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 generally come to mean representing the system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).

You may develop different models to represent the system from different perspectives.

For example:

1. An external perspective, where you model the context or environment of the system.

2. An interaction perspective where you model the interactions between a system and its environment or between the components of a system.

3. A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.

4. A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.

1. Context models

Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries. Social and organizational concerns may affect the decision on where to position system boundaries.

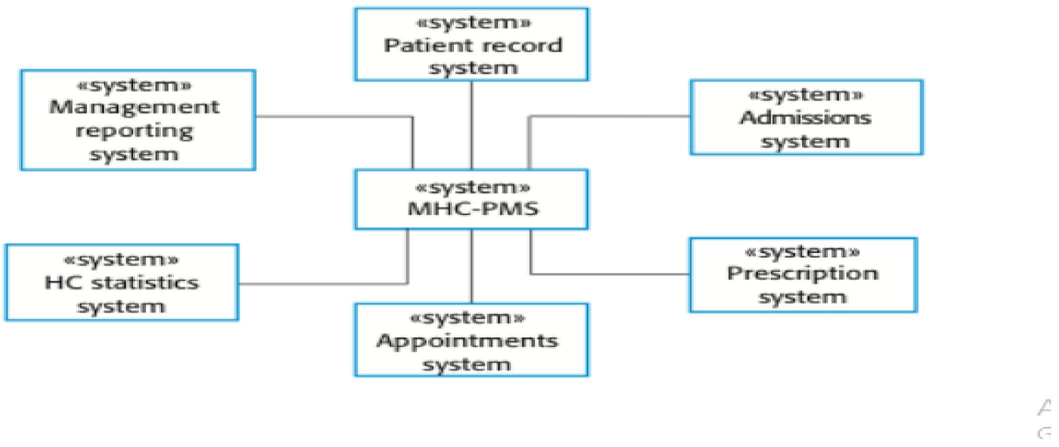
At an early stage in the specification of a system, you should decide on the system boundaries. This involves working with system stakeholders to decide what functionality should be included in the system and what is provided by the system's environment. You may decide that automated support for some business processes should be implemented but others should be manual processes or supported by different systems. You should look at possible overlaps in functionality with existing systems and decide where new functionality should be implemented. These decisions should be made early in the process to limit the system costs and the time needed for understanding the system requirements and design.

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 since there may be pressures to develop system boundaries that increase/decrease the influence or workload of different parts of an organization.

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.

(Mental Health Care-Patient Management System)

The context of the MHC-PMS



2. Interaction models

All systems involve interaction of some kind. This can be user interaction, which involves user inputs and outputs, interaction between the system being developed and other systems or interaction between the components of the system. Modeling user interaction is 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.

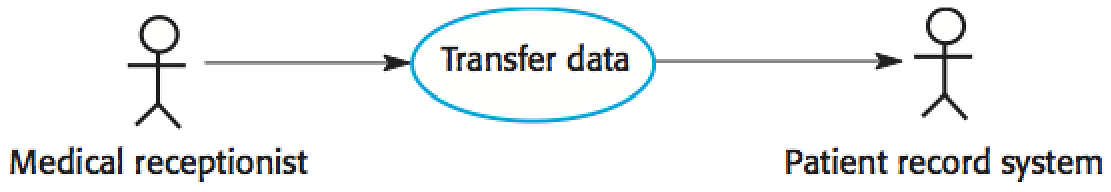
Two related approaches to interaction modeling:

1. Use case modeling, which is mostly used to model interactions between a system and external actors (users or other systems).

2. Sequence diagrams, which are used to model interactions between system components, although external agents may also be included.

1. Use case modeling. Use case modeling is widely used to support requirements elicitation. A use case can be taken as a simple scenario that describes what a user expects from a system.

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. Use cases can be represented using a UML use case diagram and in a more detailed textual/tabular format.

