Modeling Using Use Cases and UML

Modeling

- One approach to systems development is to start by building a model of the existing information system in order to better understand how it works
- Another approach is to start by identifying the requirements of a new system and then building to those requirements, which we will look at later

Modeling

- A model is a representation of something in the real world
  - Model planes, cars, architectural models
- To build this model we need two things
  - A modeling tool
  - Information about the system being modeled

Modeling

- We will use use cases and the UML to model the system
- We will gather information about the system through interviews

Stakeholder interviews

- Can be particularly effective
- Involves informal conversation with a cross-section of people

Interviews – Some initial questions

- Why is the project being done?
- For whom is it being done?
- Who are the affected stakeholders?
- What are their goals for the project?

Interviews – Some initial questions

- How does it fit with the general business objectives?
- Who are the competitors?
- Have there been any previous attempts at this project?
- How will success be measured?

Interviews – Some guidelines

- Converse naturally
- Allow the stakeholder to go off-topic
- Allow for anonymity
- Take a look at the script at
  - http://www.Goodkickoffmeetings.com/2010/04/stakeholder-frontloading

Uses of Use Cases

- Use cases provide a crucial framework for analysis, design, implementation and deployment activities.
- The main reason for use case modeling is to model the system's behavior from a conceptual viewpoint.

Uses of Use Cases

- Use cases provide a framework for
  - discovering business objects,
  - designing user interface,
  - incremental development,
  - user documentation and
  - application help, test case definition and training.

Uses of Use Cases

- Requirements Gathering
  - Use cases provide the base tools for gathering requirements within a meaningful context.
- Requirements Traceability
  - Use cases and their supporting documents are the prime sources for tracing requirements.
- Business Rules
  - Use cases are the framework for gathering business rules.
- System Behavior
  - The external behavior of any open system can be captured effectively through use cases.

Uses of Use Cases

- Object Derivation
  - By launching a cycle of gathering requirements from the use cases, we can arrive at many of the objects that would form the structure of the system.
- Incremental Development
  - By prioritizing use cases and their dependencies, we can build a system incrementally.
- Base for User Interface
  - Use cases describe the basics messages that the actor and the system must exchange to achieve a goal.

Uses of Use Cases

- Test Case Definition
  - Use cases are the conceptual blueprints for functional test cases.
- Base for User Documentation
  - Use cases are built to describe the interaction between a user type and a system.
- Business Process Modeling
  - Use cases can be used to model business processes, prior to, after, or independent from an information system.

Use Case modeling

- A use case is a unit of system behavior
- A use case is a contract that formalizes the interaction between stakeholders and the system
- A use case details the interaction of an actor with a system to accomplish a goal of value to the actor
- Use cases are technology-independent

Use Case modeling

- Use case model is a set of use cases that, together, describe the behavior of a system.
- A use case is a unit of this model.
- Use Case modeling is limited to a system's external behavior
  - Use cases do not model the system from inside.
  - Use cases are not inherently object-oriented.
  - Use cases describe what a system accomplishes, not how.

Components of Use Cases

- A use case is a textual narrative, with four well-defined components:
  - A goal as the successful outcome of the use case,
  - Stakeholders whose interests are affected by the outcome, including actor(s) who interact with the system to achieve the goal,
  - System that provides the required services for the actors, and
  - Step-by-step scenario that guides both the actor(s) and the system towards the finish line.

Goal

- A use case is successful only if its stated goal is completely achieved
- A use case's name is its goal. The name must be active, concise and decisive.
- It is the goal that decides the relevance of activities in a use case.

Stakeholders and Actors

- Stakeholders are those entities whose interests are affected by the success or the failure of the use case.
- An actor is an entity outside the system that interacts with the system to achieve a specific goal.
- A use case must enforce the interests of all stakeholders.

Actor

- Actor is a role that any user who has been given the part can play.
- The goal of the primary actor is specified by the name of the use case.
- Supporting (or secondary) actors support the primary actor in reaching the goal of the use case.
- An actor is identified by a unique name which describes a unique role.

A System

- The system defines the boundaries of a use case
- Two types of systems
  - Real system
    - Grocery store "bricks-and-mortar"
  - Information system
    - Point of Sales System (POS)
- A use case cannot leave a system, but can reach across its boundaries

Purchase Groceries — Real System Scenario

- A customer enters the supermarket. The customer takes a shopping cart or basket and strolls through the supermarket. The customer selects items from the shelves and puts them in the shopping cart or the basket. When finished, the customer brings the items to the cash register. The cashier calculates the total price of the merchandise. The customer pays for the merchandise. The cashier bags the items, issues a receipt to the customer and, if necessary, returns the change. The customer picks up the bags and leaves the supermarket.

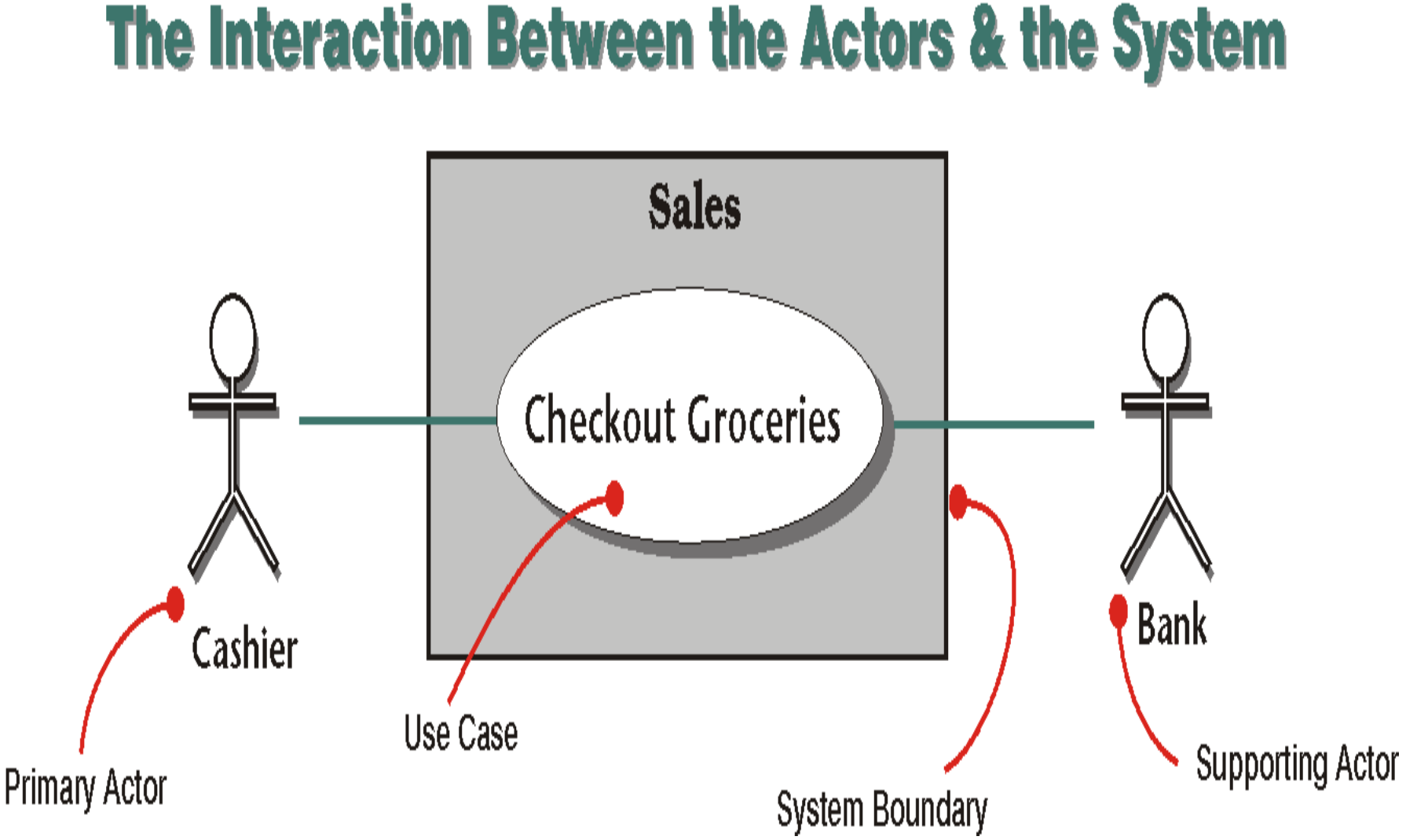
Purchase Groceries — Point-of-Sale System

- The customer deposits groceries on the checkout counter. The cashier scans each item and deposits the item on the bagging counter. When the last item is scanned, the cashier reads the total amount from the system and announces it to the customer. If the customer pays by credit card, the cashier swipes the card through the cash register to charge the amount. The customer then signs the printout. If the customer pays by cash, the cashier returns the change, if any. The cashier then gives a receipt to the customer.

