Chapter 5 – System Modeling

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 modelling 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.

UML Origin

OOD in late 1980s and early 1990s:

- Different software development houses were using different notations.
- Methodologies were tied to notations.

UML developed in early 1990s to:

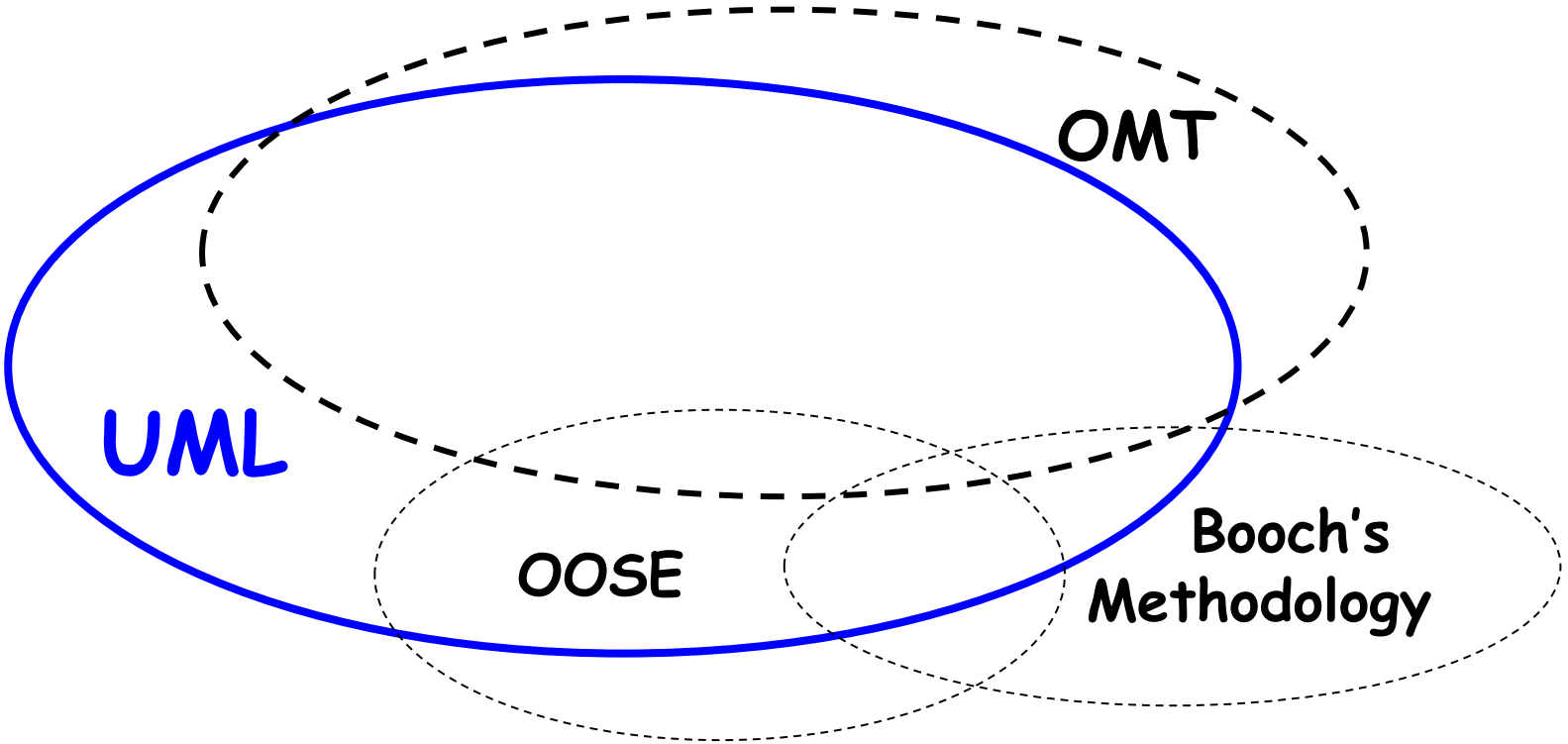
- Standardize the large number of object-oriented modelling notations

UML Lineology

Based Principally on:

- **OMT** [Rumbaugh 1991]
- **Booch's methodology**[Booch 1991]
- **OOSE** [Jacobson 1992]
- **Odell's methodology**[Odell 1992]
- **Shlaer and Mellor** [Shlaer 1992]

Different Object Modeling Techniques in UML



UML as A Standard

Adopted by Object Management Group (OMG) in 1997

OMG is an association of industries

Promotes consensus notations and techniques

Used outside software development

- Example car manufacturing

Why are UML Models Required?

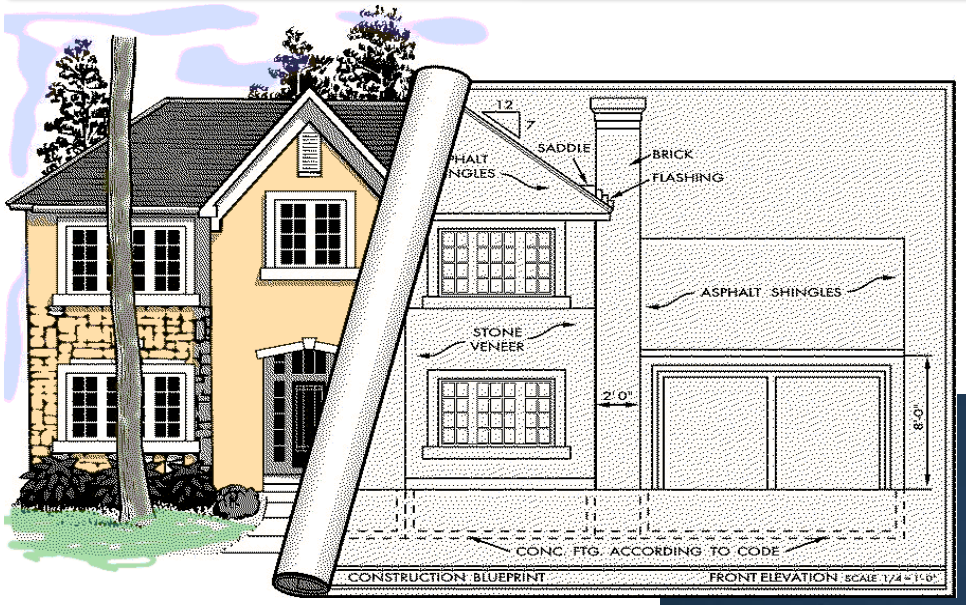
A model is an abstraction mechanism:

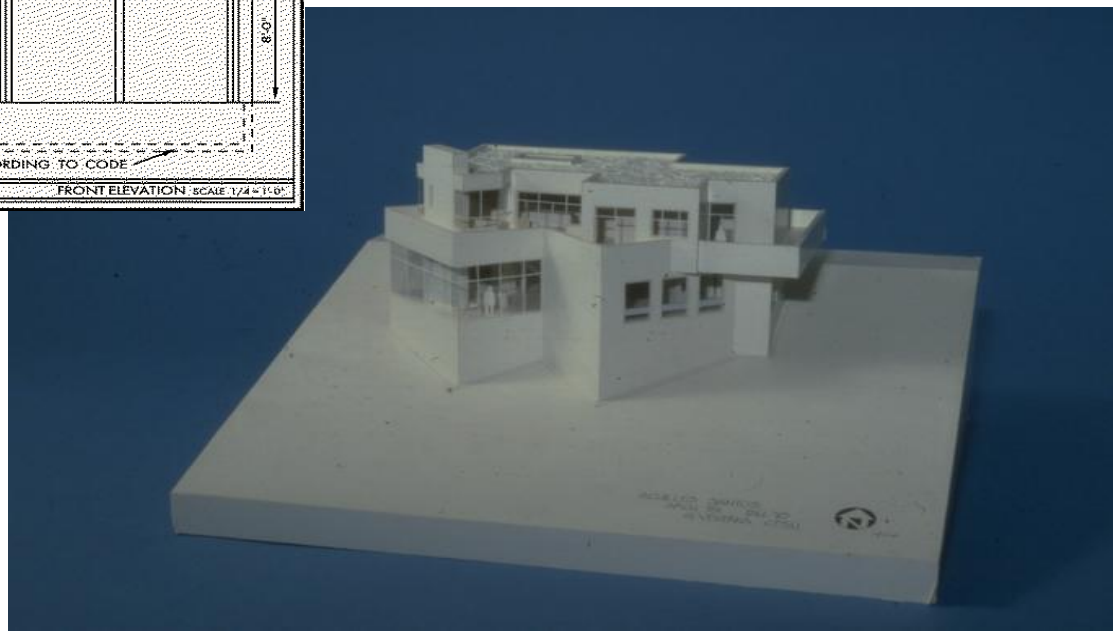
- Capture only important aspects and ignores the rest.
- Different models result when different aspects are ignored.
- An effective mechanism to handle complexity.

UML is a graphical modelling tool

Easy to understand and construct

Modeling a House





UML Diagrams

Nine diagrams are used to capture different views of a system.

Views:

- Provide different perspectives of a software system.

