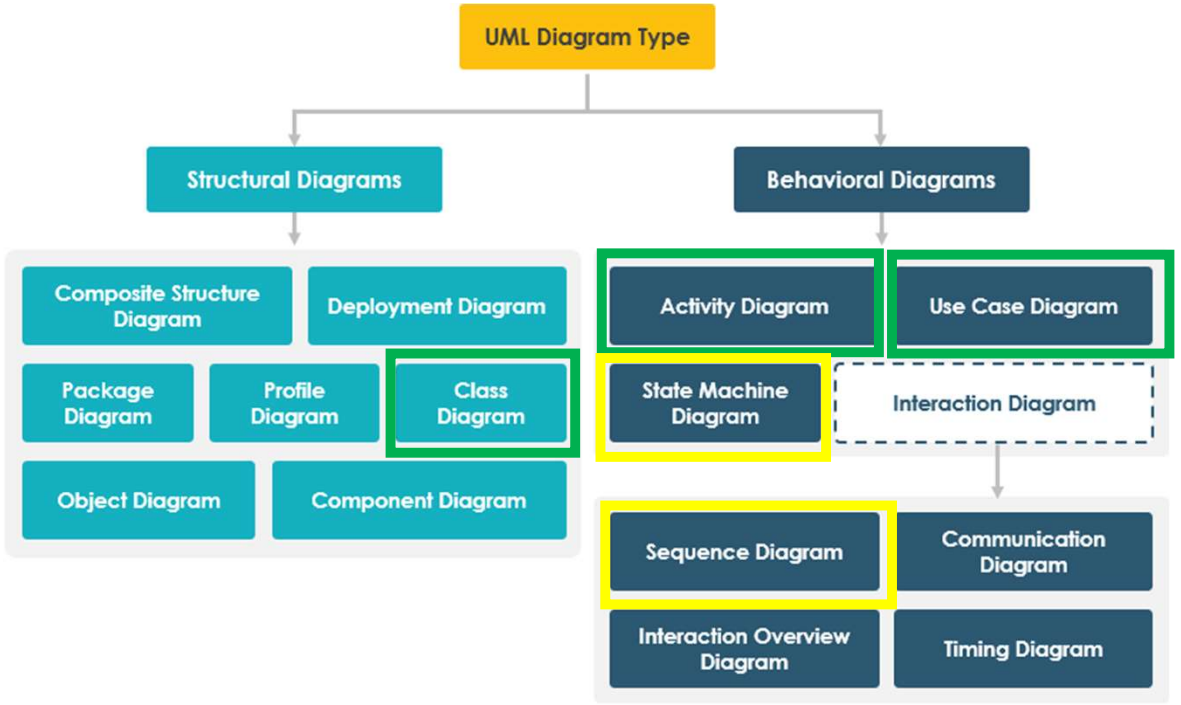
- **Structural Model** – Describe Overall Structure That Helps to Understand The System Context
  - Classes/Blocks
  - System Block Diagram
    - Class Diagram
    - Block Definition Diagram



- Behavioral Model – Describe User Interactions of the System to Model Requirements
  - Use Case Scenarios
  - Activity Diagrams



UML Diagram Types



UML Diagram Overview

Created

Will Create

Spring 2012

EECS810 – Principles of Software Engineering

Bharath Padmanabhan

UNIFIED MODELING LANGUAGE (UML) OVERVIEW

Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. It offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.

Table of Contents

Introduction

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering. UML includes a set of graphic notation techniques to create visual models of object-oriented software systems. UML combines techniques from data modeling, business modeling, object modeling, and component modeling and can be used throughout the software development life-cycle and across different implementation technologies.

Modeling

There is a difference between a UML model and the set of diagrams of a system. A diagram is a partial graphic representation of a system's model. The model also contains documentation that drives the model elements and diagrams (such as written use cases).

UML diagrams represent two different views of a system model:

Static (or structural) view

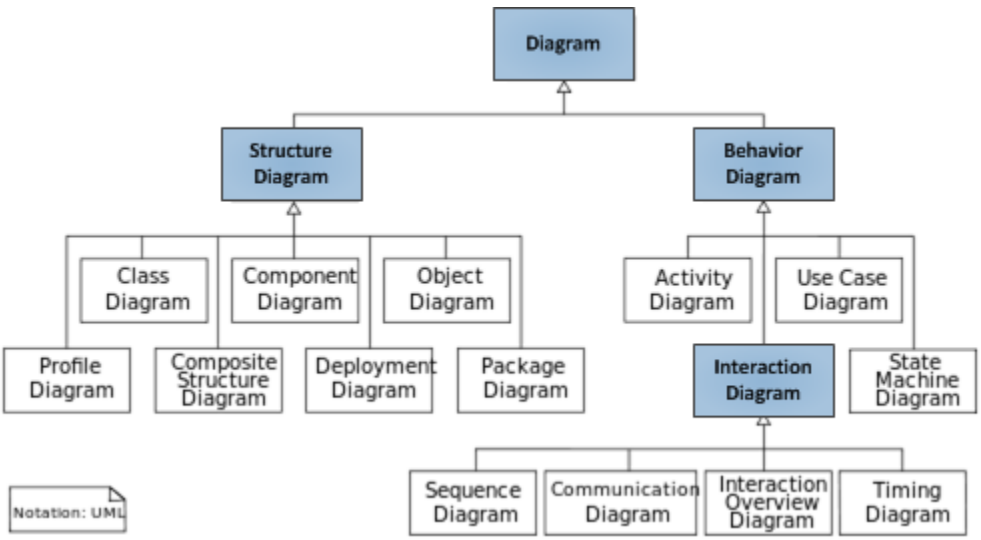
This view emphasizes the static structure of the system using objects, attributes, operations, and relationships. Ex: Class diagram, Composite Structure diagram.