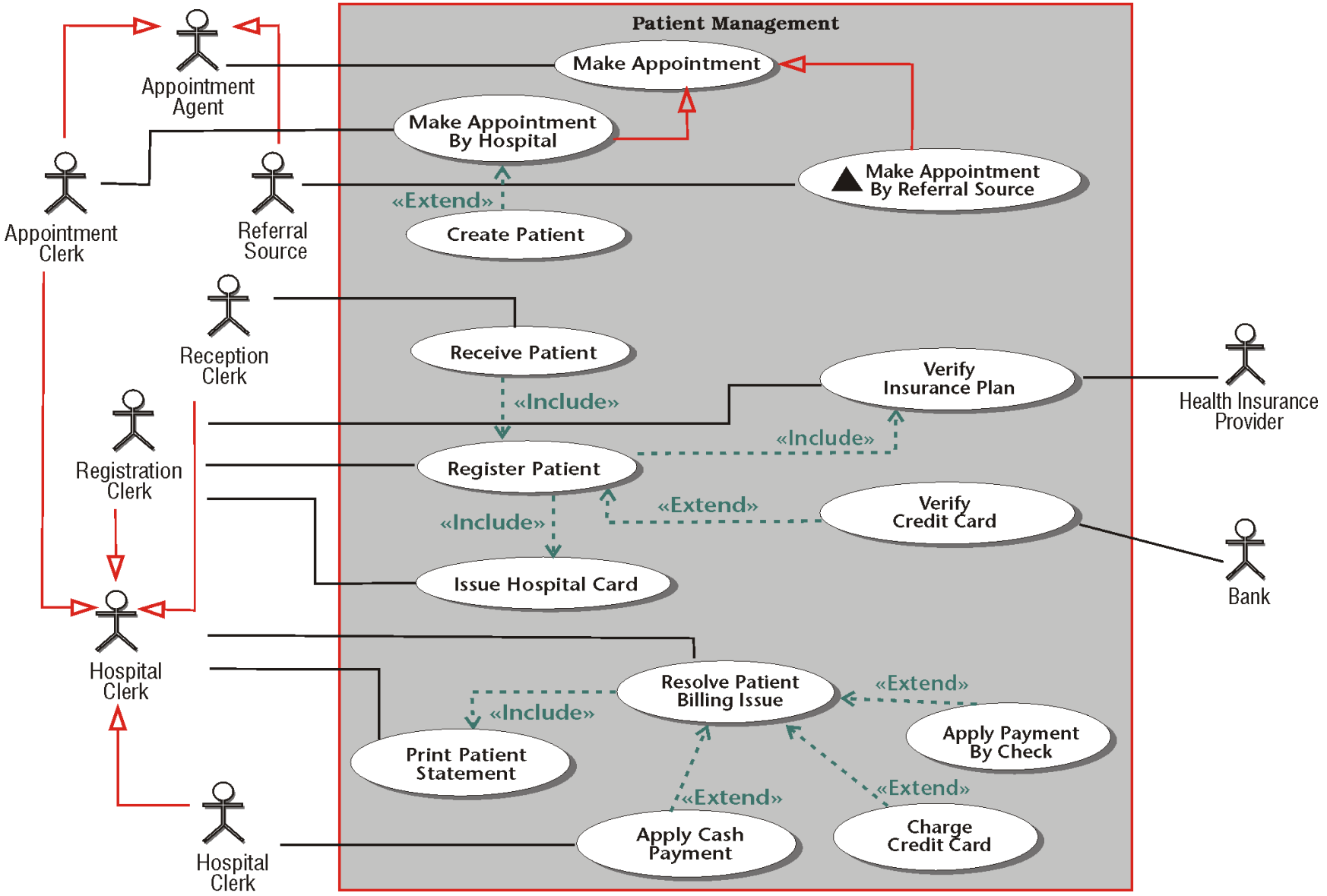
Use case or system diagram

- A “meta-model”, an abstraction of use cases.
- In its basic form, it displays
  - the boundaries of the system,
  - the name of the use cases, and
  - the actors who interact with each use case.

A very simple Use Case Diagram



A more complex system diagram



A Scenario

- The scenario is an ordered sequence of interactions between the actor(s) and the system to accomplish a goal.
- It consists of:
  - Normal Flow is the best-case scenario that results in the successful completion of the use case
  - Alternate Flow that is present only if conditional steps are needed
  - Sub-Flows if steps in the normal flow contain sub-steps
  - Exceptions that describe what may prevent the completion of one step or the entire use case

Steps in a Use Case Scenario

- Steps can be repeated
  - One step or a set of step can be repeated until a certain condition is met. In Checkout Groceries, the cashier scans purchase items until there are no more groceries are left to scan.
- A step can call on another use case
  - Each step may call on another use case to complete its function.

Steps in a Use Case Scenario

- A step is a transaction
  - Each step appears as just an interaction, but it is really a transaction between the actor and the system. That means that in each step:
    - the actor sends a request to the system,
    - the system validates the request,
    - the system changes its state as a result of validation, and then
    - the system responds.

How to Develop Initial Use Cases

- Components of use case modeling are provided by analyzing and expanding concepts that result from domain analysis.
- Domain analysis discovers entities that perform the same functions within the enterprise.

Identify Prominent Actors

- The primary candidates for becoming actors are domain concepts classified as "role."
- These entities are classified as "role" and are the prime candidates for designation as "actors" in use case modeling.
- They qualify as actors if they interact with the information system that we plan to build.

Identify Major Use Cases

- Major use cases are identified by analyzing business processes and functions
  - Sometimes we have to break up a process into more than one use case;
  - at other times we might have to combine pieces of multiple processes or functions to arrive at one use case.
- We arrive at major use cases by analyzing domain concepts marked as "process" or "function."

Identify Major Use Cases

- But the conversion ratio is not one-to-one. A use case has features that a function or a process may lack.
- Other domain concepts, such as objects or business rules might find their way into use cases if the context requires it.

The Template

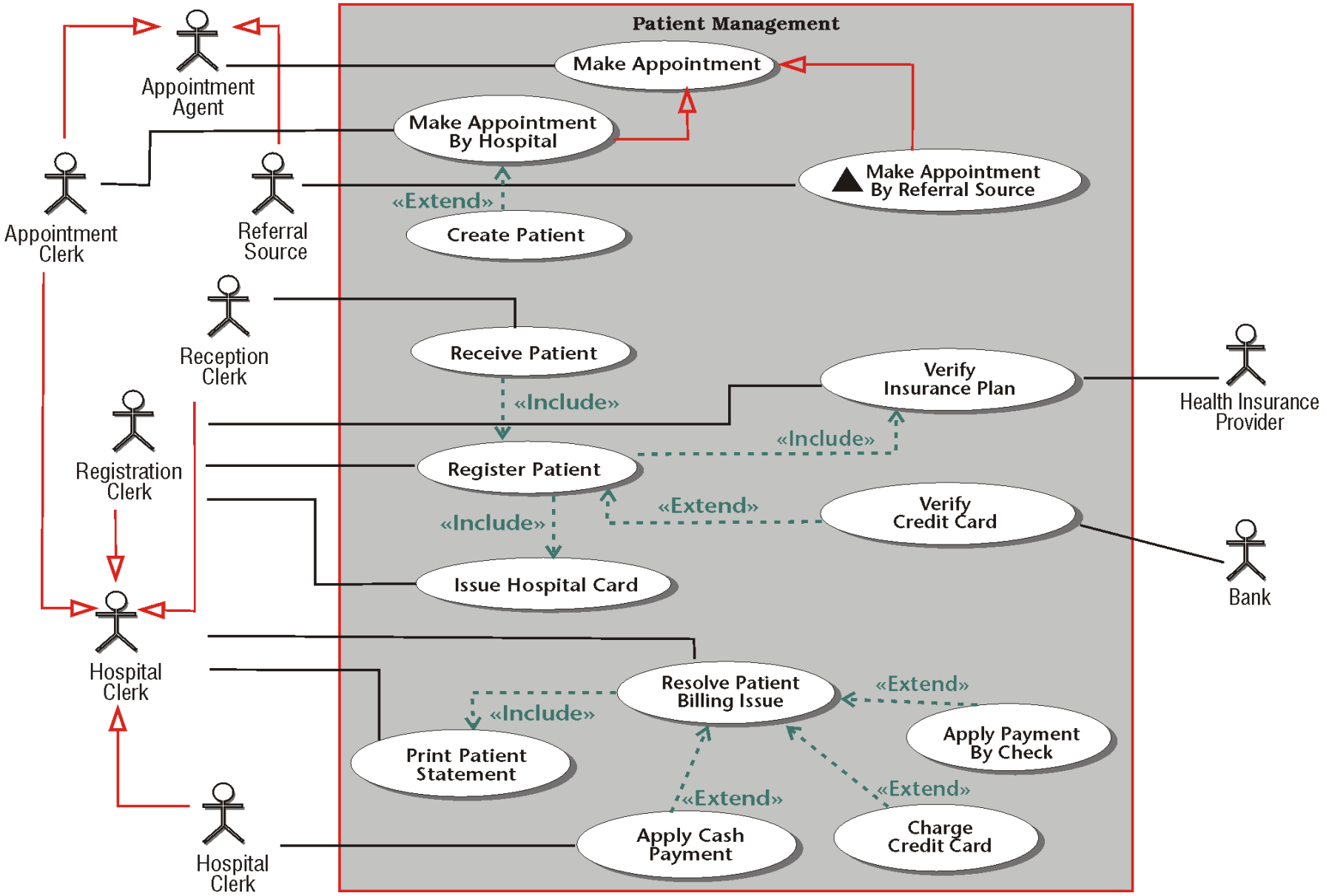
- The template structures use cases by providing well-defined and ordered fields.
- A formal template ensures that
  - a use case has the necessary building blocks and
  - it can communicate its scenario and attributes clearly.
- There is an example on Léa