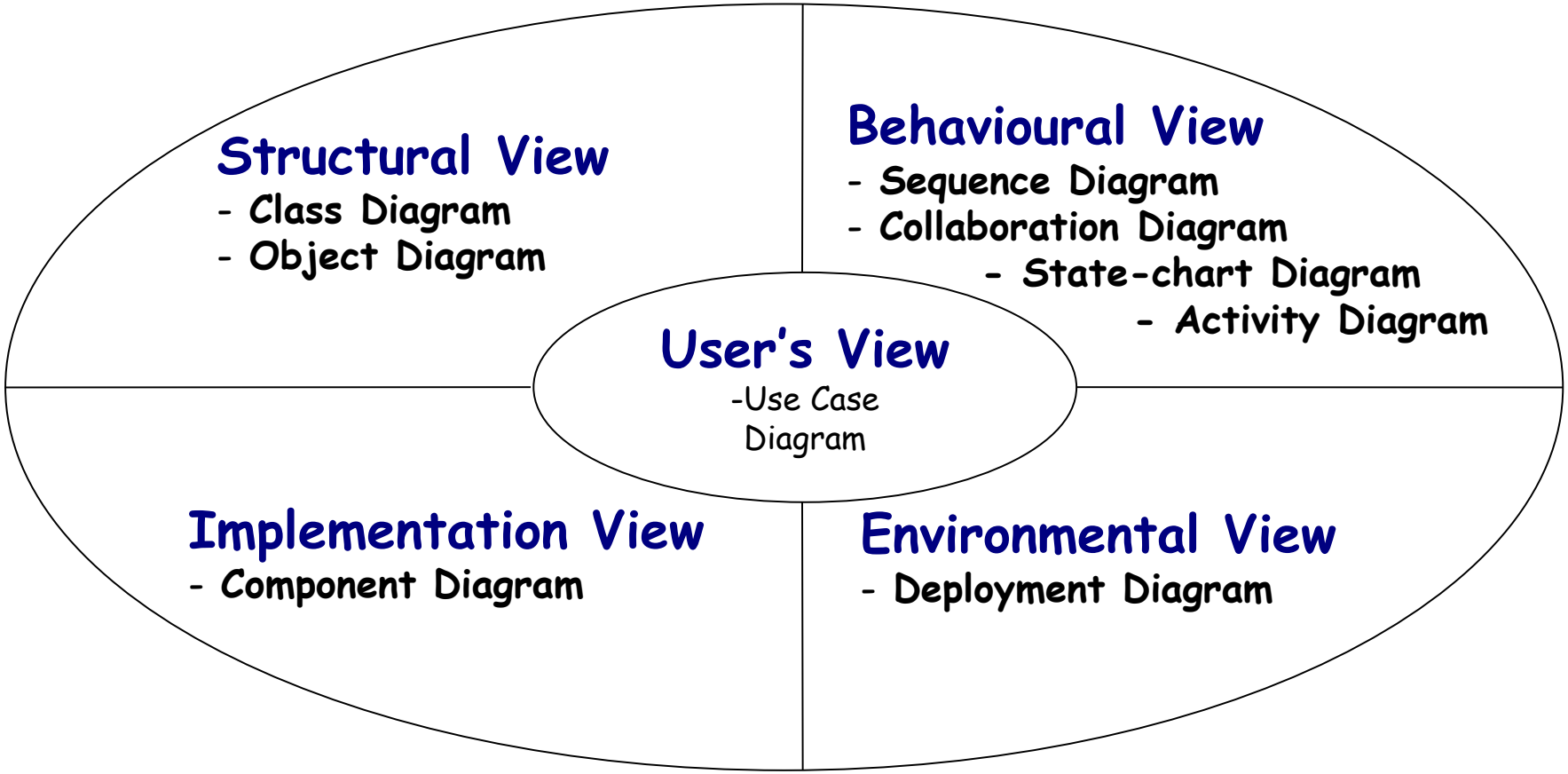
Diagrams can be refined to get the actual implementation of a system.

UML Model Views

Views of a system:

- User's view
- Structural view
- Behavioral view
- Implementation view
- Environmental view

UML Diagrams



Diagrams and views in UML

Are All Views Required for Developing A Typical System?

NO

Use case diagram, class diagram and one of the interaction diagram for a simple system

State chart diagram required to be developed when a class state changes

However, when states are only one or two, state chart model becomes trivial

Deployment diagram in case of large number of hardware components used to develop the system

Use Case Model

- Consists of set of “use cases”
- An important analysis and design artifact
- The central model:
  - Other models must confirm to this model
  - Not really an object-oriented model
  - Represents a functional or process model

Use Cases

- Different ways in which a system can be used by the users
- Corresponds to the high-level requirements
- Represents transaction between the user and the system
- Defines external behavior without revealing internal structure of system
- Set of related scenarios tied together by a common goal.

Use Cases

- Normally, use cases are independent of each other
- Implicit dependencies may exist
- **Example**: In Library Automation System, renew-book & reserve-book are independent use cases.
  - But in actual implementation of renew-book: a check is made to see if any book has been reserved using reserve-book.

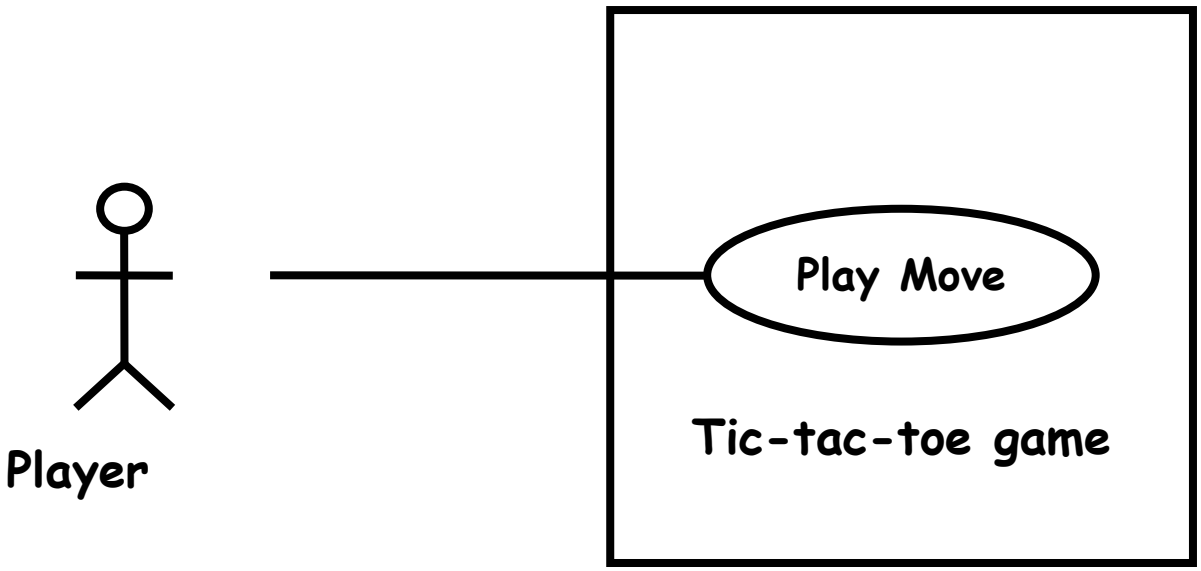
Example Use Cases

- For library information system
  - issue-book
  - query-book
  - return-book
  - create-member
  - add-book, etc.

Representation of Use Cases

- Represented by use case diagram
- A use case is represented by an ellipse
- System boundary is represented by a rectangle
- Users are represented by stick person icons (actor)
- Communication relationship between actor and use case by a line
- External system by a stereotype

An Example Use Case Diagram



Use case model

Why Develop A Use Case Diagram?

Serves as requirements specification

How are actor identification useful in software development:

- User identification helps in implementing appropriate interfaces for different categories of users
- Another use in preparing appropriate documents (e.g. **user's manual**).

Factoring Use Cases

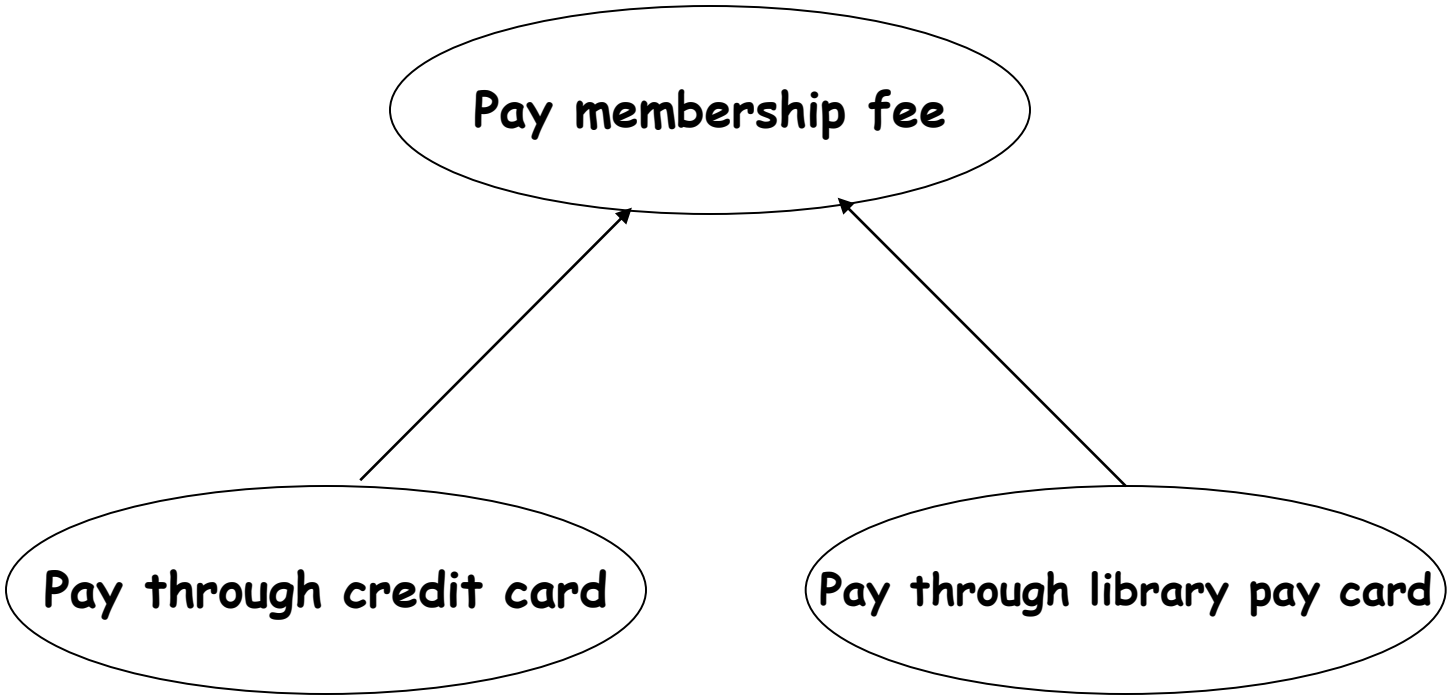
Two main reasons for factoring:

- Complex use cases need to be factored into simpler use cases
- To represent common behavior across different use cases