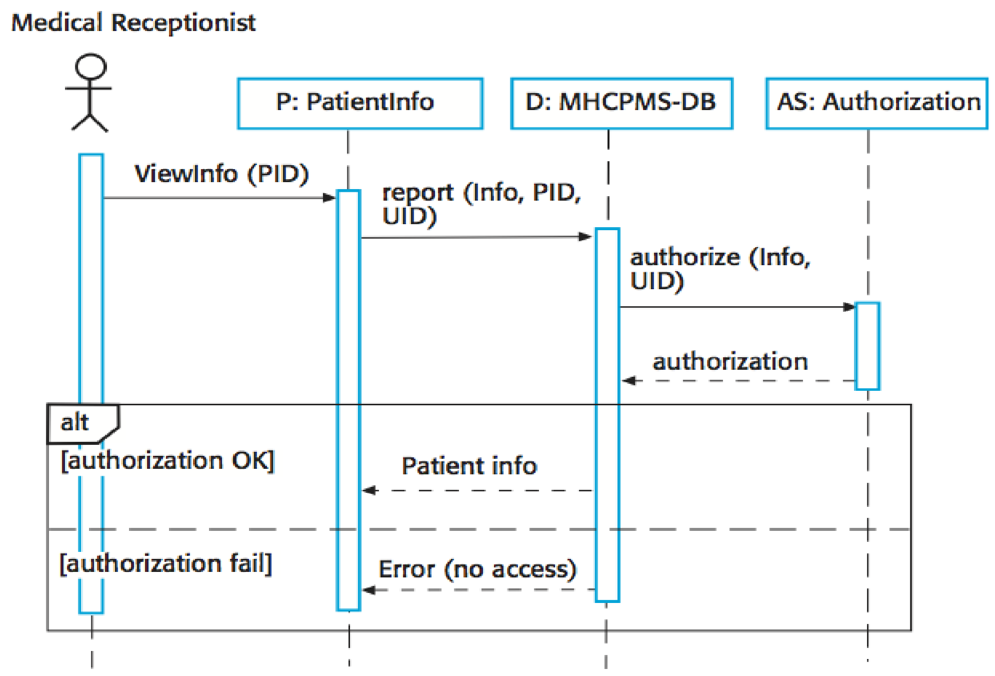


Use case description in a tabular format:

Use case title	Transfer data
Description	A receptionist may transfer data from the MHC-PMS to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient's diagnosis and treatment.
Actor(s)	Medical receptionist, patient records system (PRS)
Preconditions	Patient data has been collected (personal information, treatment summary); The receptionist must have appropriate security permissions to access the patient information and the PRS.
Postconditions	PRS has been updated
Main success scenario	1. Receptionist selects the "Transfer data" option from the menu. 2. PRS verifies the security credentials of the receptionist. 3. Data is transferred. 4. PRS has been updated.
Extensions	2a. The receptionist does not have the necessary security credentials. 2a.1. An error message is displayed. 2a.2. The receptionist backs out of the use case.

2. Sequence diagrams

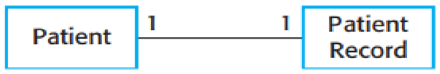
UML **sequence diagrams** are used to model the interactions between the actors and the objects within a system. A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance. The objects and actors involved are listed along the top of the diagram,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

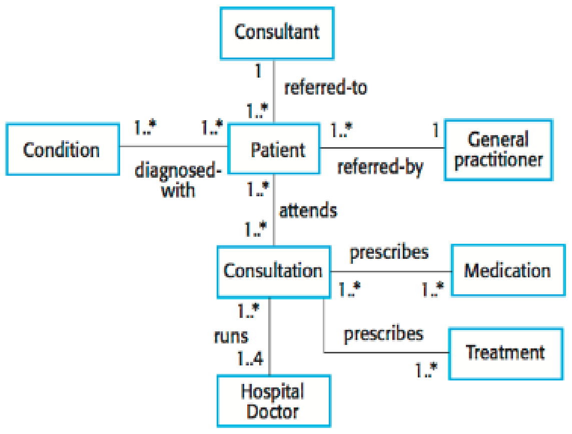


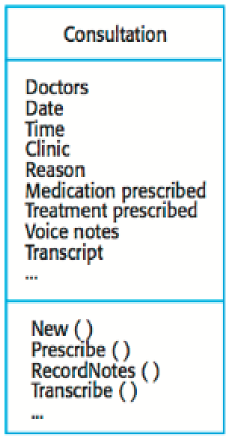
3. Structural models

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. These are not the same things—the dynamic organization of a system as a set of interacting threads may be very different from a static model of the system components.

1. **Class diagrams:** 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.







In this example, each end of the association is annotated with a 1, meaning that there is a 1:1 relationship between objects of these classes. That is, each patient has exactly one record and each record maintains information about exactly one patient. As you can see from later examples, other multiplicities are possible. You can define that an exact number of objects are involved or, by using a *, as shown in Figure.