Dependencies: Include and Extend

- When developing use cases, we may discover that some functionality has been left out:
  - we expected the customer to pay by cash, but we find out that the business accepts credit cards as well.
- Instead of rewriting the use case, we can create an "extending" use case and
  - add a conditional step to the "base" use case that branches to the new one if the customer decides to pay by credit card.

Use Case Diagram

A Meta-Model for the 'Big Picture'



Structural Modeling

Building Blocks of Information Systems

- An information system must have a structure that supports the system's behavior.
- A flexible and reliable structure, therefore, needs building blocks that satisfy the specific requirements of the structure.
- Structural modeling represents a view of the building blocks of a system or an entity and their interrelationships within a given scope.

Classes As Building Blocks

- Classes are the building blocks of structural modeling
- Objects are the structural units of the actual information system

Classes As Object Templates

- At runtime, when the information system is actually created, classes are "instantiated" into objects that function as the units of the information system.
- In the virtual world of software, a class is:
  - An abstraction of objects.
  - A template for creating objects.

Objects As Black Boxes

- An information system object is a dynamic black box; it interacts with outside entities to provide services but conceals its inner workings.
  - The internal structure of an object is known only to the object itself.

Encapsulation

- Encapsulation is enclosing data and processes within one single unit.
- Information hiding ensures that the inner entities and the workings of the object are concealed (and safe) from outside entities.

Encapsulation

- Encapsulation enables the object to enforce business rules with authority.
- Encapsulation results in two spaces:
  - Private. Data and processes that are inside the object are labeled as "private."
  - Public. Whatever the object exposes — that is, makes visible to the outside world — is "public."

Interface As A Contract

- In an information system, the only reason for an object's existence is the services that it offers to other entities
- Therefore, the interface of the object is characterized as a "contract" or a binding agreement.
- The object promises to perform services for an outside entity that behaves according to rules that the object expects.

Interface As A Contract

- Under this "contract," an object assumes certain responsibilities.
- Two categories:
  - what the object "knows", or attributes, and
  - what the object "does," or operations.
- A class defines the responsibilities and its instances, the objects, carry them out.
- The interface of an object, once formalized and made available for use, cannot be changed unless all parties to the contract agree to the change.

Structuring the Interface

- The interface of an object — its services — must itself be structured in a predictable manner. It has:
- Name
- Attributes
- Operations

Name

- Rules and conventions that apply to naming classes:
  - Class name must be a noun or a noun phrase.
  - Class name is usually singular (except for collection class).
  - Class name is always capitalized.
    - Student, ApprovalNotice
  - Definite or indefinite articles must be avoided — never APatient

Attribute

- Attribute is what an object knows.
- Class attributes are placeholders: it is the objects that fill the placeholders — or variables — with values.
- Attribute names begin with a lower-case letter; firstName.
  - The lower-case start is a convention to distinguish attributes and operations from classes.

Operation

- Operation defines what an object does or what can be done to it
  - A class merely defines what an object is expected to do.
  - It is the object that carries out the actual operation: a Plane class does not fly; a plane object does
- Rules for naming operations are the same as for naming attributes.
  - move (verb)
  - getStarted (verb)

Responsibilities

- An object's responsibilities consist of what it does and what it knows; in other words, its operations and its attributes.
- Before we can define classes in detail, we must discover class candidates and outline their tentative responsibilities.

Finding Class Candidates

- By parsing use cases
- The best starting point for discovering responsibilities are the use cases.
  - In very broad terms, use cases specify what a class must "know" and what a class must "do."

Finding Classes

- The messages exchanged between the actors and the system refer to objects that are affected by the interaction between the two:
  - by parsing the messages that the steps in a use case scenario specify, we can start the discovery of classes.

Finding Class Candidates

1) identify nouns that serve as grammatical objects,
2) search for grammatical objects hidden in verbs that have both transitive and intransitive forms,
3) discard obvious misfits,
4) outline the general responsibilities of the candidates by analyzing its relationships with other nouns and sometimes verbs within the flow of the use case, and finally
5) consolidate our findings in a preliminary list of candidates and their responsibilities.

Finding Class Candidates

- In this process, we intentionally avoid including details and, instead, outline the responsibilities only in broad terms.
  - Even if use cases do offer details — which they usually should not and do not — we must disregard them for the moment.

Finding Class Candidates

Make Appointment	Normal Flow:	1. Appointment clerk verifies that the needed medical service is provided by the hospital. 2. Appointment clerk records patient's personal and contact data. 3. Appointment clerk records information about the referral source. 4. Appointment clerk consults hospital's schedule to find a free slot for the required medical service. 5. Appointment clerk verifies that the patient is available for the appointment. ➔ Loop 1: **Repeat** steps 4-5 until hospital's schedule matches patient's availability. 6. Appointment clerk makes the appointment. ➔ Loop 2: **Repeat** steps 4-6 for each appointment.

Elaborating Classes

- To fully define a class or an object is to define responsibilities in detail.
- Discovery of class candidates and their general responsibilities is required but is not enough.
- We must also
  - ❶ confirm that the tentative responsibilities do belong to the class and
  - ❷ specify the exact nature of those responsibilities.
- In the process, we often discover new classes that must collaborate with original class to fulfill some of its responsibilities.

Class Relationships

- To build an structural model we must identify its building blocks, but to complete the task we must also define how its constituent units relate to each other.
- Association
- Aggregation
- Composition