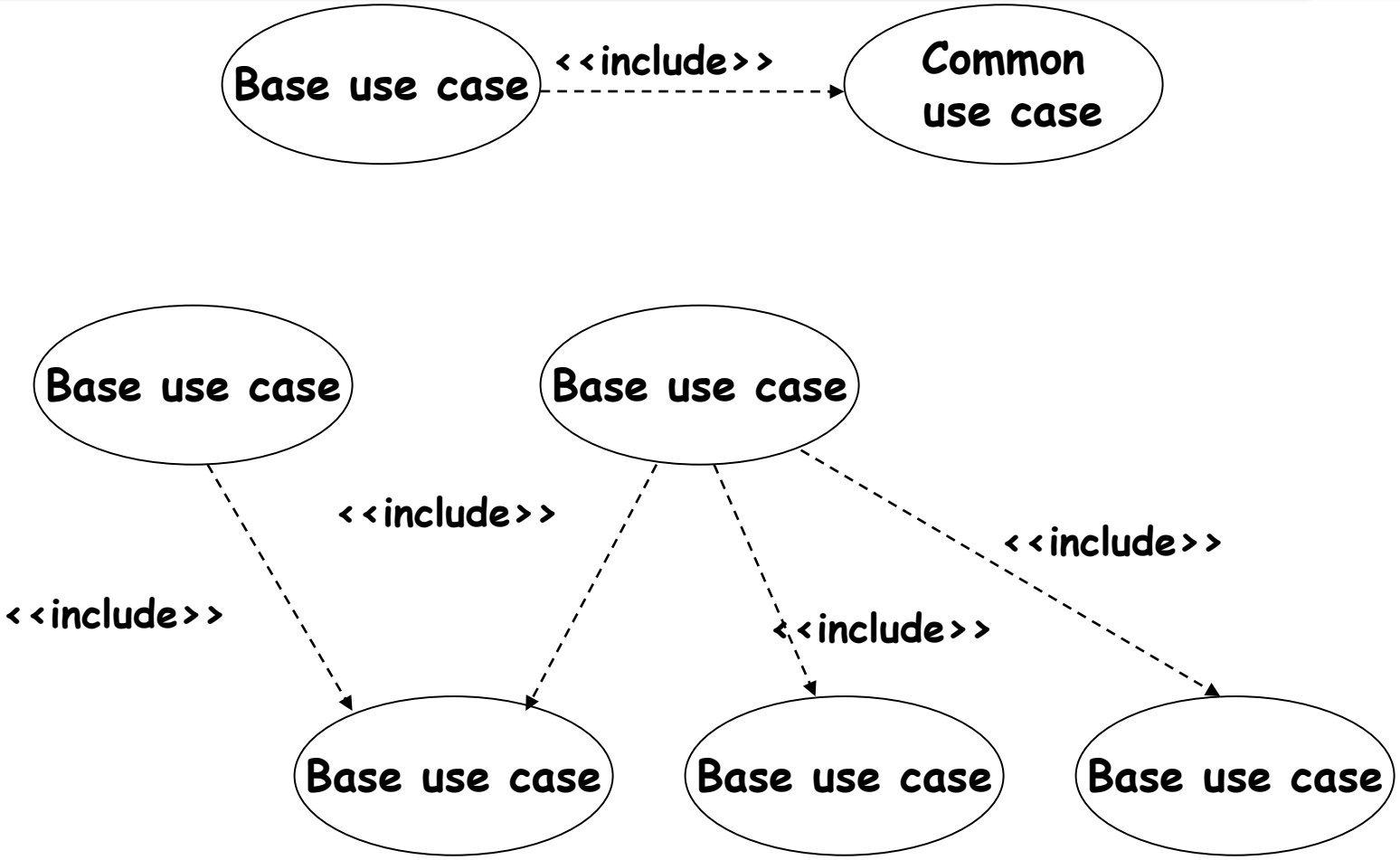
Three ways of factoring:

- Generalization
- Includes
- Extends

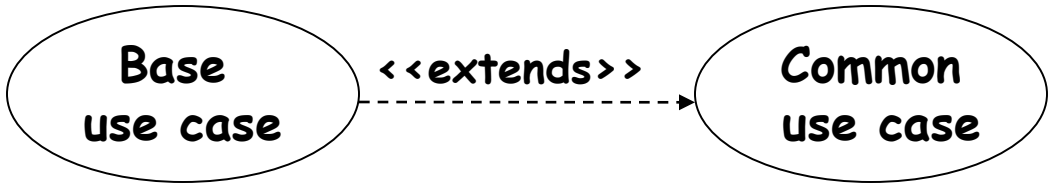
Factoring Use Cases Using Generalization



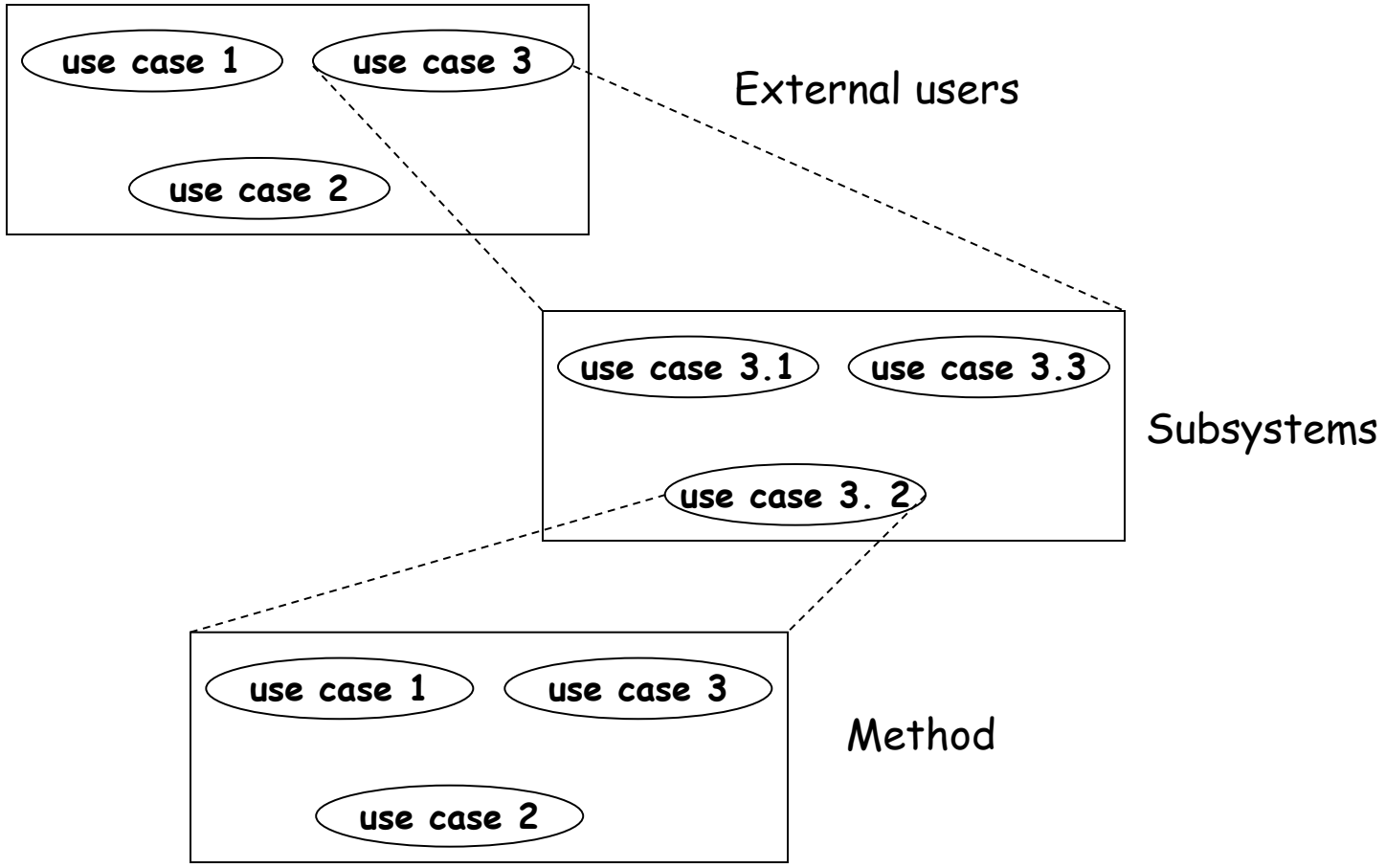
Factoring Use Cases Using Includes



Factoring Use Cases Using Extends



Hierarchical Organization of Use Cases



Class Diagram

Describes static structure of a system

Main constituents are classes and their relationships:

- Generalization
- Aggregation
- Association
- Various kinds of dependencies

Class Diagram

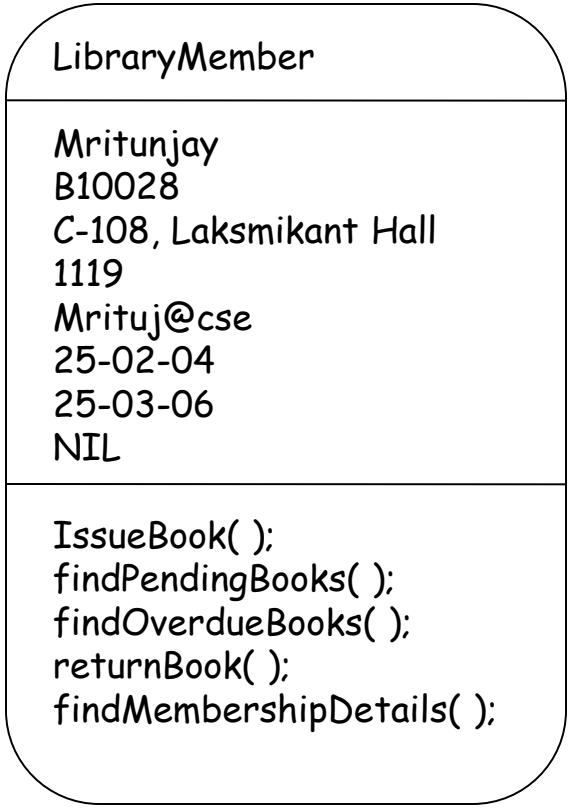
Entities with common features, i.e. attributes and operations

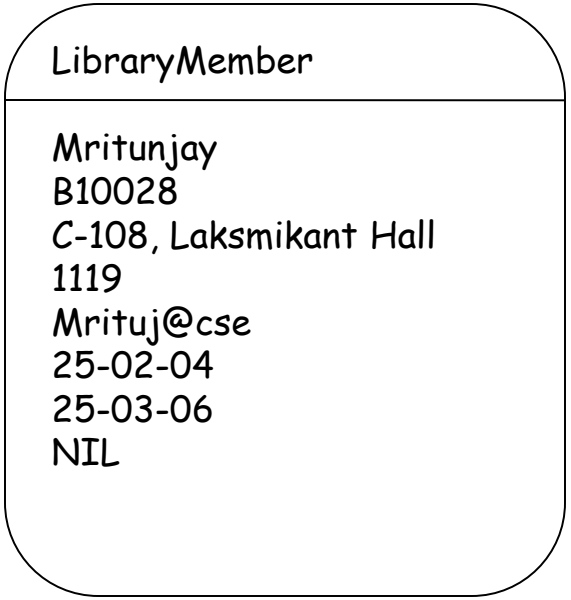
Classes are represented as solid outline rectangle with compartments

Compartments for name, attributes, and operations.