Dynamic (or behavioral) view

This view emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. Ex: Sequence diagram, Activity diagram, State Machine diagram.

Diagrams Overview

UML 2.2 has 14 types of diagrams divided into multiple categories as shown in the figure below.

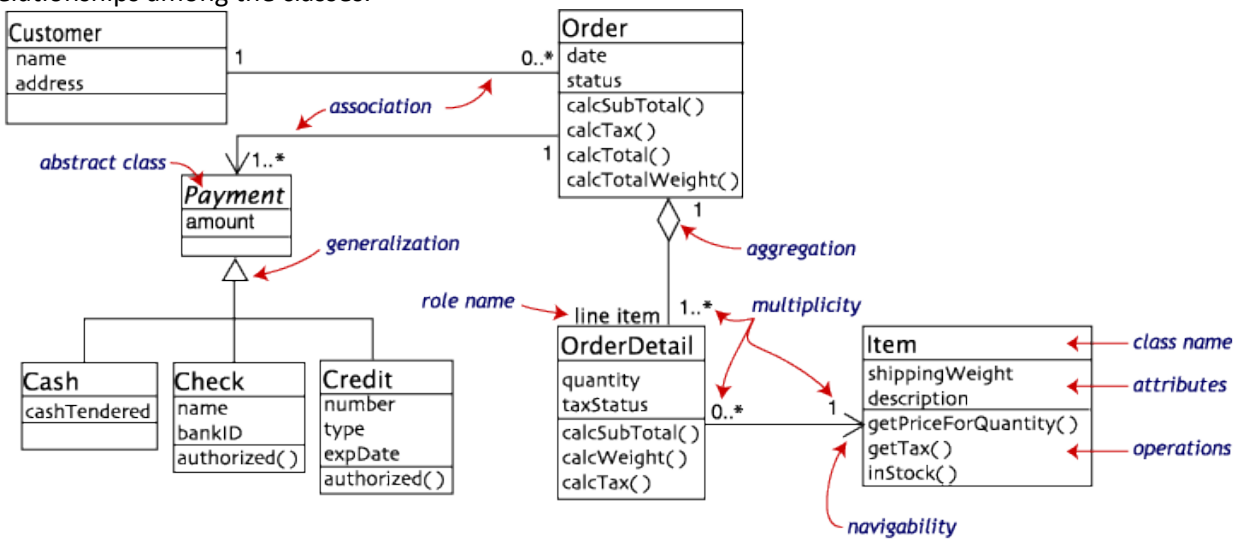


Structure Diagrams

These diagrams emphasize the things that must be present in the system being modeled. Since they represent the structure, they are used extensively in documenting the software architecture of software systems.

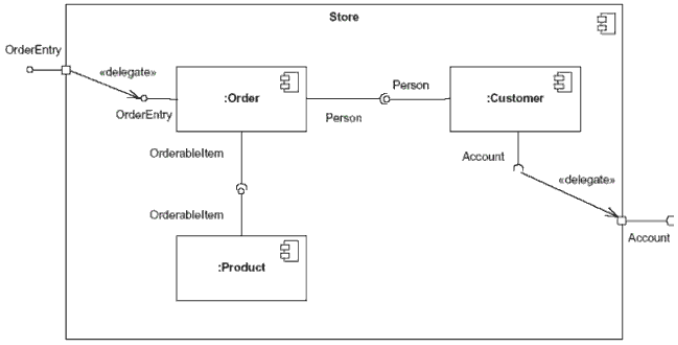
1. Class Diagram

Describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes.



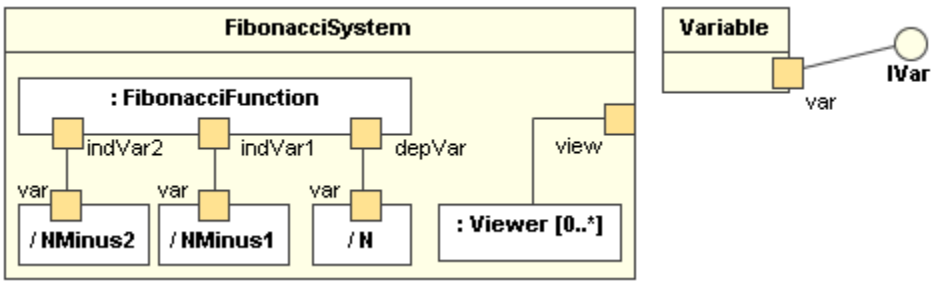
2. Component Diagram

Describes how a software system is split-up into components and shows the dependencies among these components.