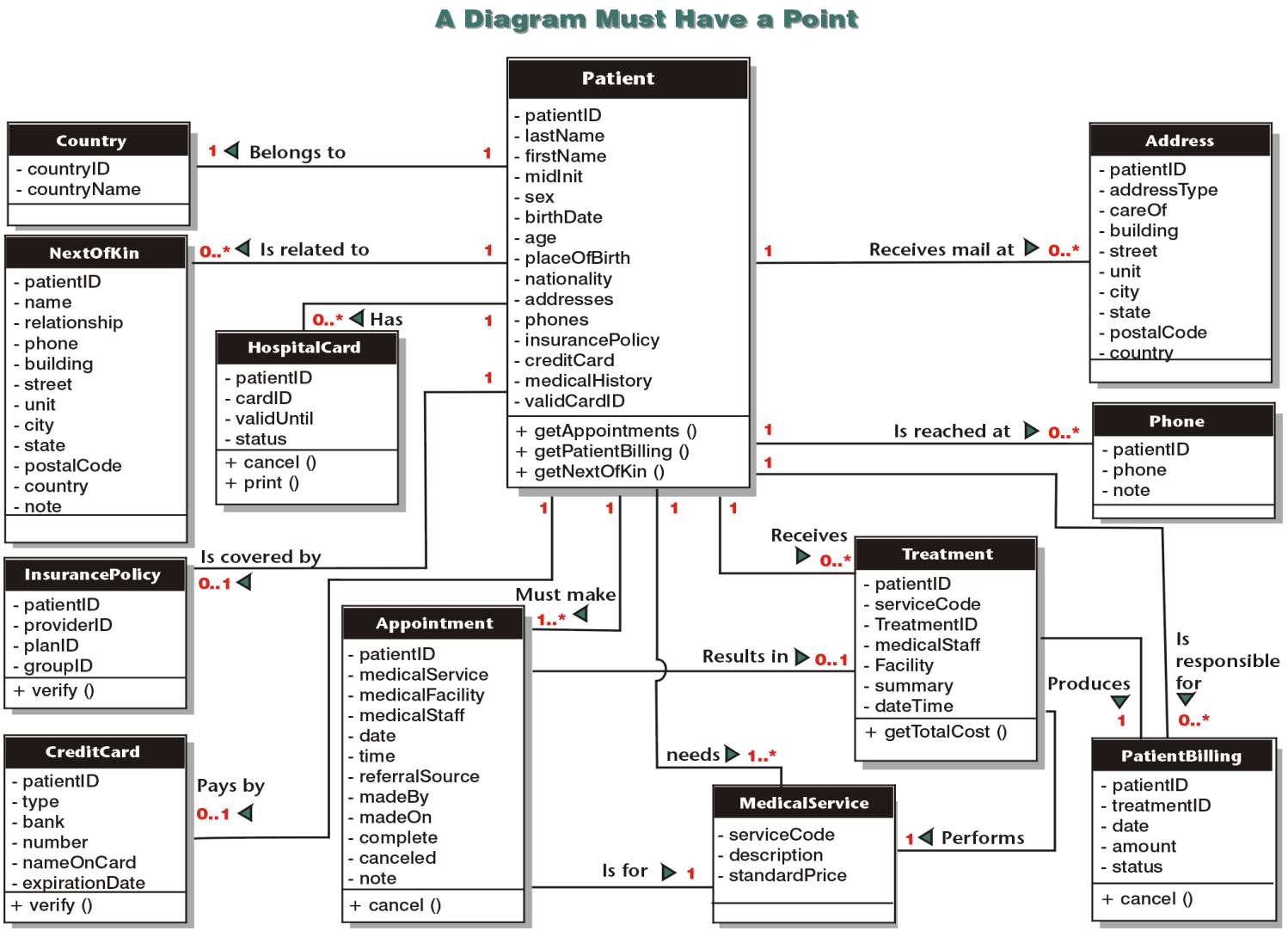
Association

- Defines the connection between objects of one class with the objects of another class in semantic — that is, word-based — terms:
  - a Patient is covered by an InsurancePolicy,
  - A Customer pays by a CreditCard,
  - an Author is the writer of a Book, and so on.
- Constraints are rules that apply to associations:
  - a fulltime student must take at least 12 credits per semester,
  - a patient can be issued many hospital cards but only one card can be valid at any given time.

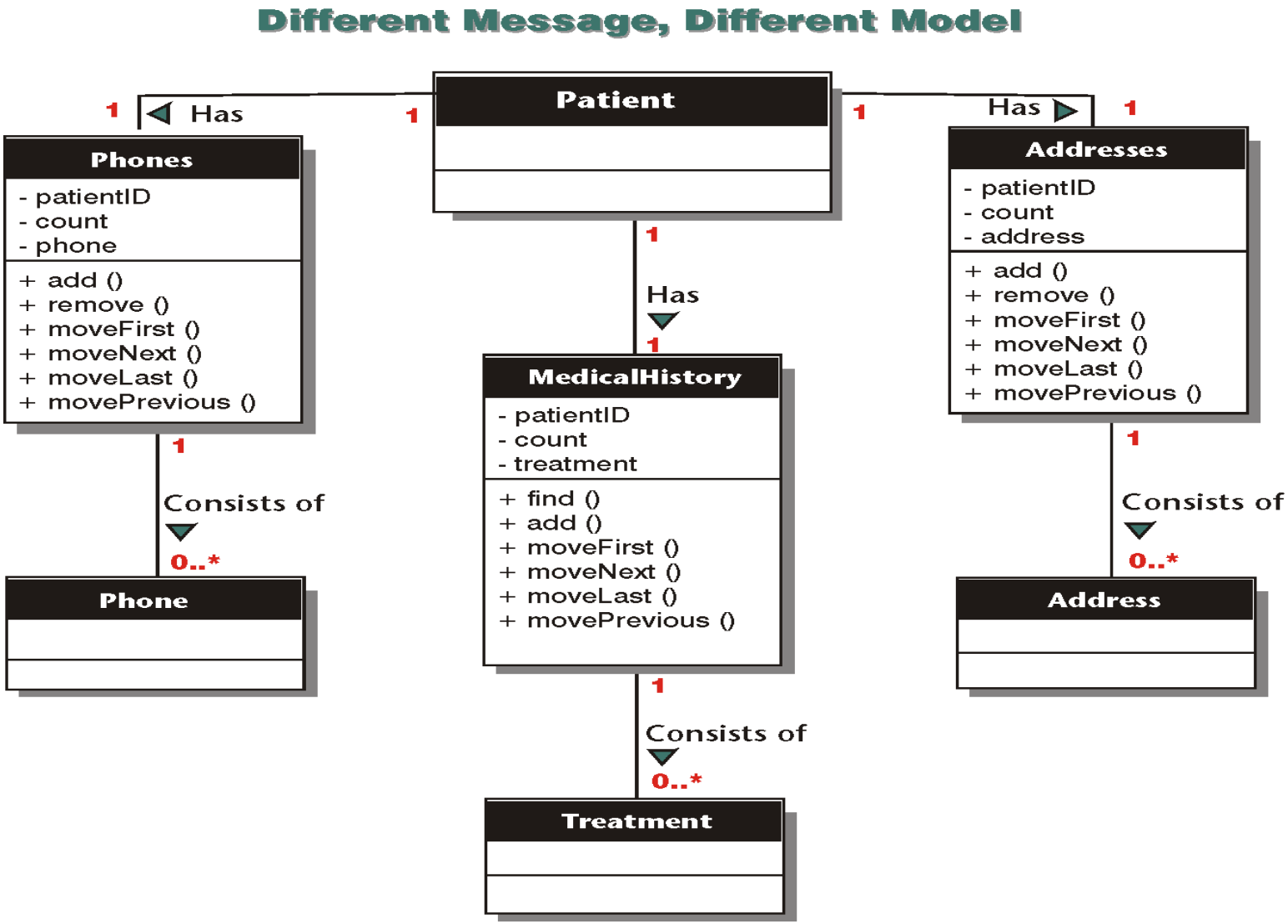
Class Diagram

- Shows a set of classes and their interrelationships.
- Indispensable tool for modeling class relationships.
- Elements are few and simple but can model a wide variety of actual relationships.
- No single class diagram can model all classes and all their relationships for even a small system.
- Therefore, to represent the structure of a system, we need to create a set of class diagrams, each with a recognizable focus and purpose.