Attribute and operation compartments are optional depending on the purpose of a diagram.

Object Diagram





Different representations of the LibraryMember object

Interaction Diagram

- Models how groups of objects collaborate to realize some behaviour
- Typically each interaction diagram realizes behaviour of a single use case

Interaction Diagram

Two kinds: Sequence and Collaboration diagrams.

Two diagrams are equivalent

- Portray different perspectives

These diagrams play a very important role in the design process.

Sequence Diagram

Shows interaction among objects as a two-dimensional chart

Objects are shown as boxes at top

If object created during execution then shown at appropriate place

Objects existence are shown as dashed lines (lifeline)

Objects activeness, shown as a rectangle on lifeline

Sequence Diagram Cont…

Messages are shown as arrows

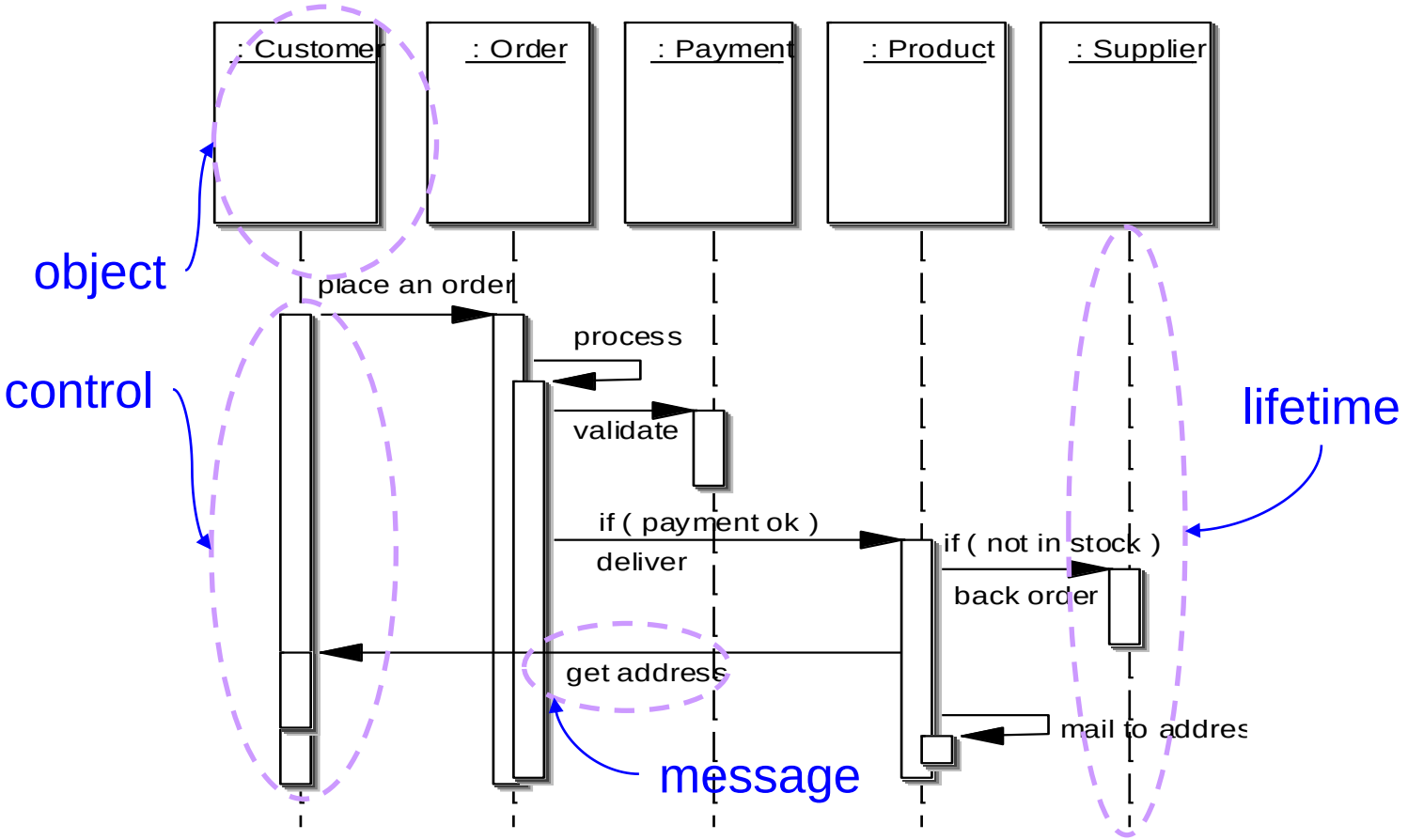
Each message labelled with corresponding message name

Each message can be labelled with some control information

Two types of control information

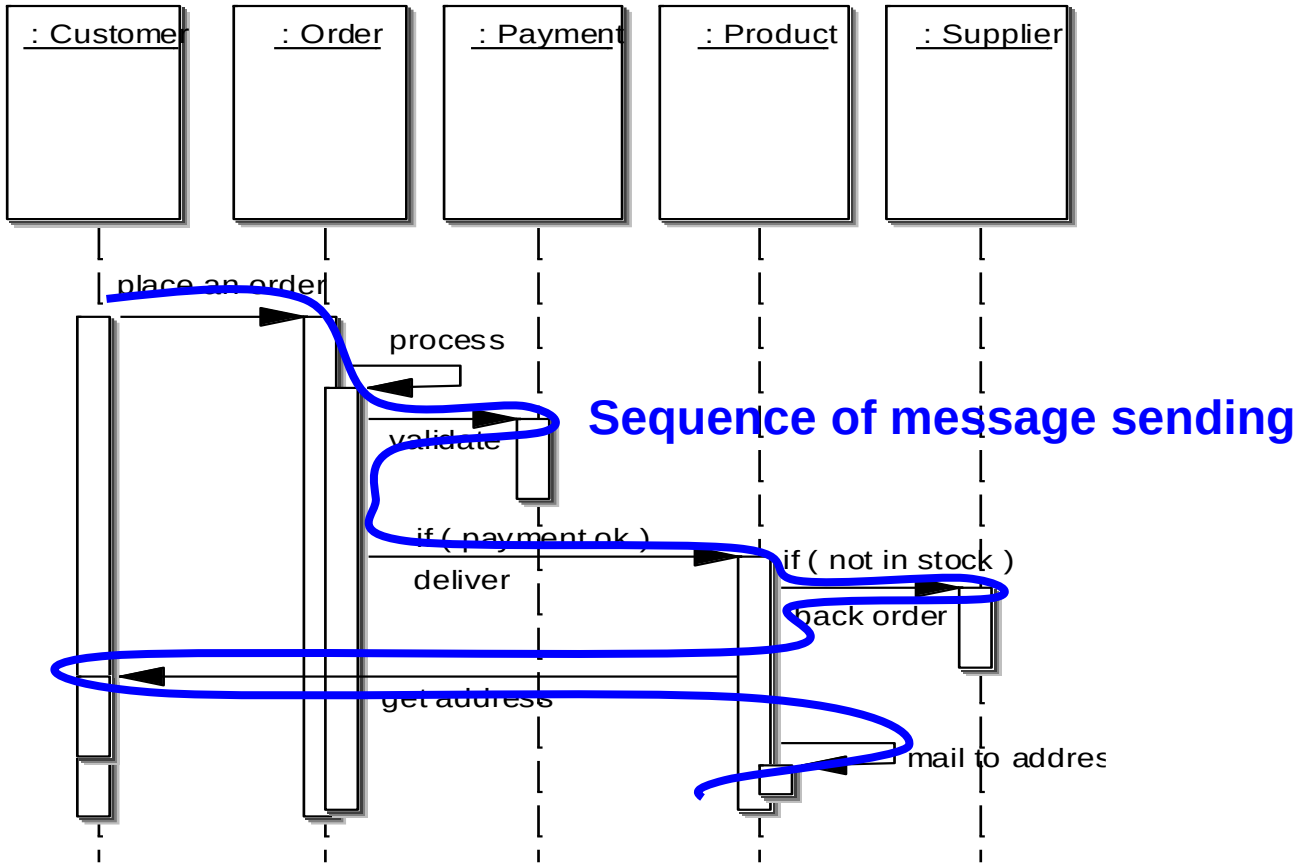
- condition ([])
- iteration (*)