Class diagram shows possible attributes and operations on the class Consultation

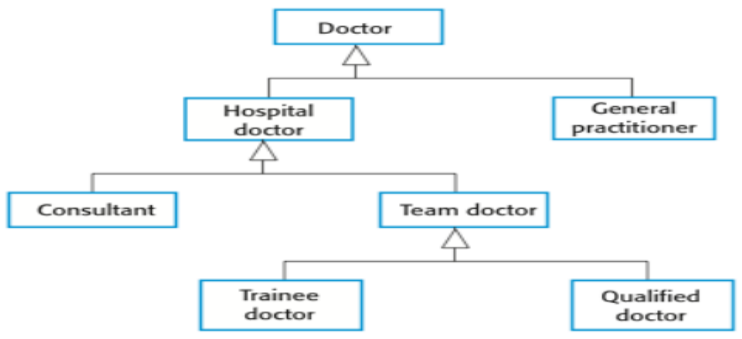
1. The name of the object class is in the top section.

2. The class attributes are in the middle section. This must include the attribute names and, optionally, their types.

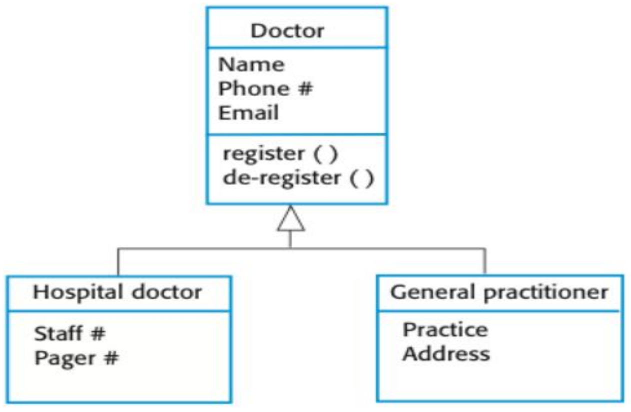
3. The operations (called methods in Java and other OO programming languages) associated with the object class are in the lower section of the rectangle.

2. **Generalization:** Generalization is an everyday technique that we use to manage complexity. In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. 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 super classes. These lower-level classes then add more specific attributes and operations.

A generalization hierarchy

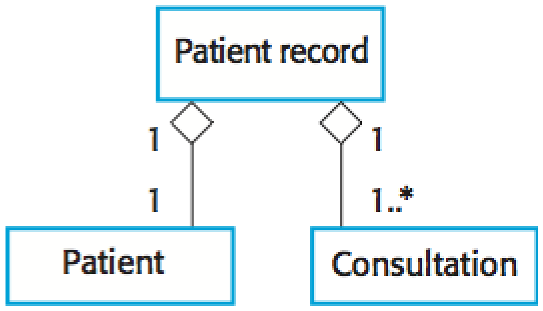


A generalization hierarchy with added detail



In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. This means that common information will be maintained in one place only. This is good design practice as it means that, 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.

3. **Aggregation:** Objects in the real world are often composed of different parts. For example, a study pack for a course may be composed of a book, PowerPoint slides, quizzes, and recommendations for further reading. Sometimes in a system model, you need to illustrate this. The UML provides a special type of association between classes called aggregation that means that one object (the whole) is composed of other objects (the parts). To show this, we use a diamond shape next to the class that represents the whole. which shows that a patient record is a composition of Patient and an indefinite number of Consultations.



4. Behavioral models

Behavioral models are models of the dynamic behavior of the 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:

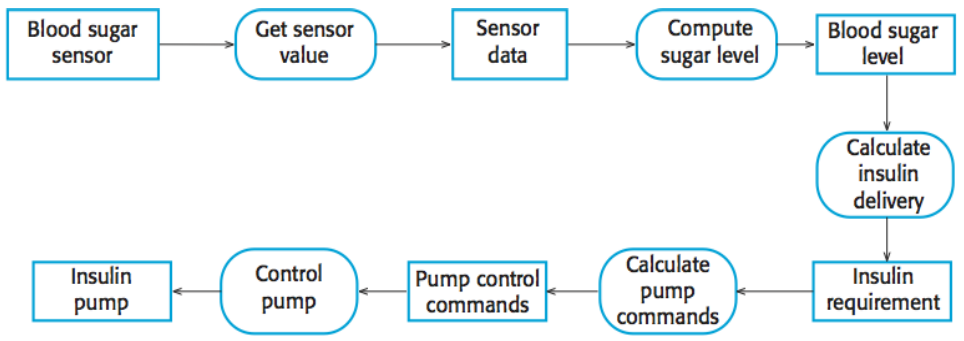
1**. Data** Some data arrives that has to be processed by the system.

2. **Events** Some event happens that triggers system processing. Events may have associated data but this is not always the case.