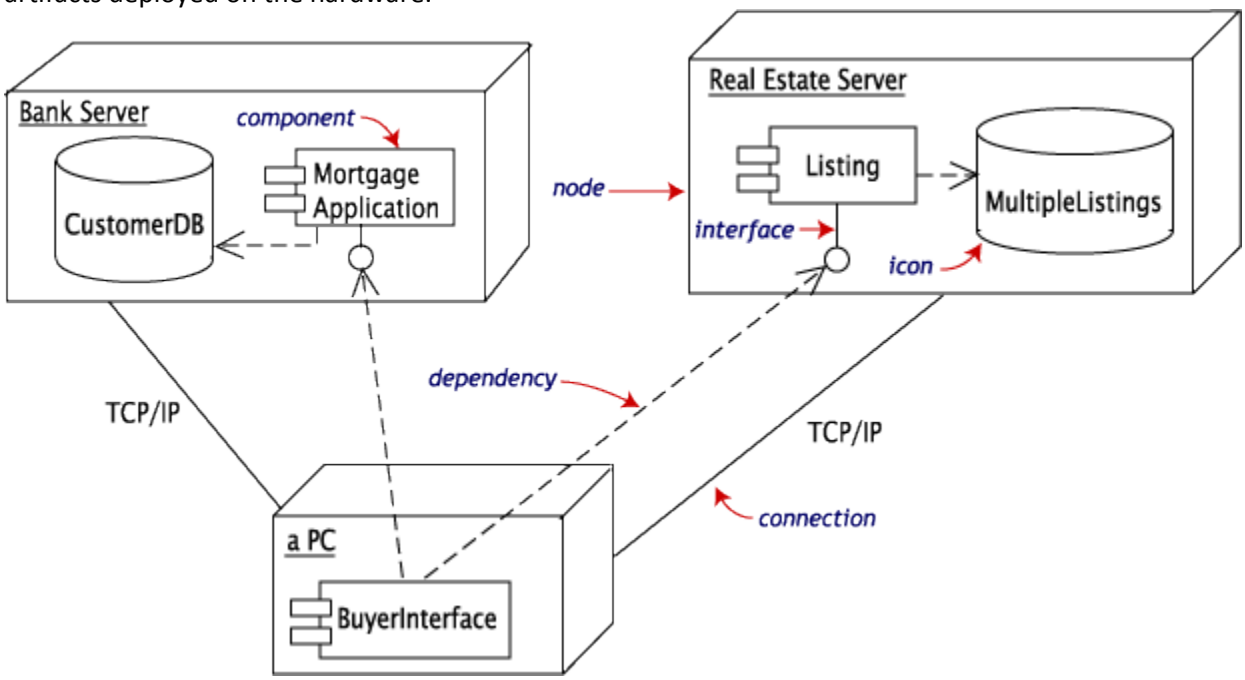
3. Composite Structure Diagram

Describes the internal structure of a class and the collaborations that this structure makes possible.



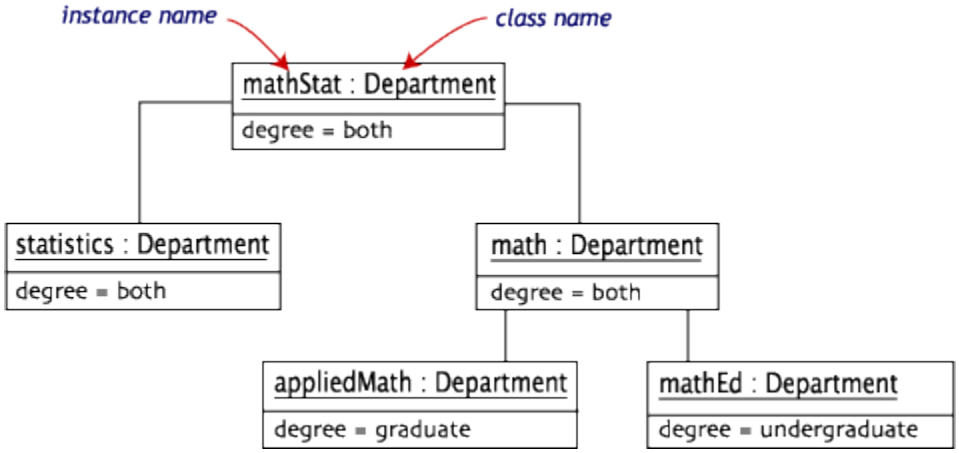
4. Deployment Diagram

Describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.