Elements of a Sequence Diagram

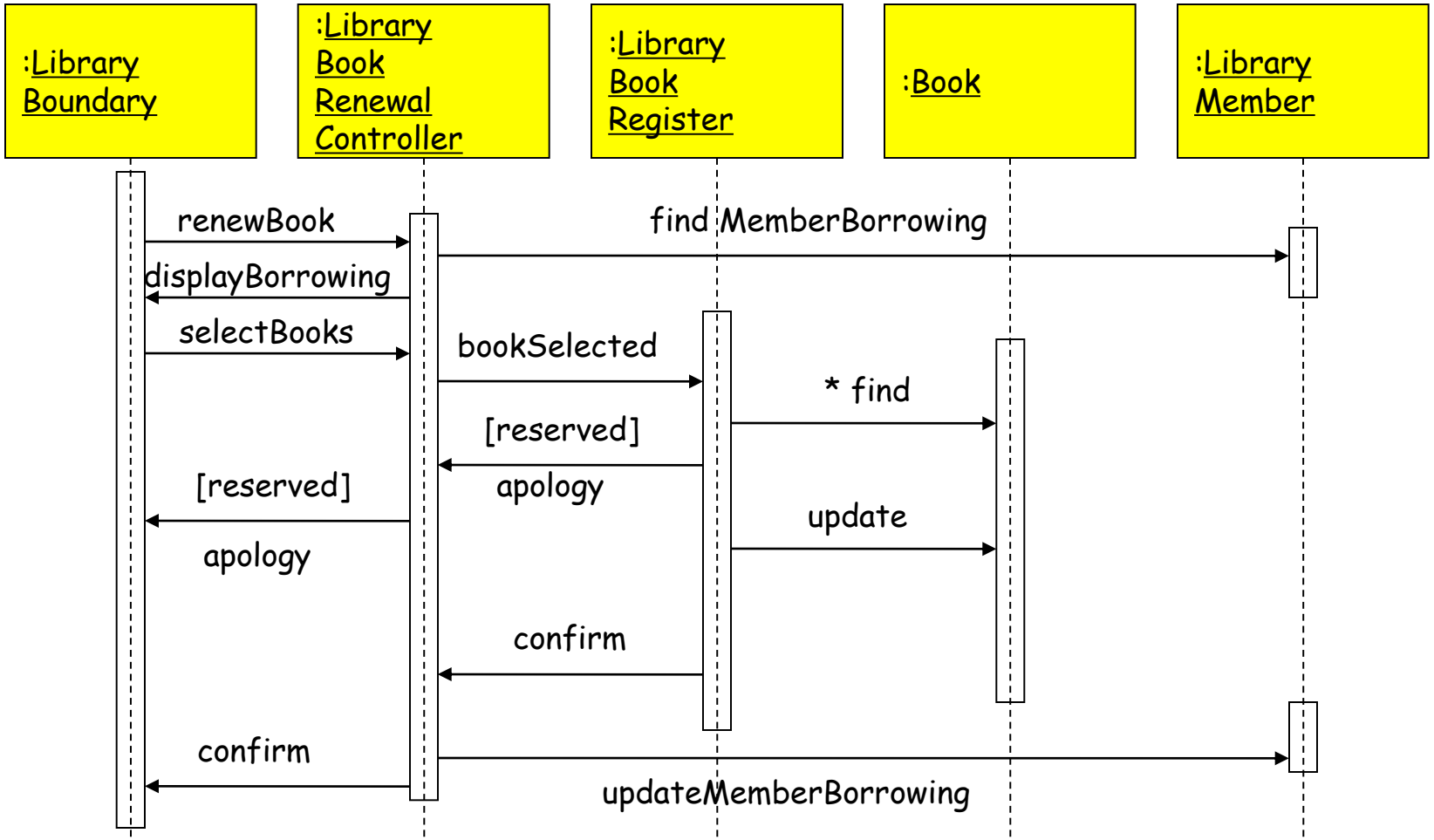


Example


Software Engineering
Ian Sommerville


: Customer
: Order
: Payment
: Product
: Supplier
place an order
process
validate
if (payment ok)
deliver
if (not in stock)
back order
get address
mail to addres
Sequence of message sending

An Example of a Sequence Diagram



Sequence Diagram for the renew book use case

Collaboration Diagram

Shows both structural and behavioural aspects

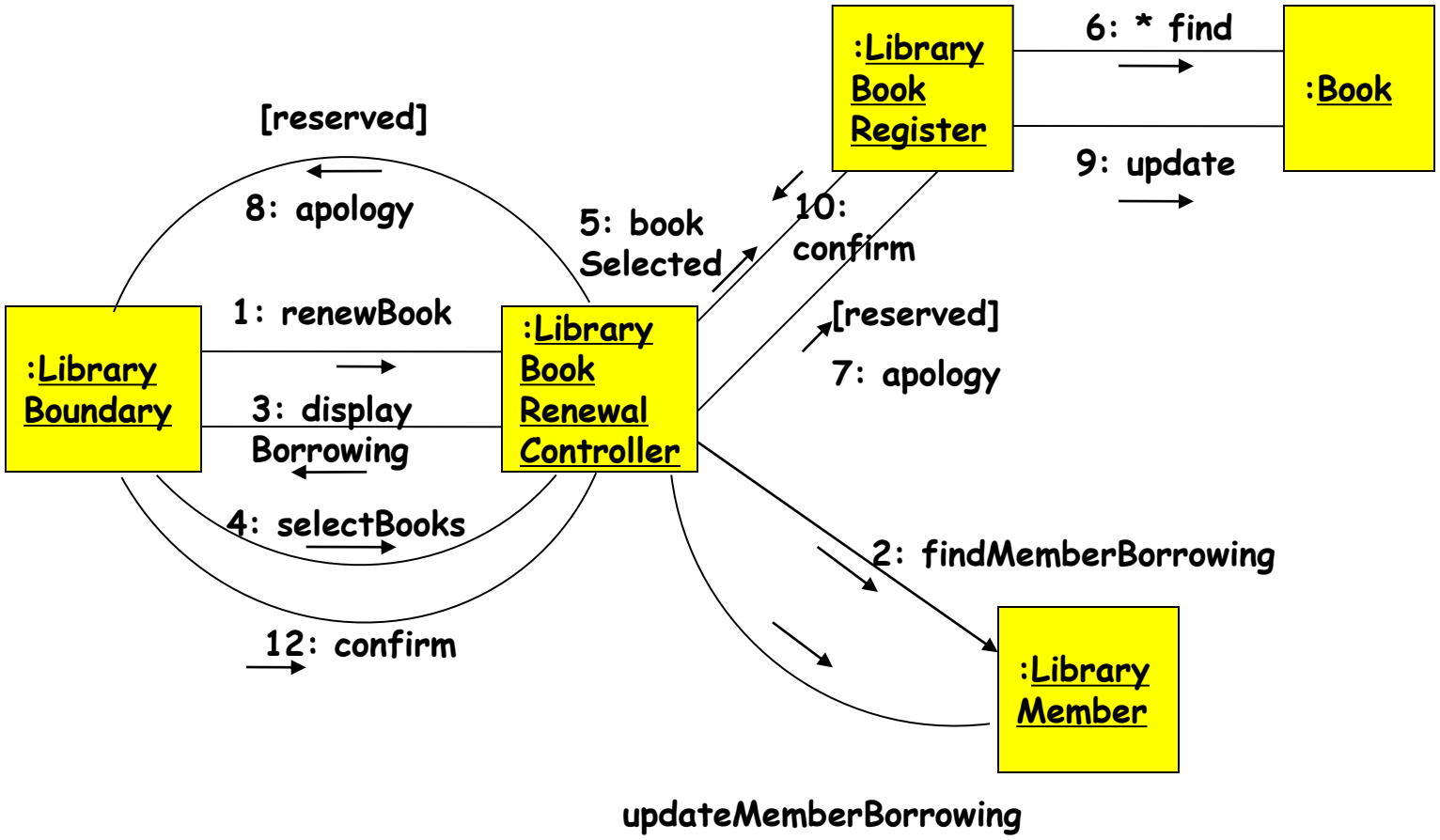
Objects are collaborator, shown as boxes

Messages between objects shown as a solid line

A message is shown as a labelled arrow placed near the link

Messages are prefixed with sequence numbers to show relative sequencing

An Example of A Collaboration Diagram



Collaboration Diagram for the renew book use case

Activity Diagram

- Not present in earlier modelling techniques:
  - Possibly based on event diagram of Odell [1992]
- Represents processing activity, may not correspond to methods
- Activity is a state with an internal action and one/many outgoing transitions
- Somewhat related to flowcharts

Activity Diagram vs Flow Chart

Can represent parallel activity and synchronization aspects

Swim lanes can be used to group activities based on who is performing them

Example: academic department vs. hostel

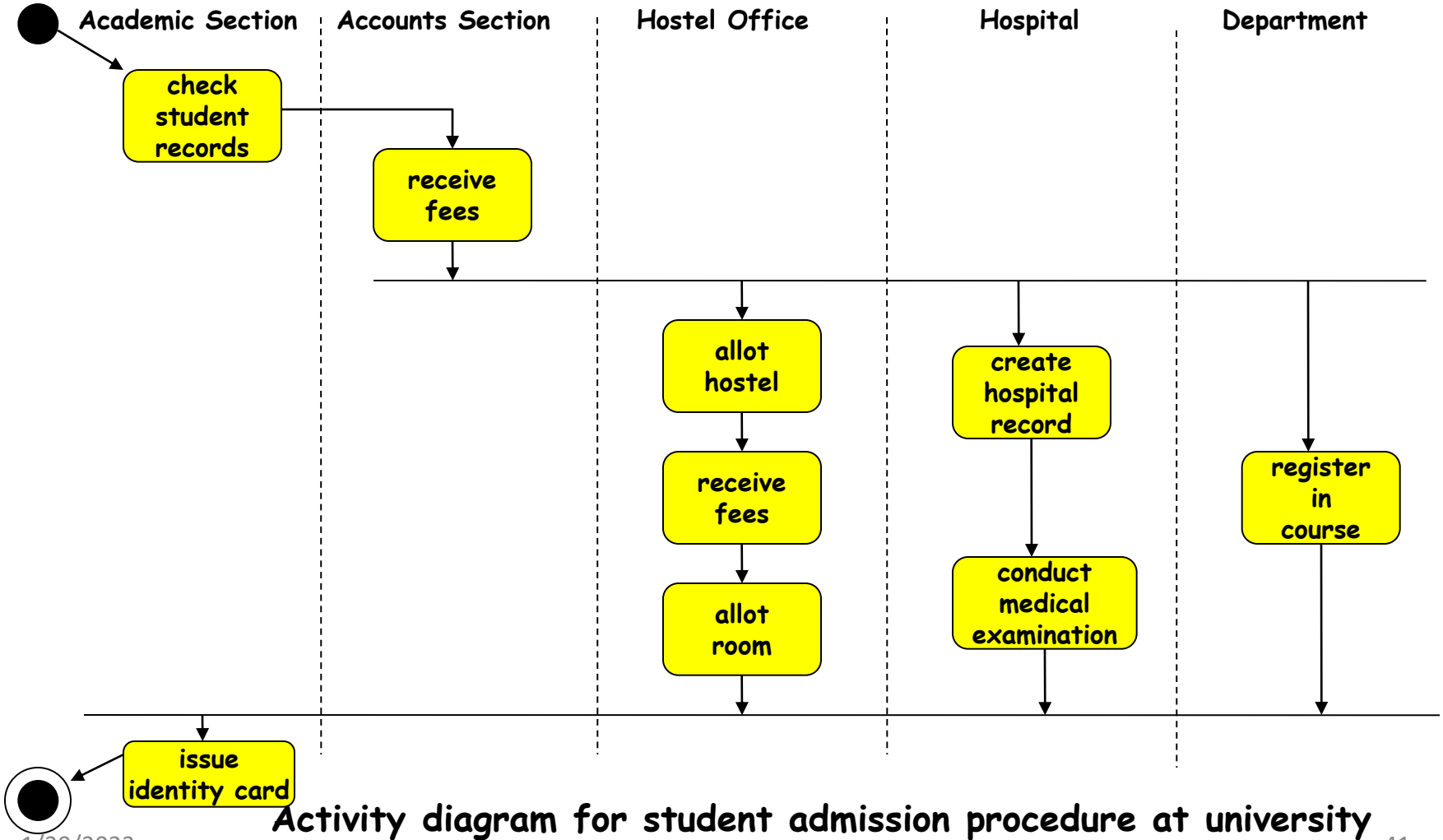
Activity Diagram

Normally employed in business process modelling.