Patient and its Associations



Patient and its Collection Attributes



Multiplicity

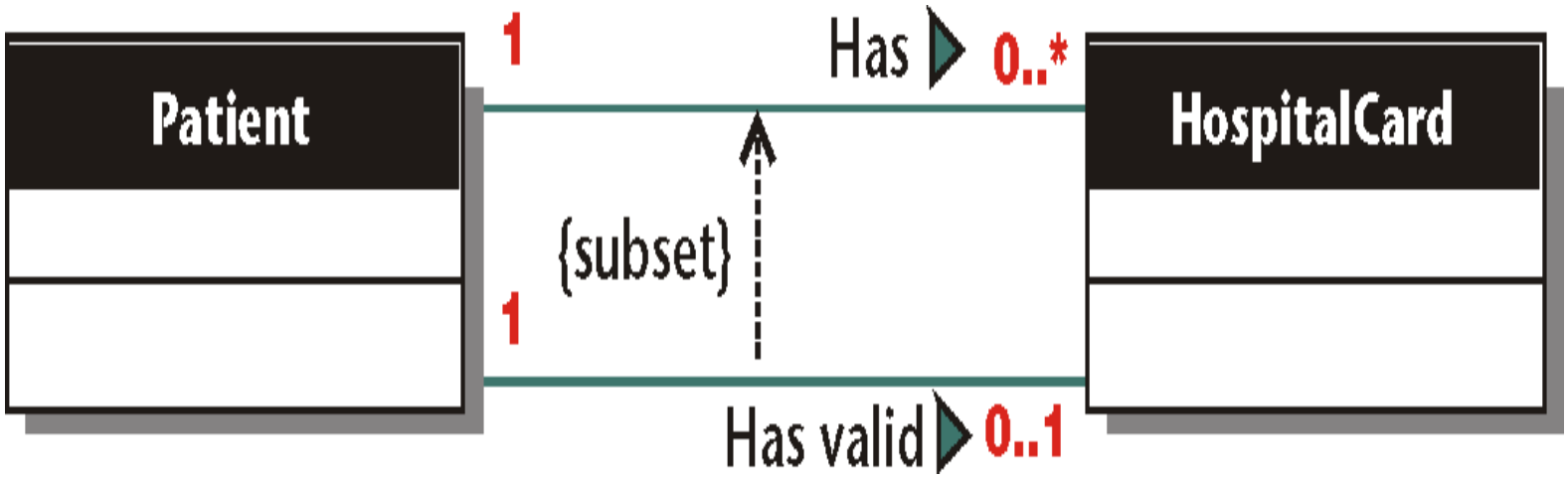
- Specifies how many instances of one class can associate with instances of another class.
- The quantitative association between instances of one class with instance of another class:
  - 1 patient can have only 1 nationality,
  - 1 customer can purchase up to 100 books,
  - 1 student must register for at least 6 credits per semester but cannot enroll in more than 18, etc.

Multiplicity

Multiplicity	Meaning	Example
1	Exactly one	A patient must have one, and only one, nationality.
0..1	Zero or one	A patient can have no insurance plan or can have one.
1..*	One or more	A patient must have at least one appointment to receive medical service, but can have as many as necessary.
0..*	Zero or more	A patient can have no billing activity or many.
20..40	A defined range	A part-time worker must work at least twenty hours a week, but no more than forty.
2,4, 6, 8	A non-continuant range	Tables are set for 2, 4, 6, or 8 people.

Constraints

- In structural modeling, constraints are rules that apply to associations.



Generalization & Specialization

- Generalization is a relationship in which one class is the more abstract expression of the properties of a set of classes:
  - a Tree class embodies properties common to Oak, Birch, and Cedar classes.

Generalization & Specialization

- Conversely, specialization is a relationship in which a class is the less abstract expression of another class:
  - a Rose is a Flower but the Flower class does not express the specific features of a Rose that sets it apart from other flowers.
- The result of generalization is a superclass (or a "parent"), while specialization arrives as subclasses (or "children").

Dynamic Modeling

Dynamic Modeling

- Dynamic modeling represents the interaction of the building blocks of the information system with each other and with the outside world to satisfy the behavioral requirements of the system.
- Dynamic modeling is also interaction.
- Unlike the real world in which objects interact in many different ways, in the virtual world objects can only interact through messages.

Dynamic Modeling

- Interactions happen in time.
- Dynamic modeling must show not only who interacts with whom and how, but in what order.
- Dynamic modeling represents structure in motion.

Object Interaction

- In an object-oriented virtual system the only possible way of interaction is through sending and receiving messages.
- In a virtual system, a message is equivalent to an action in the real world.

Object Interaction

- Remember, an object is a black box.
  - Its inner workings and its knowledge are hidden from the outside world.
  - The users of the system and other objects interact with the object only through its public interface.
- How do the users interact with the system?
  - The answer is by exchanging messages.

Messages

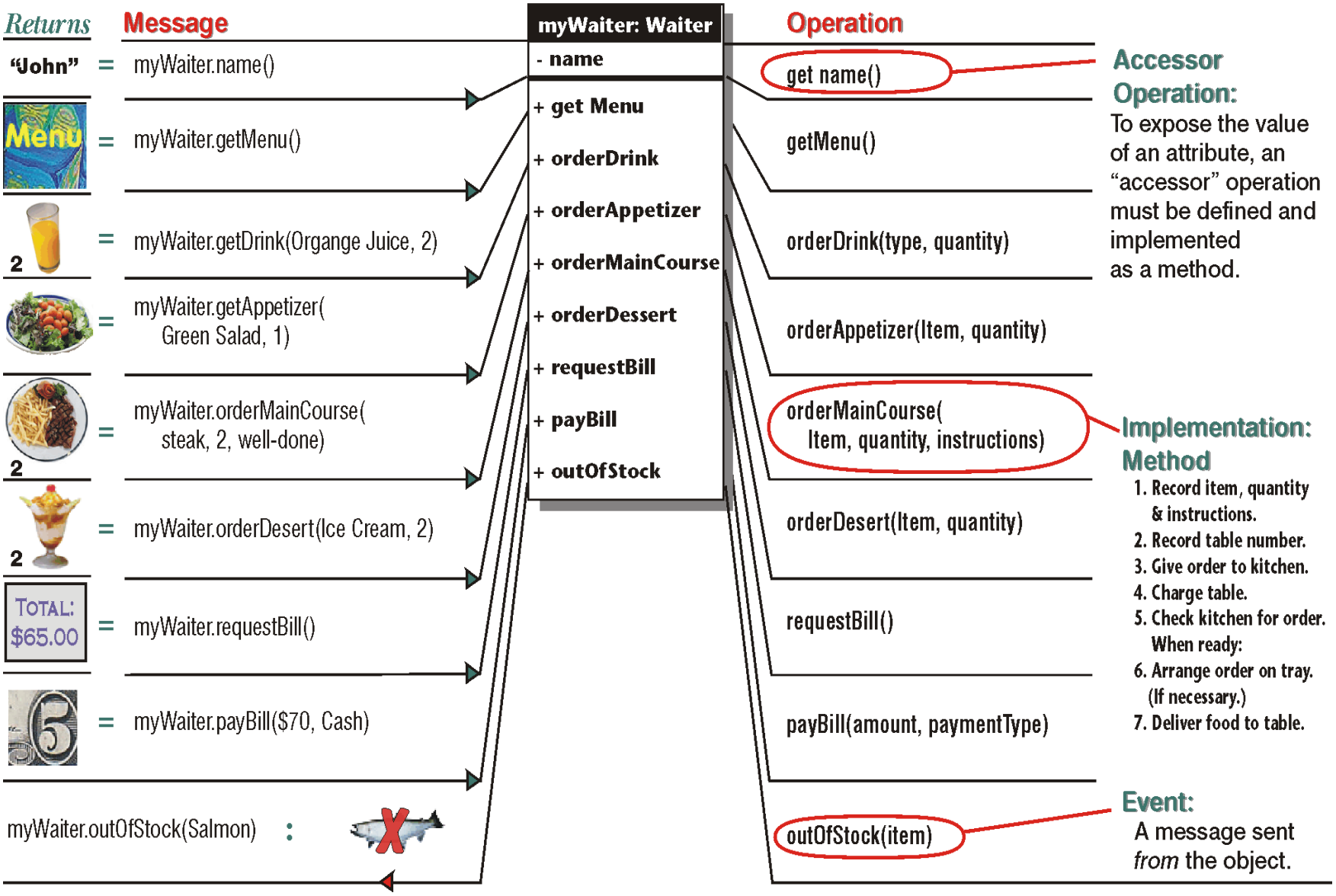
- Messages are instructions and information sent to objects in the expectation that the recipient objects will carry out certain actions.

Restaurant Example

Use Case:	Have Dinner
Scope:	Restaurant
Primary Actor:	Customer
Normal Flow:	**1.** Customer is seated at a table **2.** Customer asks the waiter for the menu. **3.** Customer orders drink(s). **4.** Customer orders appetizer(s). **5.** Customer orders main course(s). **6.** Customer orders dessert(s). **7.** Customer asks for the bill. **8.** Customer pays the bill and receives change (if any).
Alternate Flow / Exceptions:	**1.a** Customer asks the waiter for his or her name.

Methods & Messages

How Objects Interact



Parameters

- Parameters, or arguments, specify the data that must be supplied to an object to carry out a specific operation:
- Examples:
  - orderDrink(drink, quantity)
  - orderAppetizer(Appetizer)
  - getMenu()
  - orderMainCourse(whisky, on the rocks, many)
- The object responsible for an operation defines the types (and provides the containers or variables) for the data

Return Value

- Return value is the reply that a message may invoke from the receiving object after an operation is complete.
- After the recipient object has carried out an operation, it might send a return result to the sender of the message.