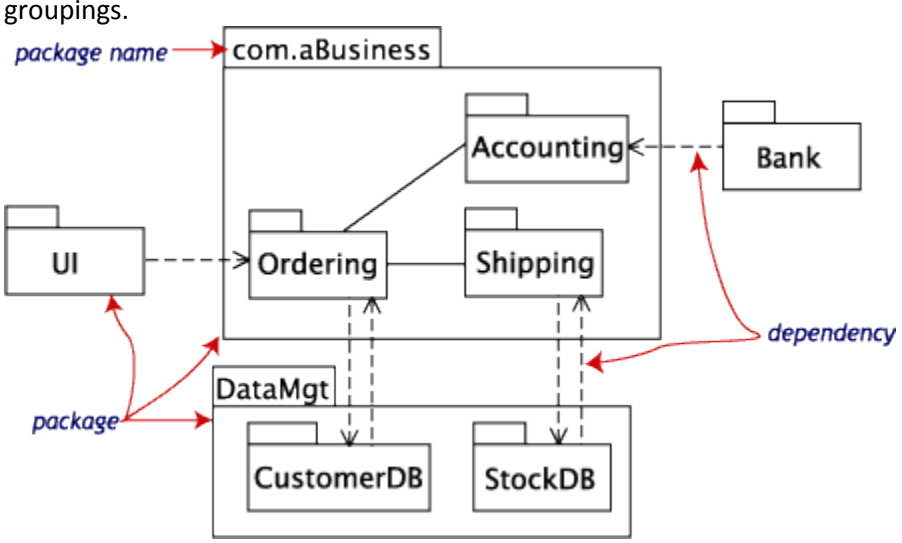
5. Object Diagram

Shows a complete or partial view of the structure of an example modeled system at a specific time.



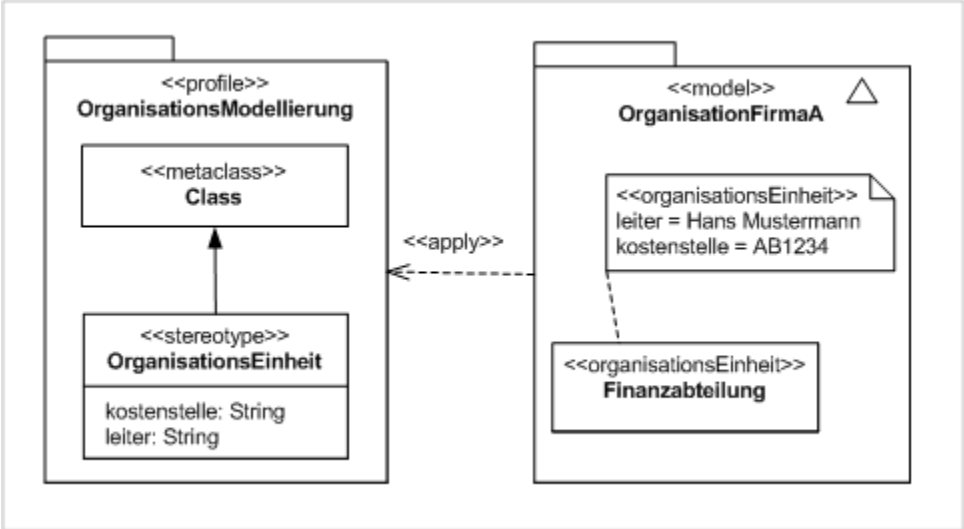
6. Package Diagram

Describes how a system is split-up into logical groupings by showing the dependencies among these groupings.



7. Profile Diagram

Operates at the metamodel level to show stereotypes as classes with the <<stereotype>> stereotype, and profiles as packages with the <<profile>> stereotype. The extension relation (solid line with closed, filled arrowhead) indicates what metamodel element a given stereotype is extending.