Carried out during requirements analysis and specification stage.

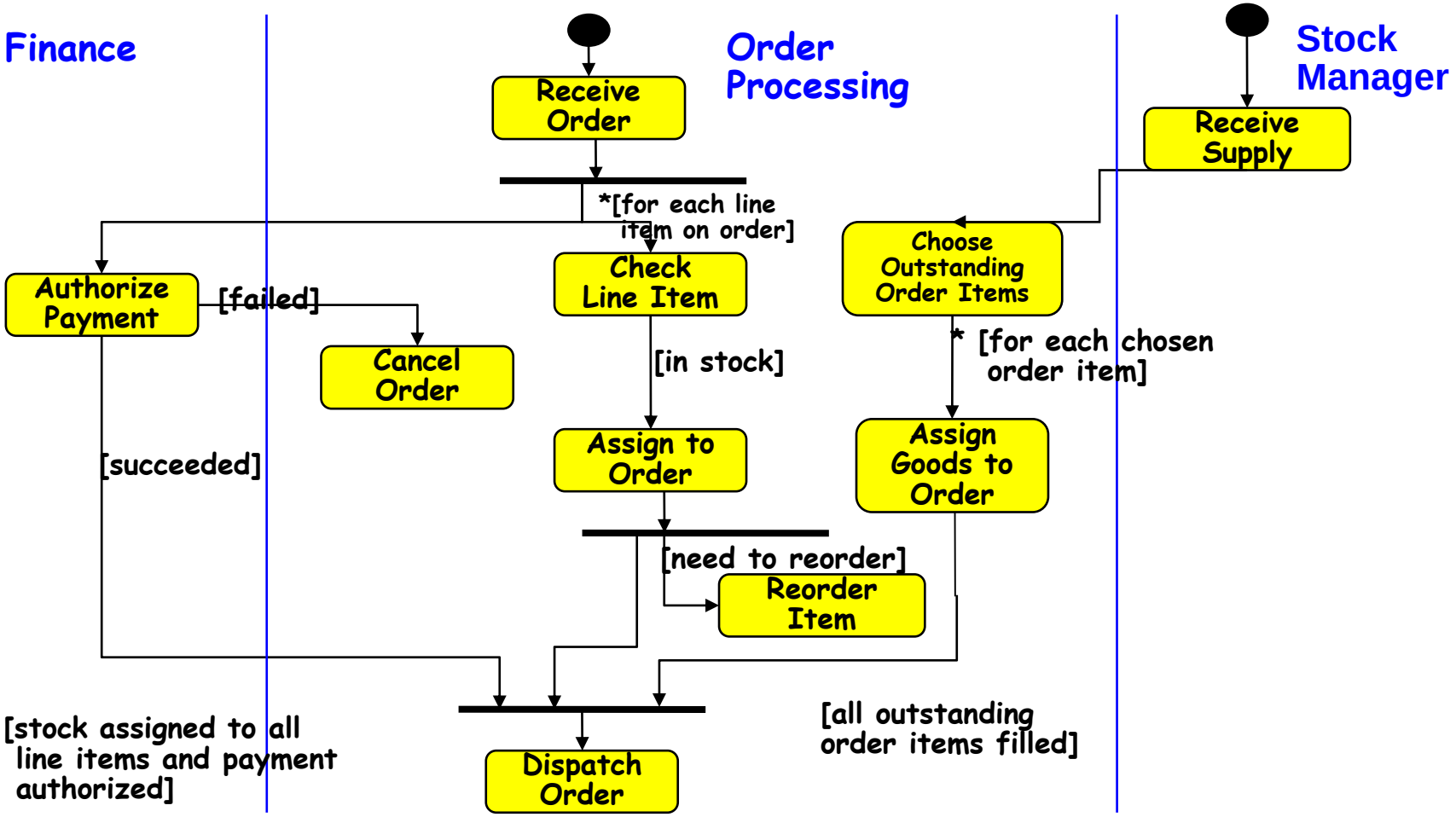
Can be used to develop interaction diagrams.

An Example of An Activity Diagram



Activity diagram for student admission procedure at university

Activity Diagram: Example 2



State Chart Diagram

Based on the work of **David Harel** [1990]

Model how the state of an object changes in its lifetime

Based on finite state machine (FSM) formalism

State Chart Diagram

State chart avoids the problem of state explosion of FSM.

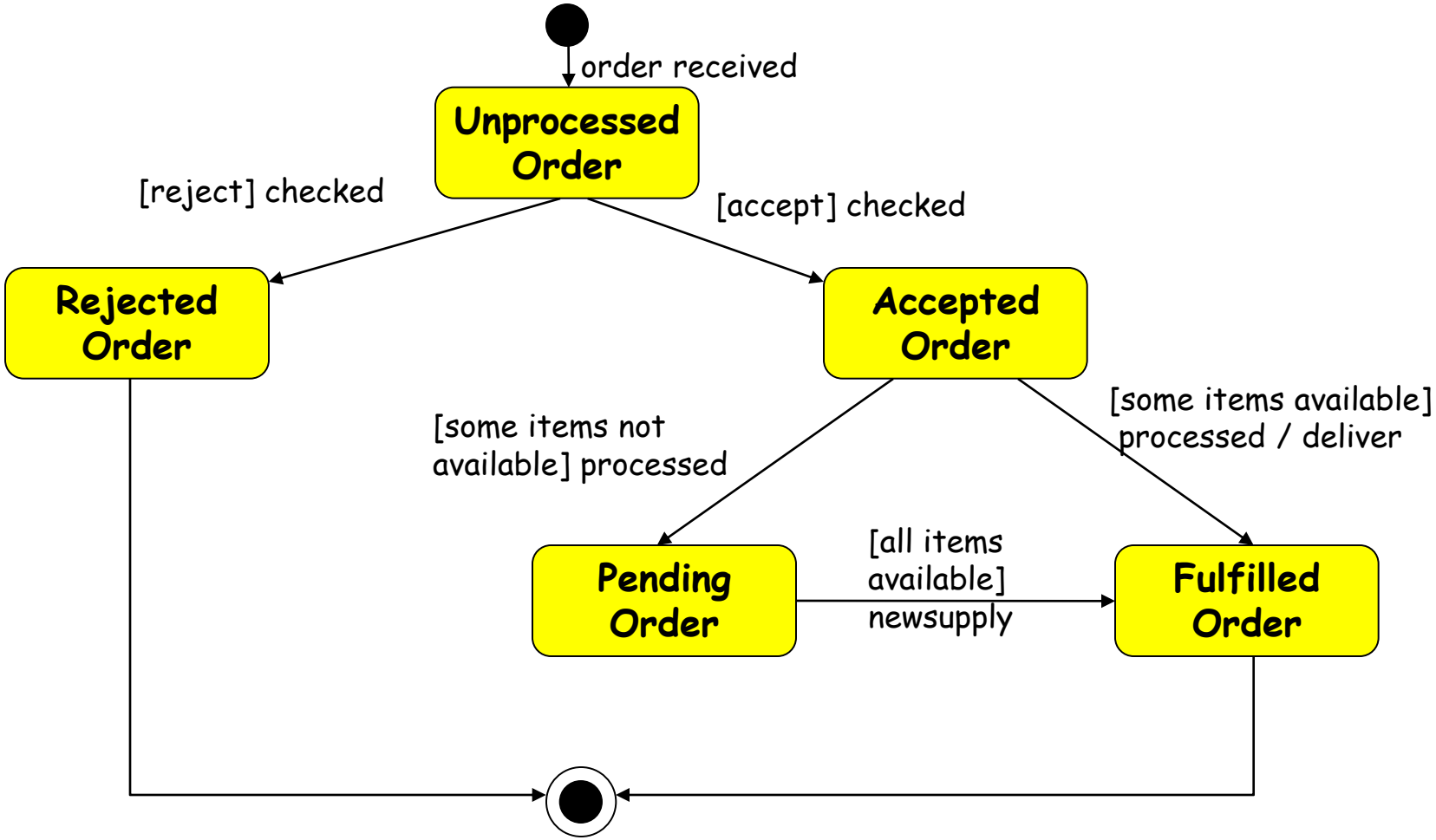
Hierarchical model of a system:

- Represents composite **nested** states

State Chart Diagram

- Elements of state chart diagram
- Initial State: A filled circle
- Final State: A filled circle inside a larger circle
- State: Rectangle with rounded corners
- Transitions: Arrow between states, also boolean logic condition (guard)