Syntax

- The exact syntax for declaring operations and sending messages is language dependent.
  - [Visibility] [Return Type] [Name](Param 1, . . . , Param n)
  - Public Currency payBill(amount, paymentType)

Methods

- A method is how an operation is implemented or actually carried out by the object responsible for the operation.
  - polymorphism; one operation => more than one method.
- An operation defines what the objects instantiated from a class must do, but not how they are expected to do it.
- It is the method that implements the actual steps required to carry out a task.

Methods

- Instances of two classes may have the same exact operation, but they can implement it very differently even if the classes are related.
- Even instances of the same class may not carry out the same operation in the same way if their states are different.
  - Very important to dynamic modeling, conceptual or otherwise.

Events

- Actions by one object that interrupt the existing condition of one or more other objects.
- Three types
  - Call
  - Signal
  - Time

Call Events

- Invoke an operation
- When an object sends a message to another object, the recipient of the message views the arrival of the message as an event to which it must respond.
- They are the other side of the coin to messages.
- Usually both targeted (directed to a specific object) and synchronous (the event and the response to it work within the same phase or timeframe).

Signal Events

- Anonymous, meaning that they are not directed to a specific object, but will be received by any object that "catches" them.
- "Call" events may be compared to receiving a personal telephone call, while "signal" events are like buying newspapers or tuning to a radio station to learn about "events."
- Usually asynchronous
  - the action that results in the event may not occur in the same "phase" or timeframe that the event occurs.

Time Events

- Like signal events: disruptive, asynchronous, and anonymous.
- The important difference
  - triggered by the passage of time, not by any prior action.
- From the viewpoint of object-orientation, all the types above could be significant events if they change the state of an object (or a system).

Dynamic Diagrams

- Sequence diagram
  - Emphasizes the order of interactions in time.
- Statechart diagram
  - Traces the results of interactions on the state of objects belonging to a specific class.
- Activity diagram
  - Concentrates on the logical flow of activities.

Sequence Diagram

- Sequence diagram represents the interaction between objects, or between actors and objects ordered in time.
- A sequence diagram is composed of a timeline, objects that interact across this timeline, and the messages that they exchange.

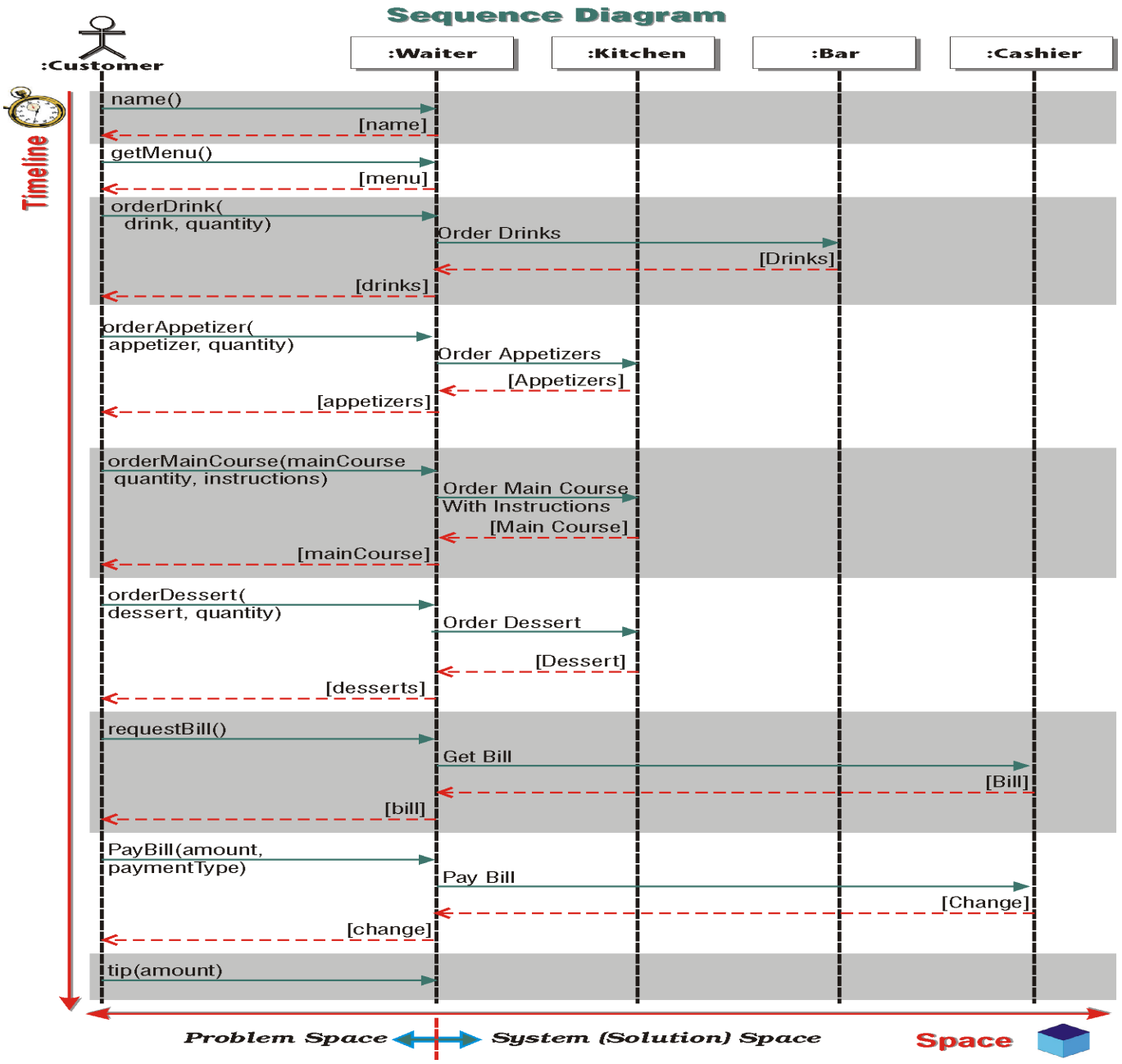
Sequence Diagram

- A sequence diagram is basically a space-time matrix for interactions.
- Two axes contain the diagram.
  - The vertical axis is timeline, or a chronology of events.
  - The horizontal access identifies the space and the actions that take place within the space.
  - Movement on the timeline axis is one way, but bidirectional on the space axis.

Sequence Diagram

- Each column in the sequence diagram identifies an instance of an actor or a class.
- Each "row" is a messages sent or received by the entities in the column.

Have Dinner Sequence Diagram



Sequence Diagram

- The objects across the top of the diagram are interacting with each other by passing messages back and forth (horizontal).
- An actor is the user whose interactions with the system are described in each of the use cases (horizontal).
- The object lifetime represents time and starts with the top-most message (vertical).

Elements of Sequence Diagram

- Actor Instance: outside the system.
  - Instance Name:Actor Name
  - aCustomer:Customer
- Class Instance: inside the system
  - Instance Name:Class Name
- Timeline---dotted vertical line.

Elements of Sequence Diagram

- Object Lifetime: A hollow box on the timeline identifies the lifetime of an object.
- Instantiation: specifies when the lifetime of the instance starts.
- Destruction: specifies when the lifetime of the instance ends and the object is destroyed.
- Message & Timeline Forking: Specifies alternates.
- Message to Self & Looping: Specifies repeats.