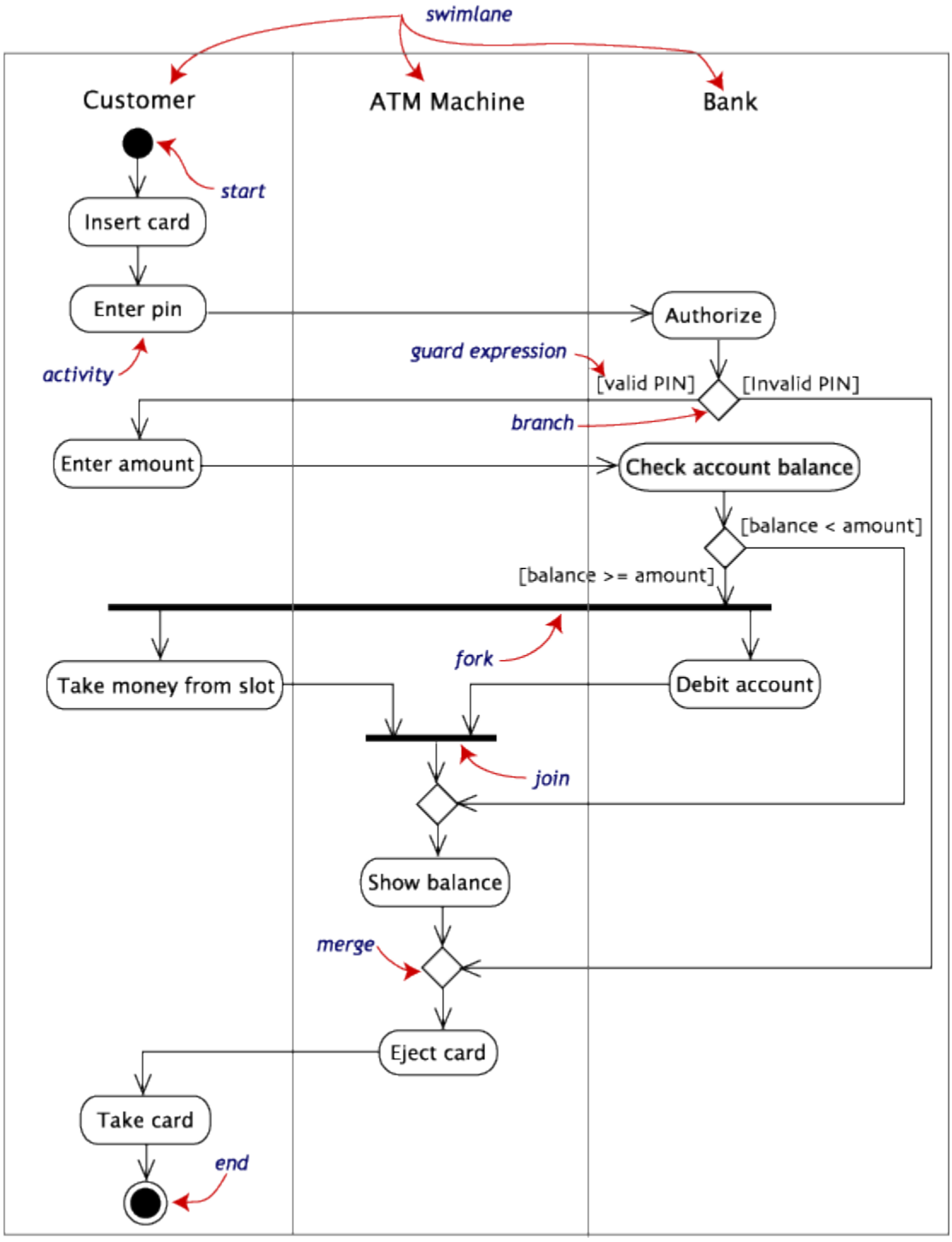


Behavior Diagrams

These diagrams emphasize what must happen in the system being modeled. Since they illustrate the behavior of a system, they are used extensively to describe the functionality of software systems.