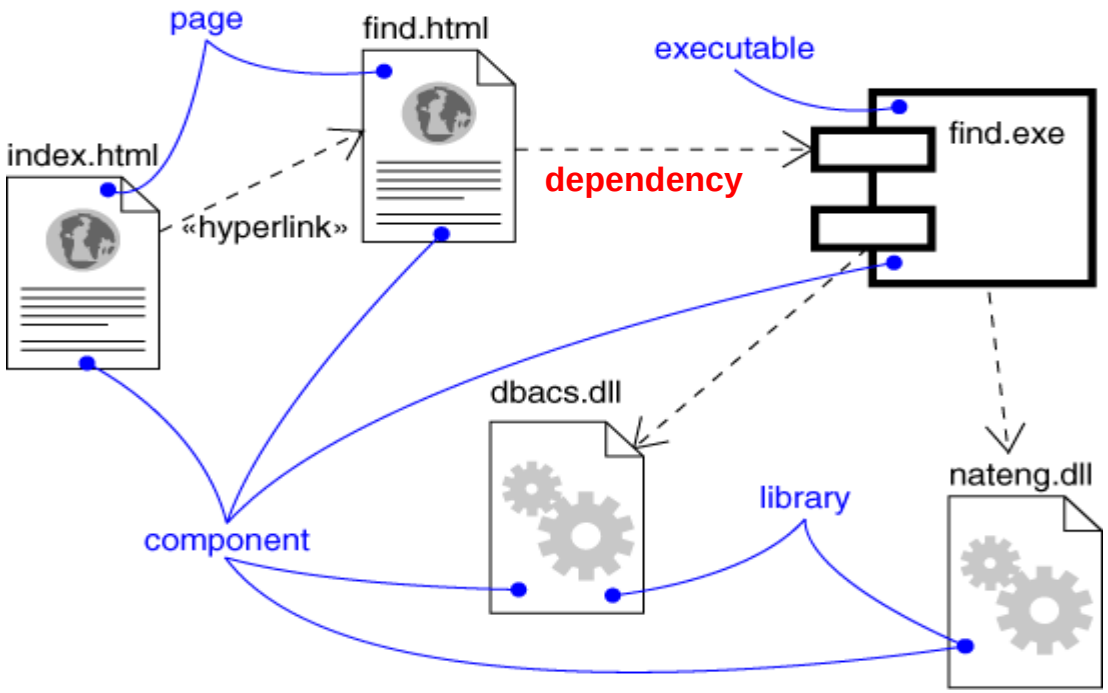
An Example of A State Chart Diagram



Example: State chart diagram for an order object

Component Diagram

Captures the physical structure of the implementation (code components)



Components:

- **Executables**
- **Library**
- **Table**
- **File**
- **Document**

Component Diagram

Captures the physical structure of the implementation

Built as part of architectural specification

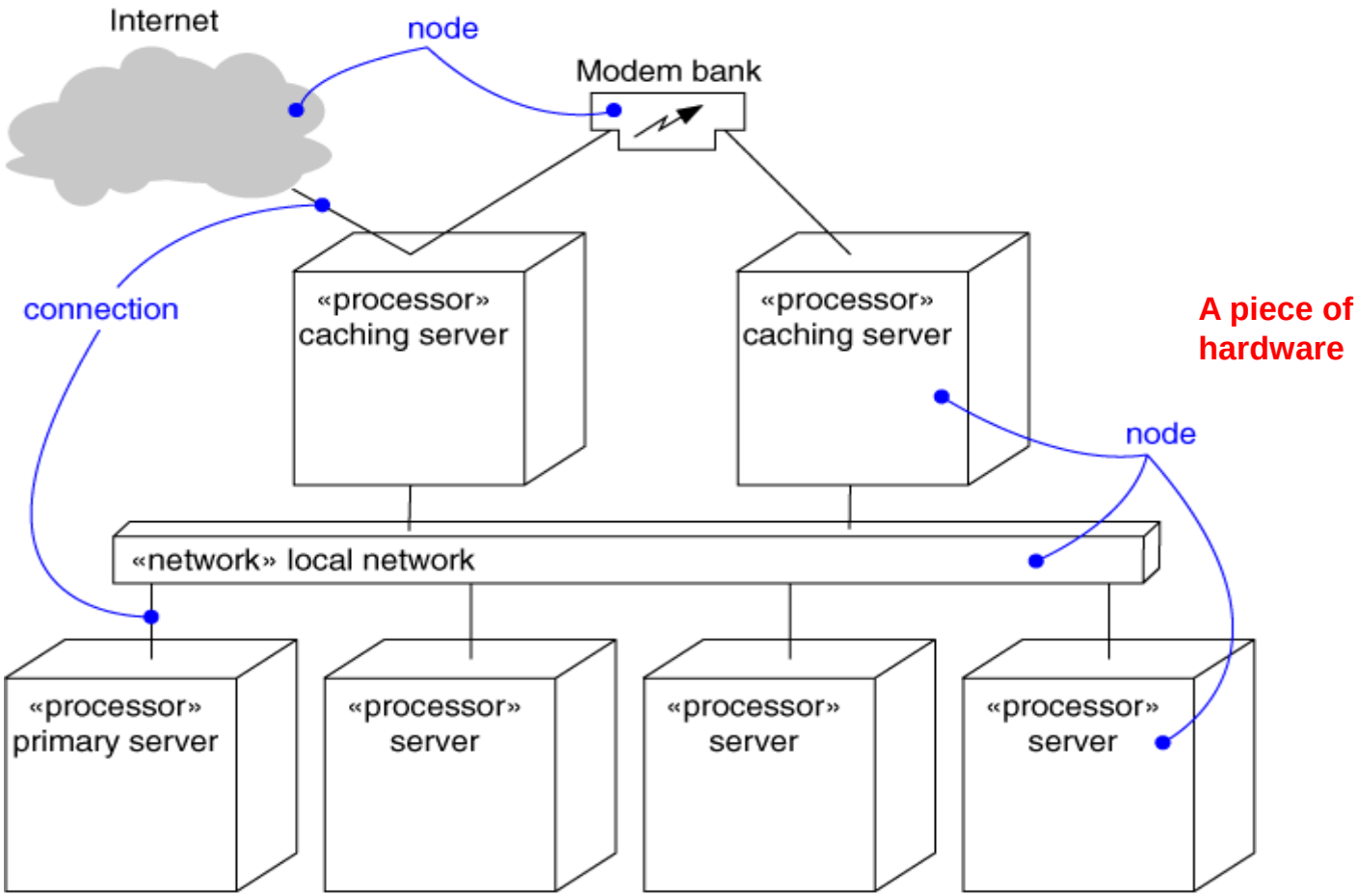
Purpose

- Organize source code
- Construct an executable release
- Specify a physical database

Developed by architects and programmers

Deployment Diagram

✧ Captures the topology of a system's hardware



Context models

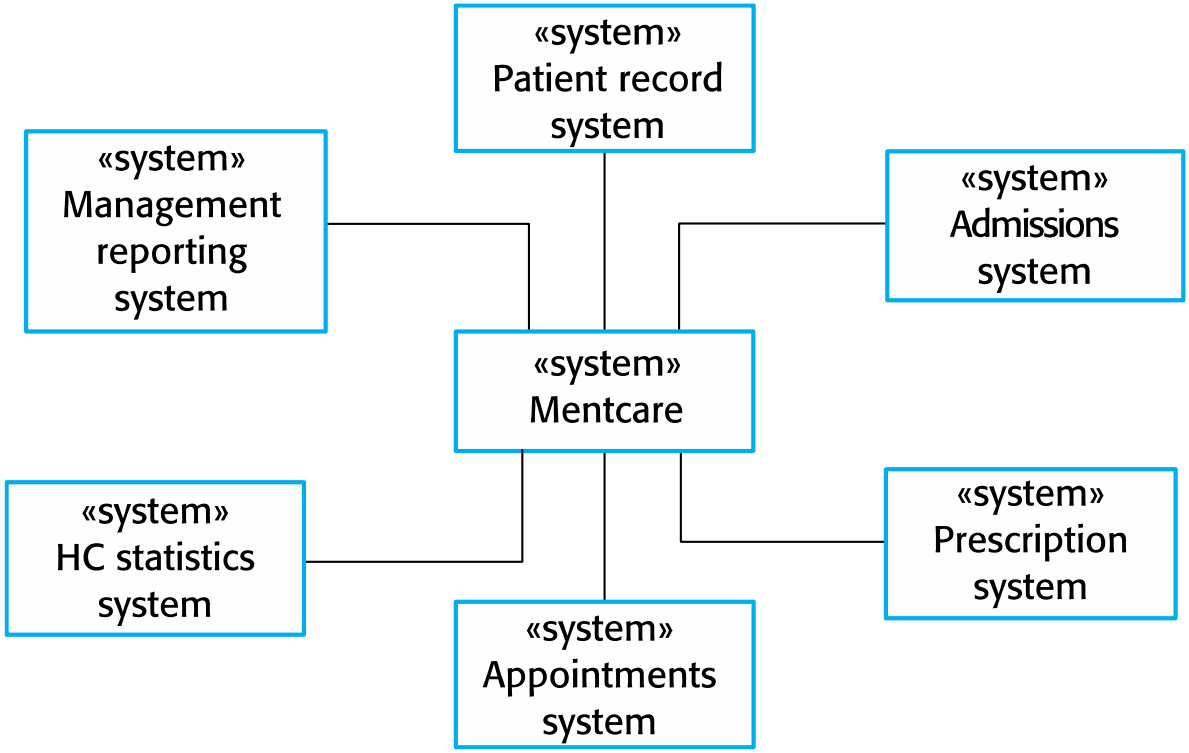
Context models

- Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- Architectural models show the system and its relationship with other systems.

System boundaries

- 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.