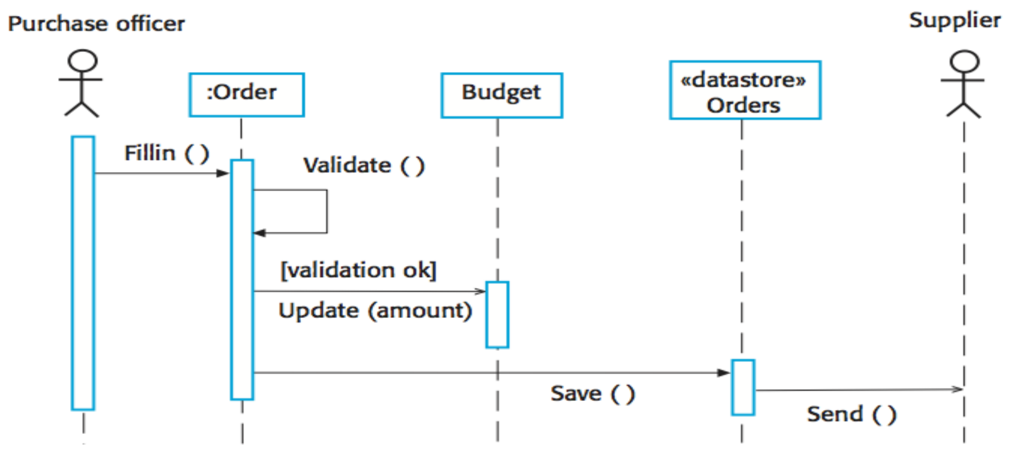
1. Data-driven modeling

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. That is, they show the entire sequence of actions that take place from an input being processed to the corresponding output, which is the system's response.

Data-driven models can be created using UML **activity diagrams**:



Data-driven models can also be created using UML **sequence diagrams**:



2. 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 models** 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. Event-driven models can be created using UML **state diagrams**:

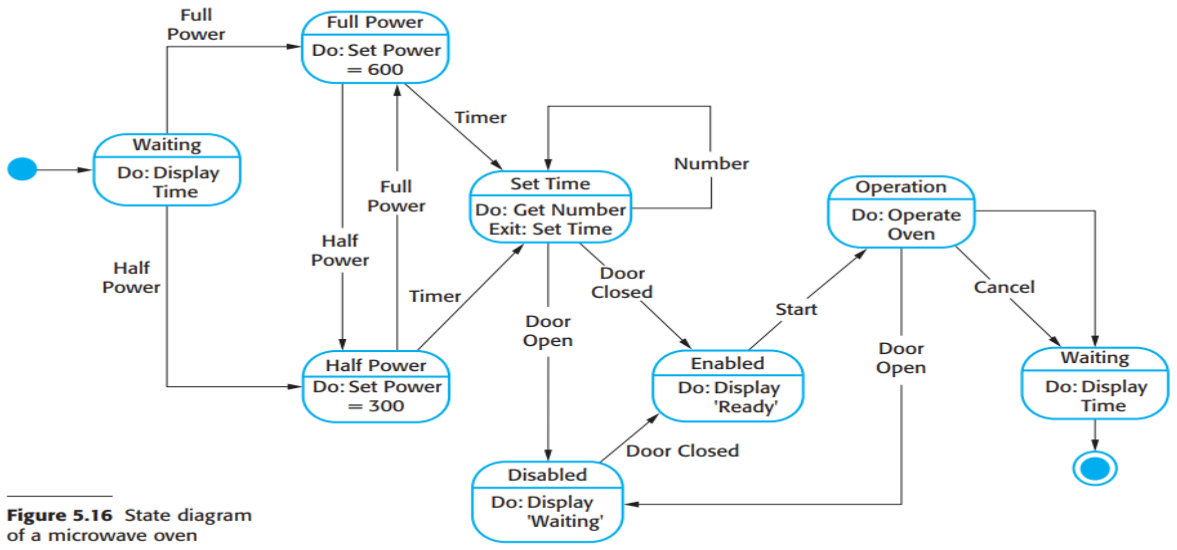


Figure 5.16 State diagram of a microwave oven

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows 'Half power'.
Full power	The oven power is set to 600 watts. The display shows 'Full power'.
Set time	The cooking time is set to the user's input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows 'Not ready'.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows 'Ready to cook'.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for five seconds. Oven light is on. Display shows 'Cooking complete' while buzzer is sounding.

Model-driven engineering

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.

Usage of model-driven engineering

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

Types of model

1. **A computation independent model (CIM)** – These model the important domain abstractions used in a system. CIMs are sometimes called domain models.
2. **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.
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.