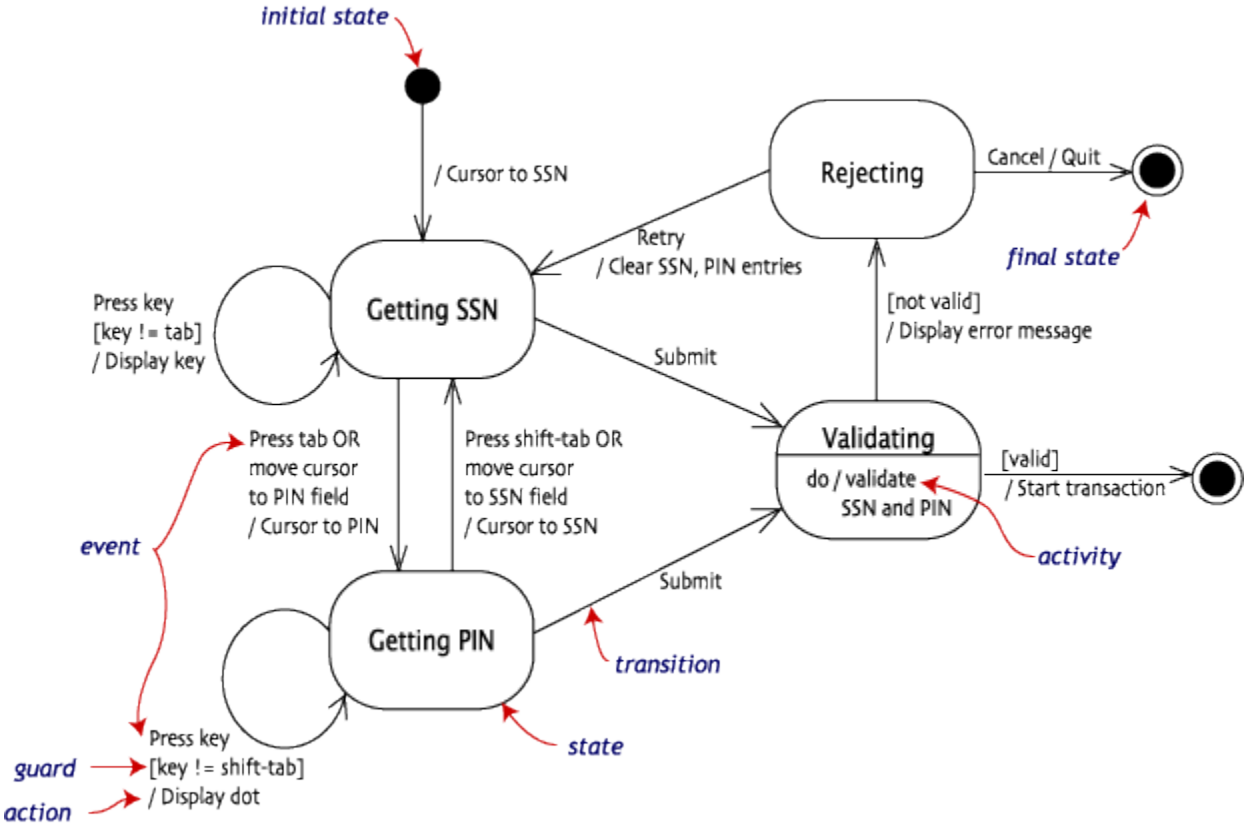
1. Activity Diagram

Describes the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



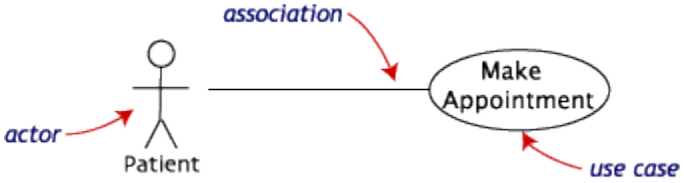
2. State Machine Diagram

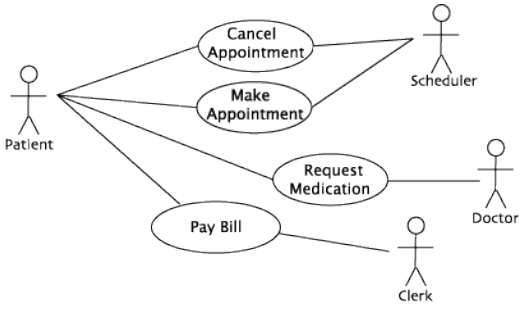
Describes the states and state transitions of the system.



3. Use Case Diagram

Describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.



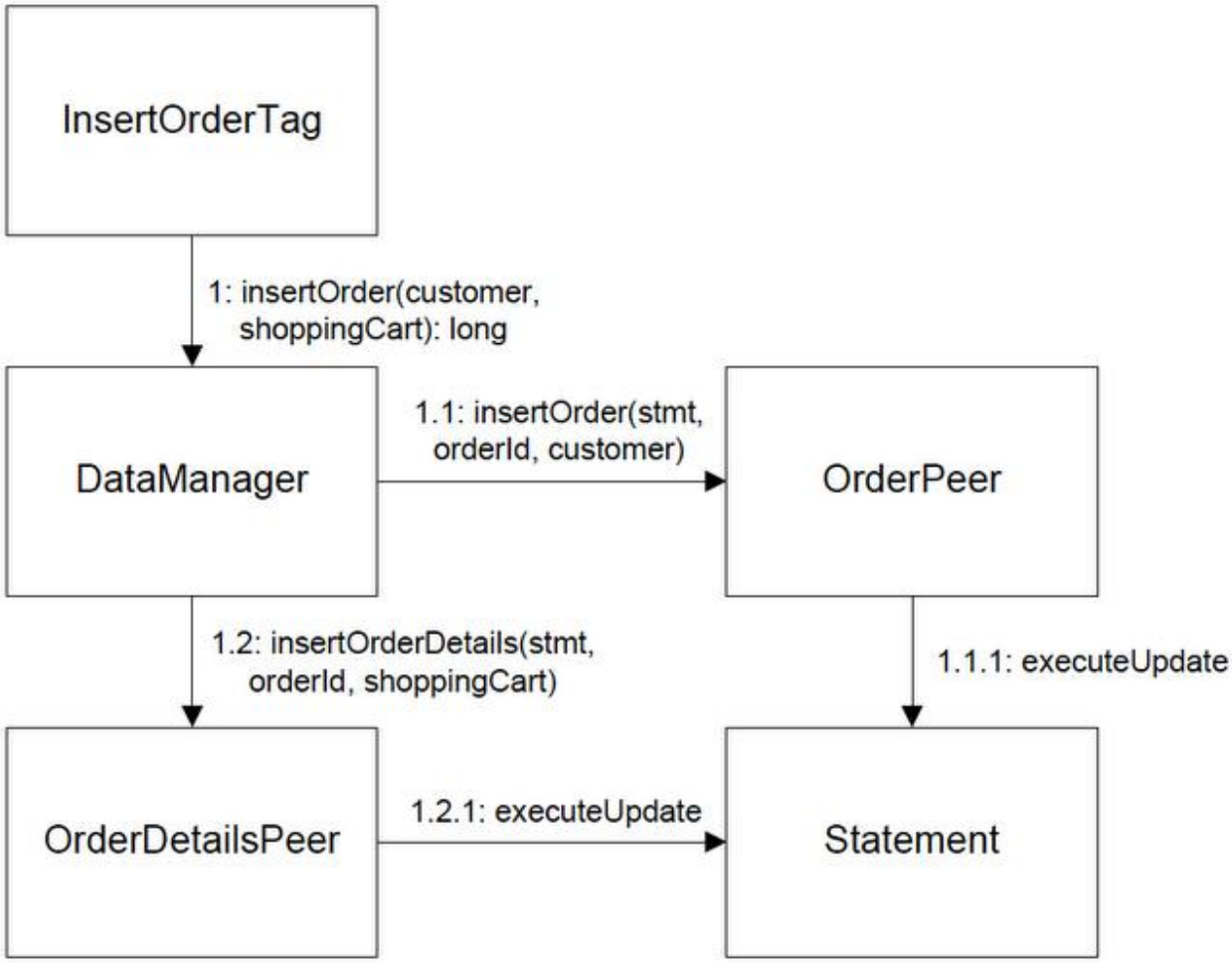


Interaction Diagrams

These diagrams are a subset of behavior diagrams, emphasizing the flow of control and data among the things in the system being modeled.