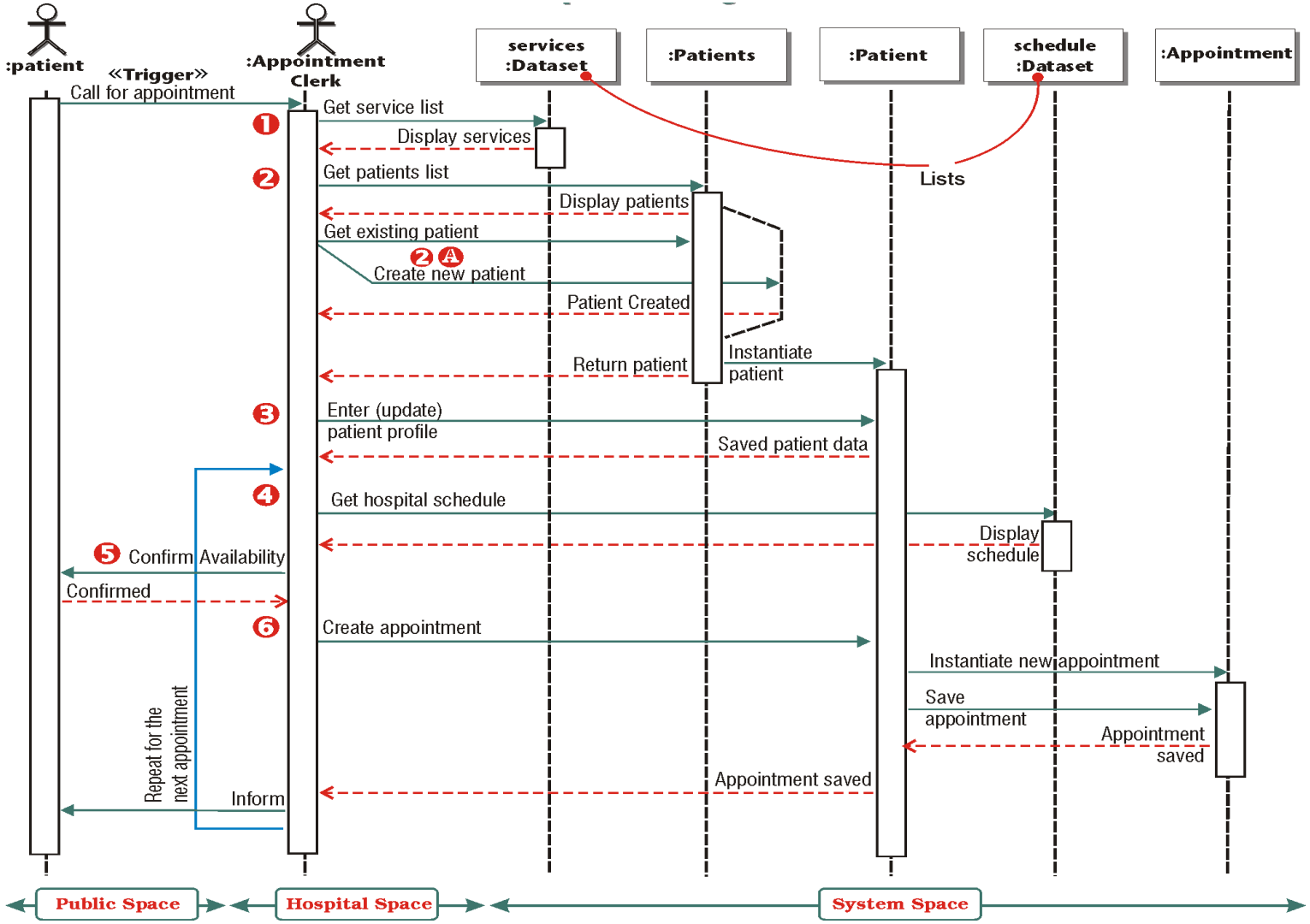
Object Lifetime

- The lifetime of an object specifies when it is instantiated, how long it exists and when it is destroyed.

Make Appointment Use Case

Normal Flow:	**1.** Appointment clerk verifies that the needed medical service is provided by the hospital. **2.** Appointment clerk records patient's personal and contact data. **3.** Appointment clerk records information about the referral source. **4.** Appointment clerk consults hospital's schedule to find a free slot for the required medical service. **5.** Appointment clerk verifies that the patient is available for the appointment. ➔ Loop 1: **Repeat** steps 4-5 until hospital's schedule matches patient's availability. **6.** Appointment clerk makes the appointment. ➔ Loop 2: **Repeat** steps 4-6 for each appointment.
Alternate Flow/ Exceptions:	**2.a** Patient is not on file. Create new patient. (Extend: 141 - Create Patient.)

Make Appointment Sequence Diagram



Activity Diagram

- Provides the most lucid tool for modeling the logical flow of activities that takes place between the system and the outside world or within the system among its components.
- Another view of the same scenario that the sequence diagram illustrates, but
  - while sequence diagram focuses on the exchange of messages between objects and object lifetimes,
- Activity diagram is concerned with the logical flow of activities.

Activity Diagram

- Sequence diagram is a basically a sequential presentation of actions
- Activity diagram can efficiently portray parallel activities

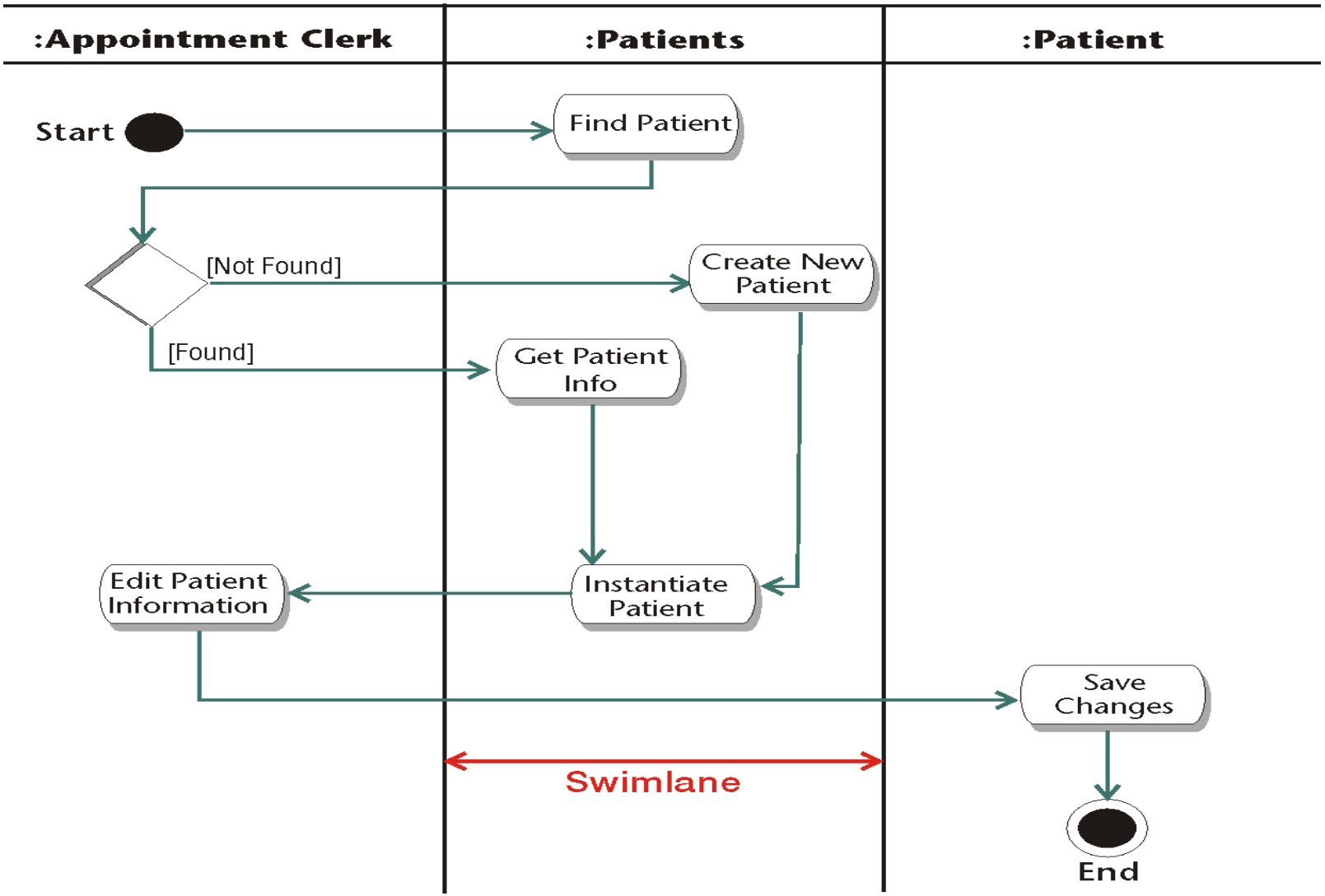
Activity Diagram

- Activity diagram is a general-purpose diagram for visualizing and verifying a logical flow.
- Depicts the flow from activity to activity.
- It presents a visual, dynamic view of the system and its components.
- For a use case with a complicated alternate flow, activity diagram is a very helpful tool.
- Its simplicity especially facilitates communications between development stakeholders.