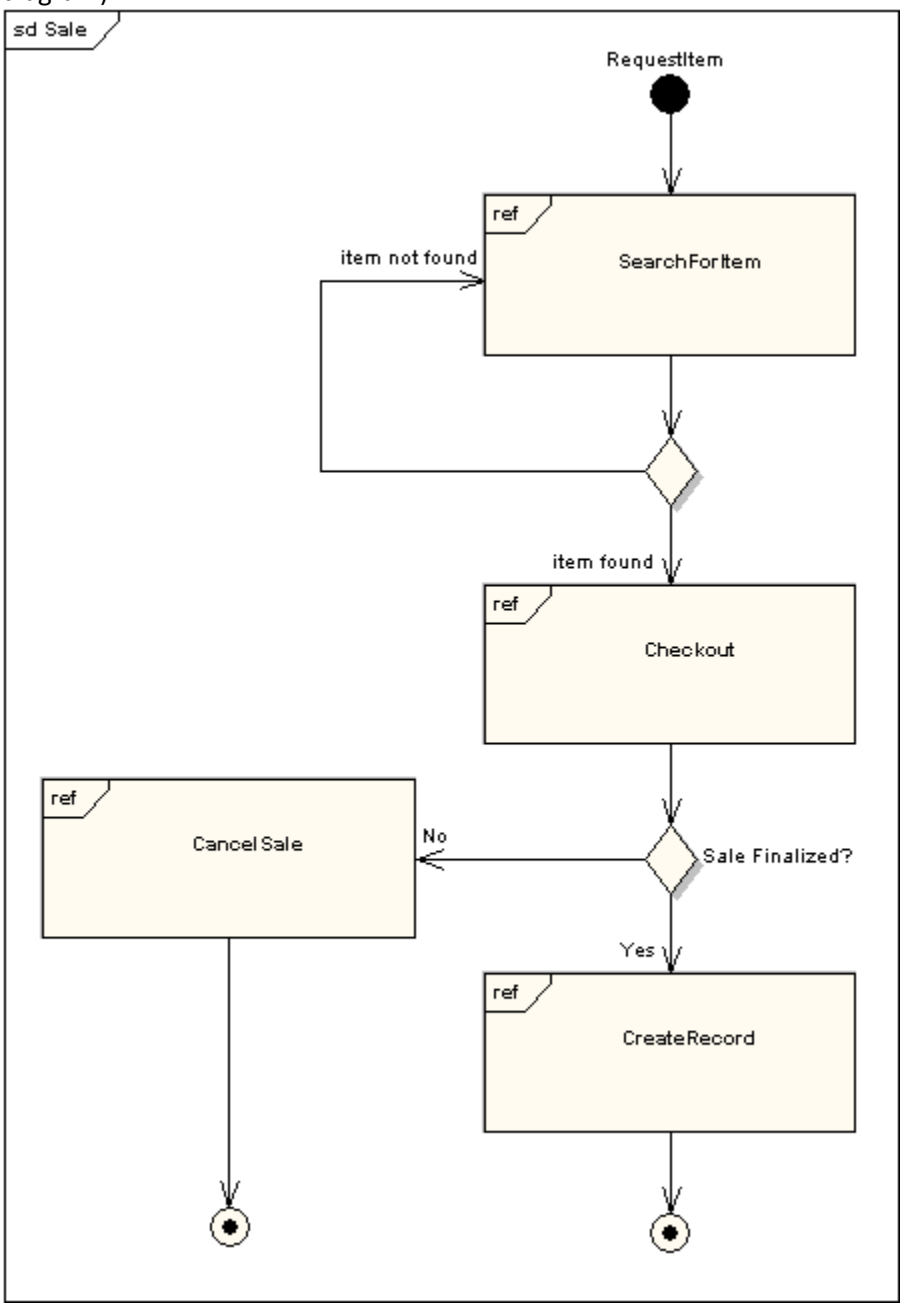
1. Communication Diagram

Shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.



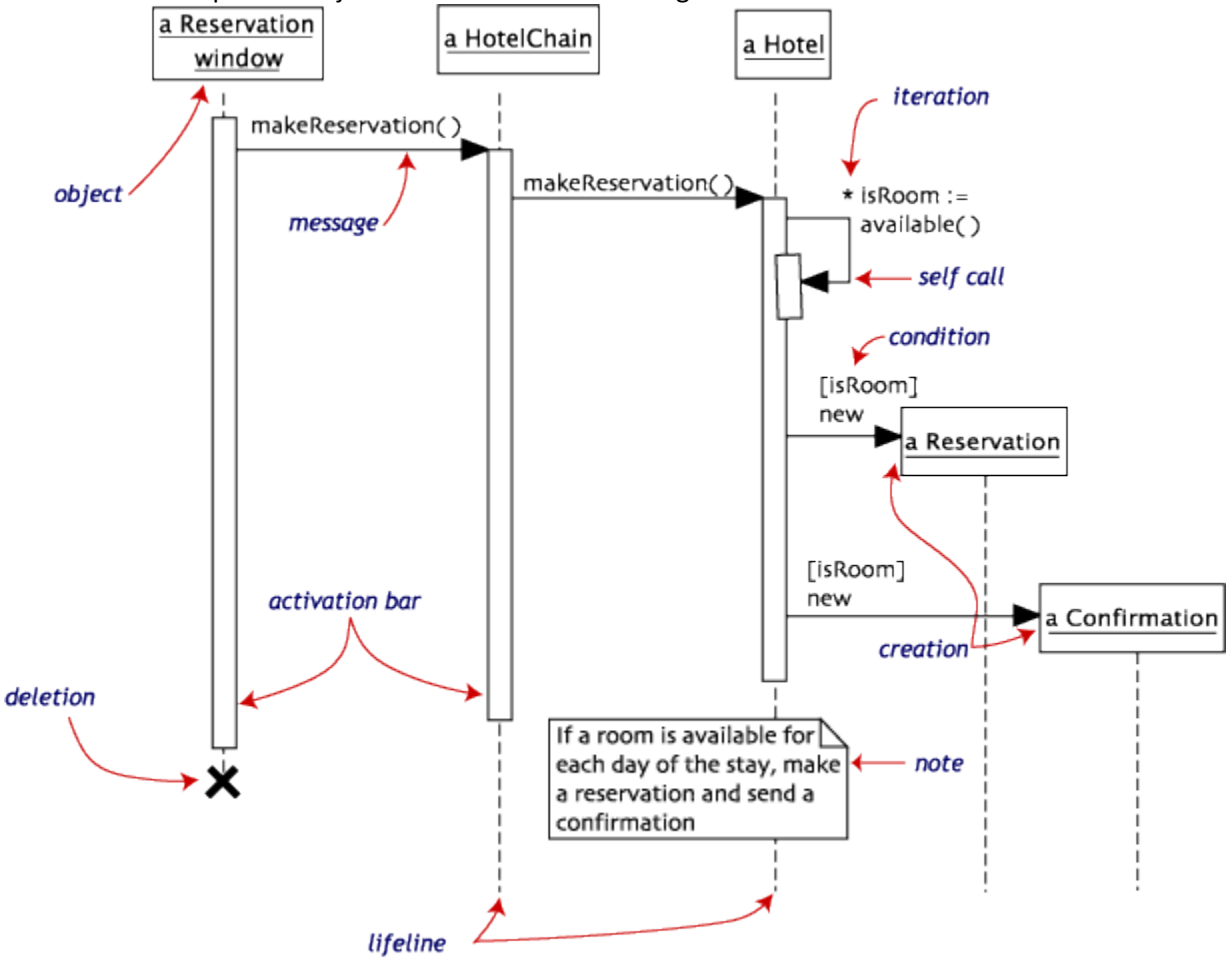
2. Interaction Overview Diagram

Provides an overview in which the nodes represent communication diagrams. They are activity diagrams in which every node, instead of being an activity, is a rectangular frame containing an interaction diagram (i.e., a communication, interaction overview, sequence, or UML timing diagram).



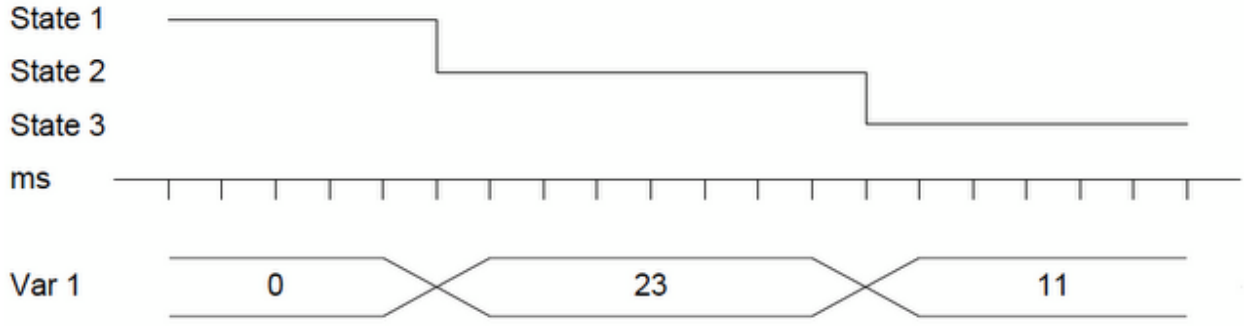
3. Sequence Diagram

Shows how objects communicate with each other in terms of a sequence of messages. Also indicates the lifespans of objects relative to those messages.



4. Timing Diagram

A specific type of interaction diagram where the focus is on timing constraints. Timing diagrams model sequence of events and their effects on states and property values. Time flows along a horizontal axis from left to right. They can be used to show method execution profiling or concurrency scenarios.



References

Unified Modeling Language
http://en.wikipedia.org/wiki/Unified_Modeling_Language

UML basics: The component diagram
http://www.ibm.com/developerworks/rational/library/dec04/bell/

Practical UML: A Hands-On Introduction for Developers
http://wwwis.win.tue.nl/2R690/together/

OO – UML Behavior Diagrams
http://giuliozambon.blogspot.com/2010/09/oo-uml-behavior-diagrams.html

UML 2 Tutorial
http://www.sparxsystems.com/resources/uml2_tutorial/