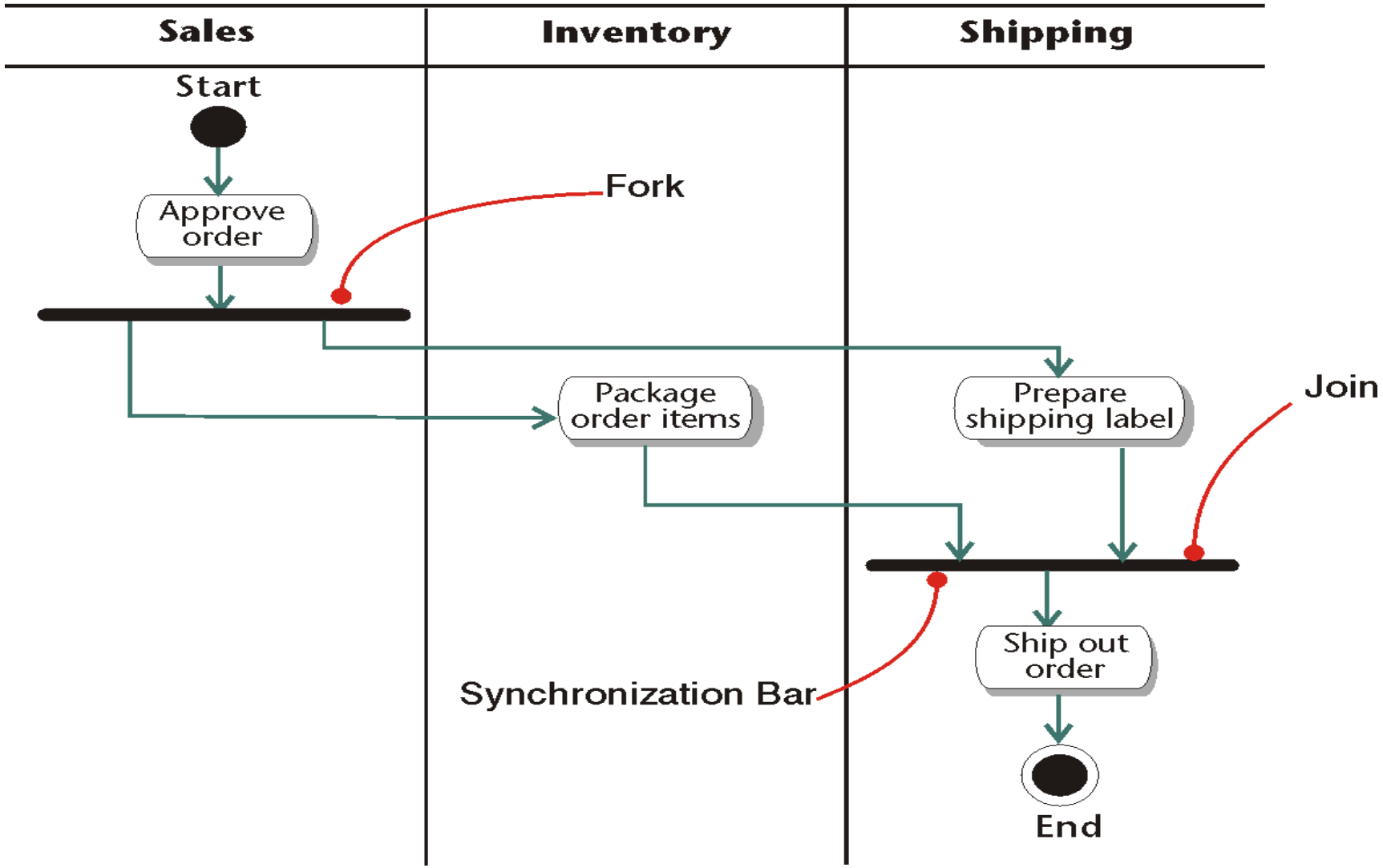
Activity Diagram

- Swimlane, a vertical partition on the activity diagram that organizes responsibilities for actions
- Synchronization bar where parallel actions (or forks) merge back.
- With these tools, activity diagram can be employed as a workflow diagram to model actions across the enterprise.

Enter Patient Data Activity Diagram



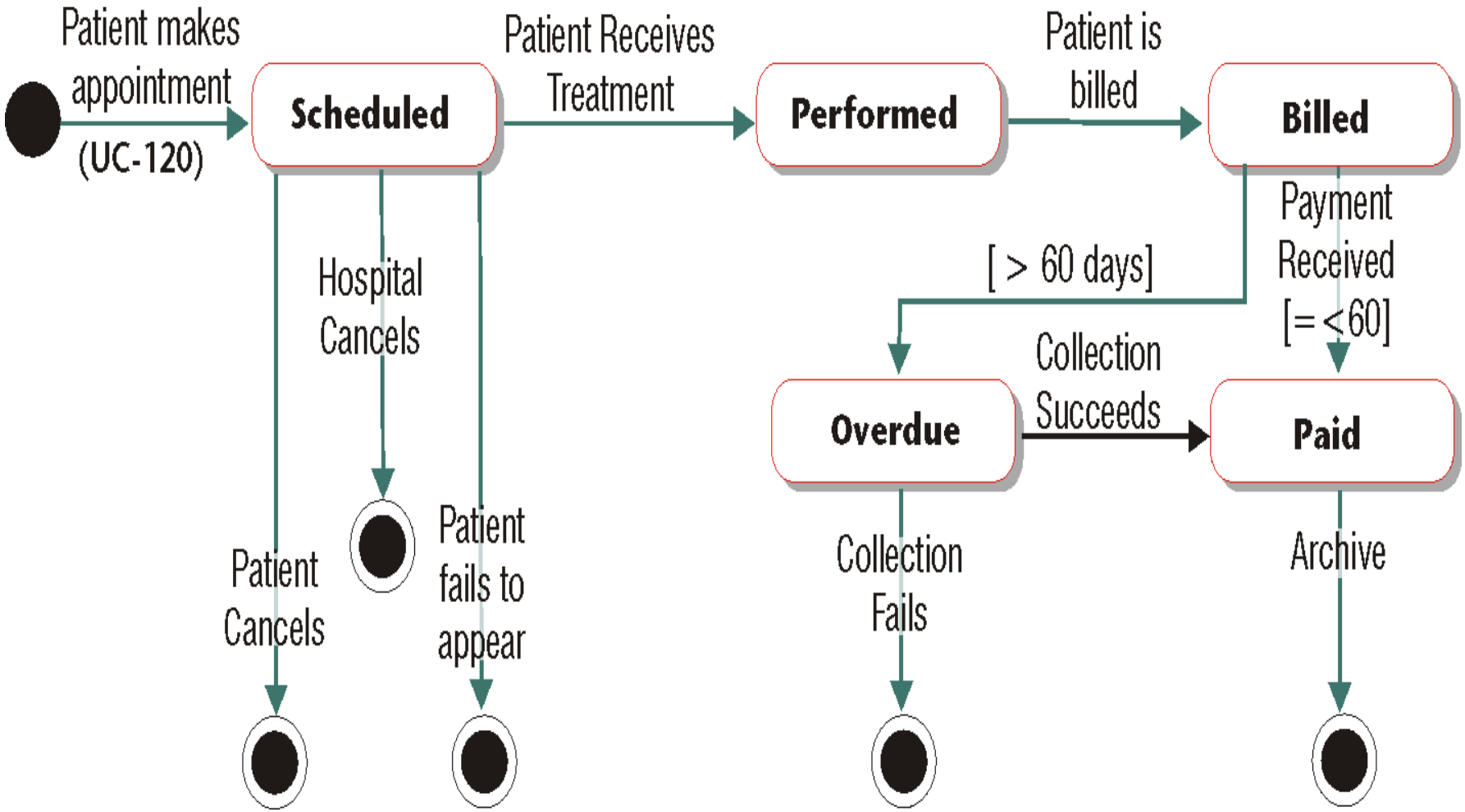
Ship Order Activity Diagram for Workflow Model



Statechart Diagram

- Like living objects in the real world, a virtual object is born and changes during its lifetime.
  - Like living objects, it might also cease to exist.
  - Statechart diagram models the milestones in the life of an object when its state is changed by a significant event.
- A state is an object's condition at a certain stage and from a certain viewpoint.
  - It is a snapshot of the object in a usually important point in time.
- Statechart diagram is composed of the states of an object and the flow of events that change its state.

Treatment Statechart Diagram



Elements of Statechart Diagram

- Initial State (Starting Point)
- State
- Transition & Event
- Final State (Termination Point)

The Value of Statechart Diagram

- Statechart diagram is the only dynamic model that illustrates the milestones in the lifetime of one class of objects in its entirety.