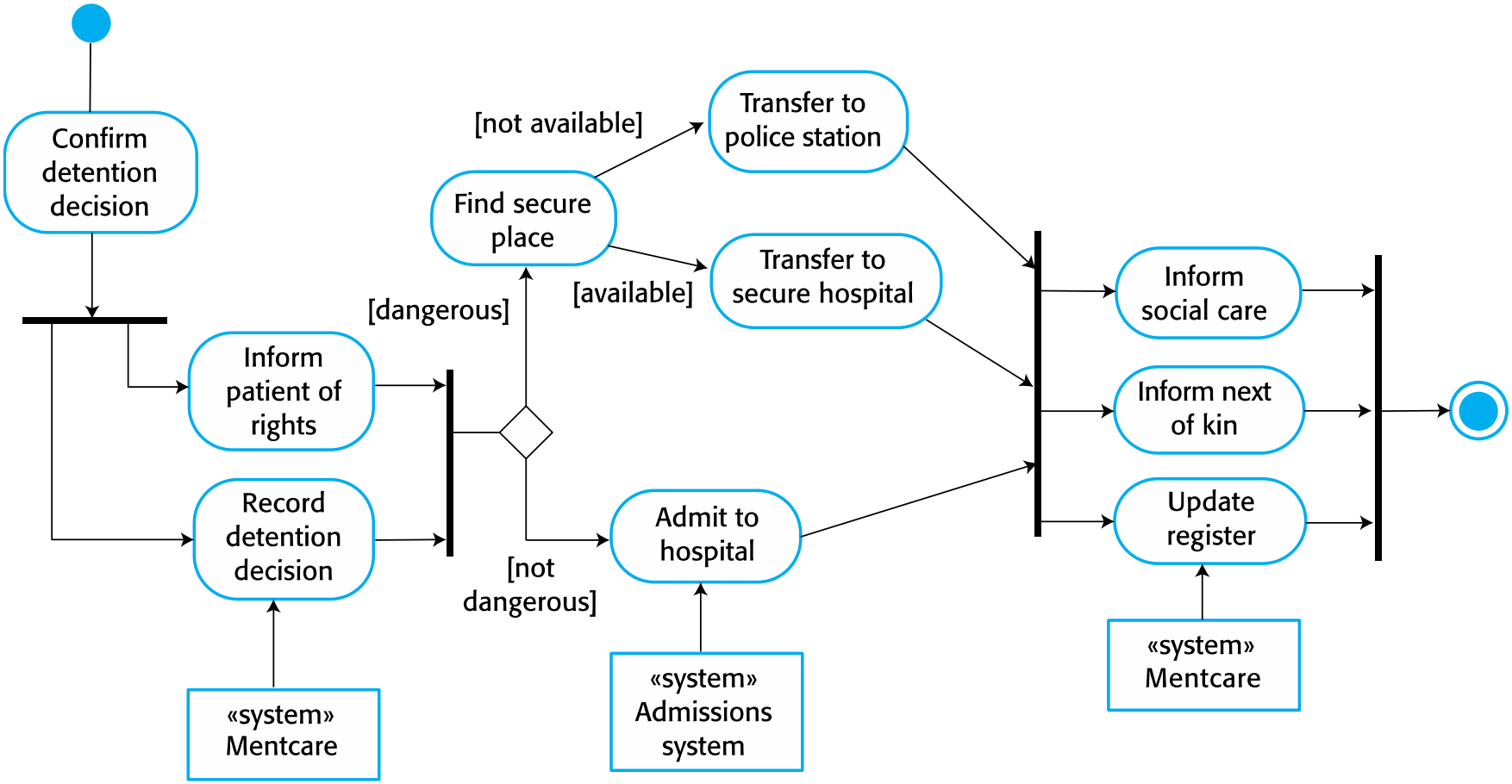
The context of the Mentcare system



Process perspective

- 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.

Process model of involuntary detention



Interaction models

Interaction models

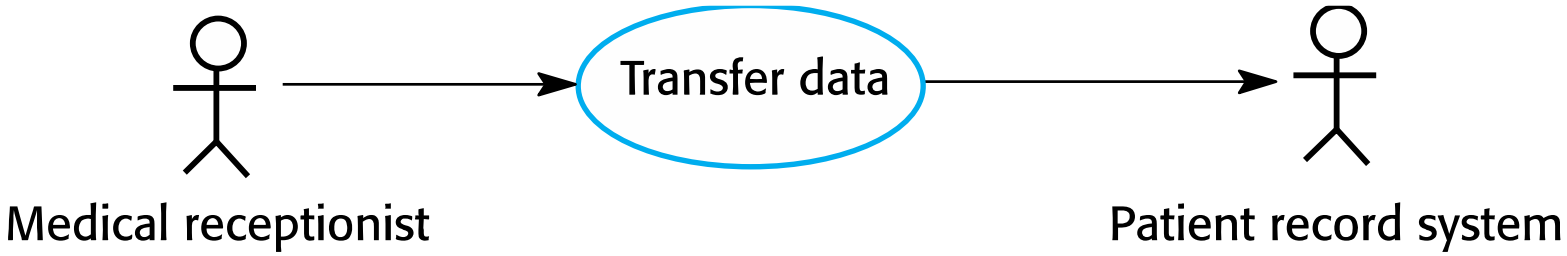
- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system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

Use case 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 diagramatically to provide an overview of the use case and in a more detailed textual form.

Transfer-data use case

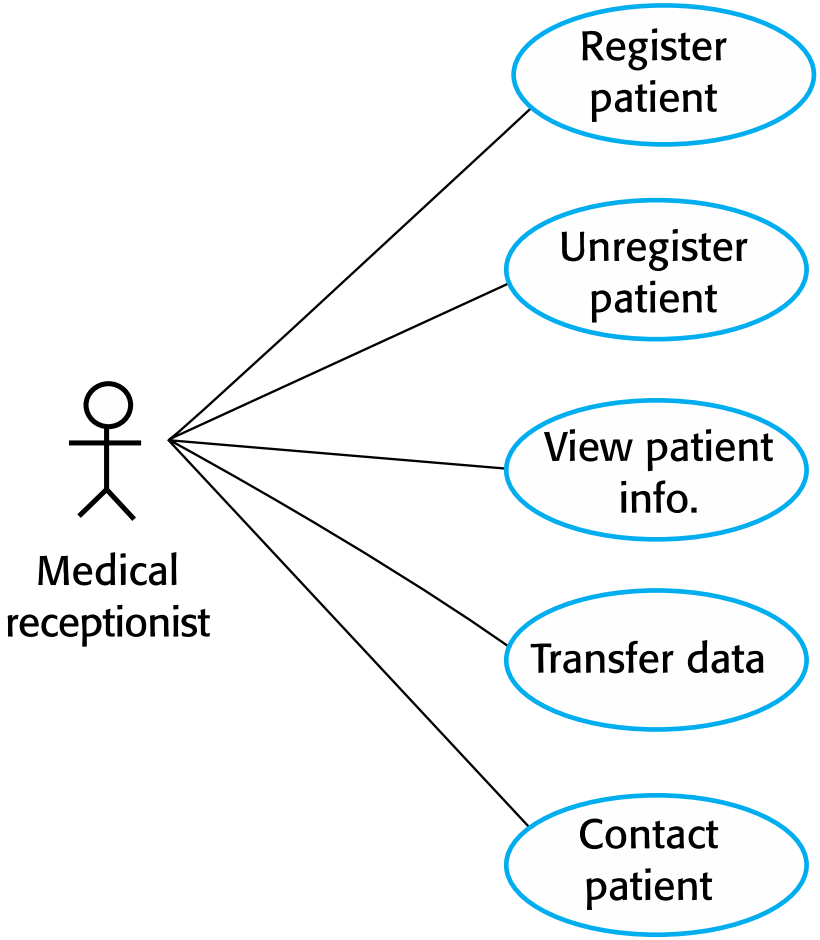
✧ A use case in the Mentcare system



Tabular description of the ‘Transfer data’ use-case

MHC-PMS: Transfer data	
Actors	Medical receptionist, patient records system (PRS)
Description	A receptionist may transfer data from the Mentcase system 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.
Data	Patient’s personal information, treatment summary
Stimulus	User command issued by medical receptionist
Response	Confirmation that PRS has been updated
Comments	The receptionist must have appropriate security permissions to access the patient information and the PRS.

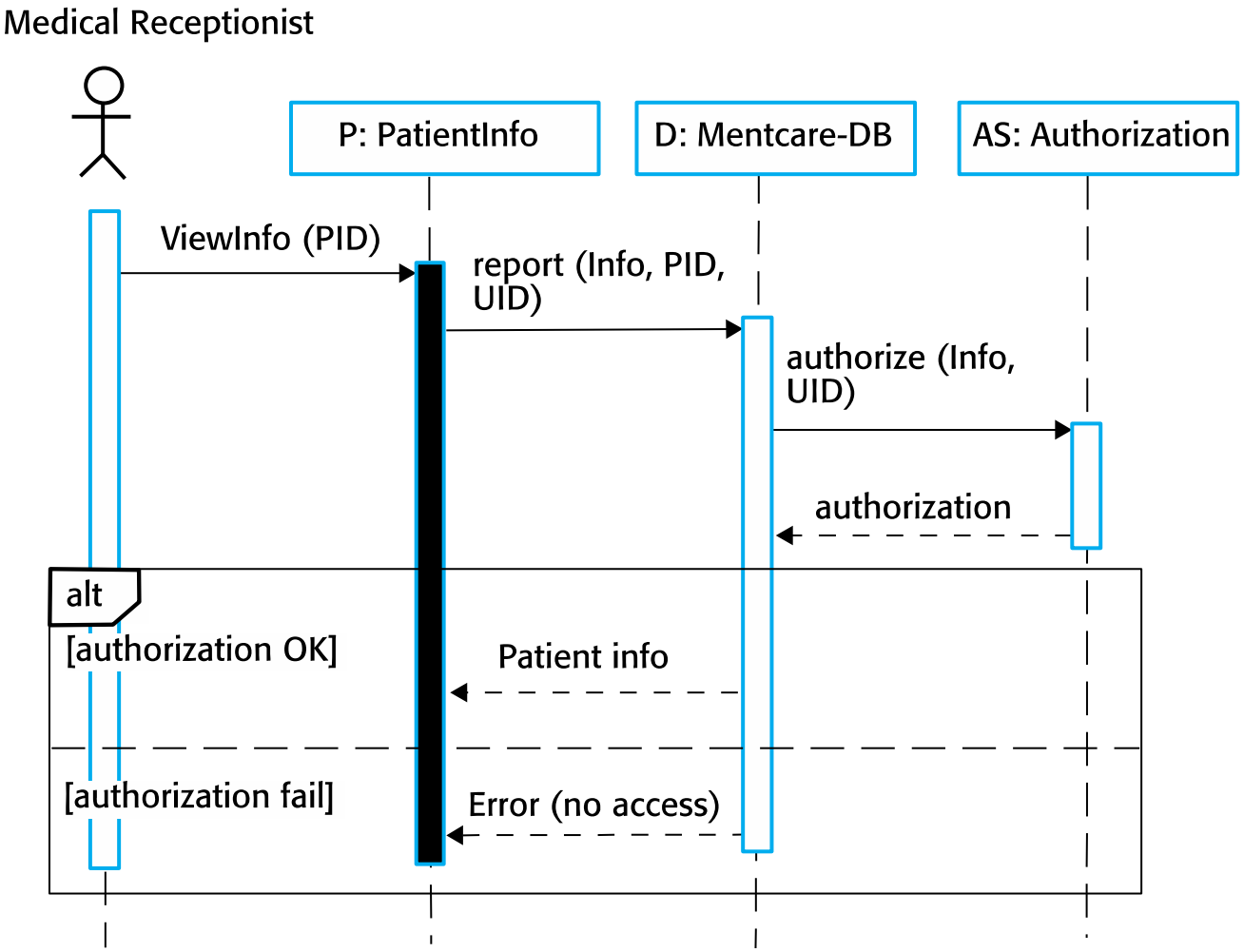
Use cases in the Mentcare system involving the role ‘Medical Receptionist’

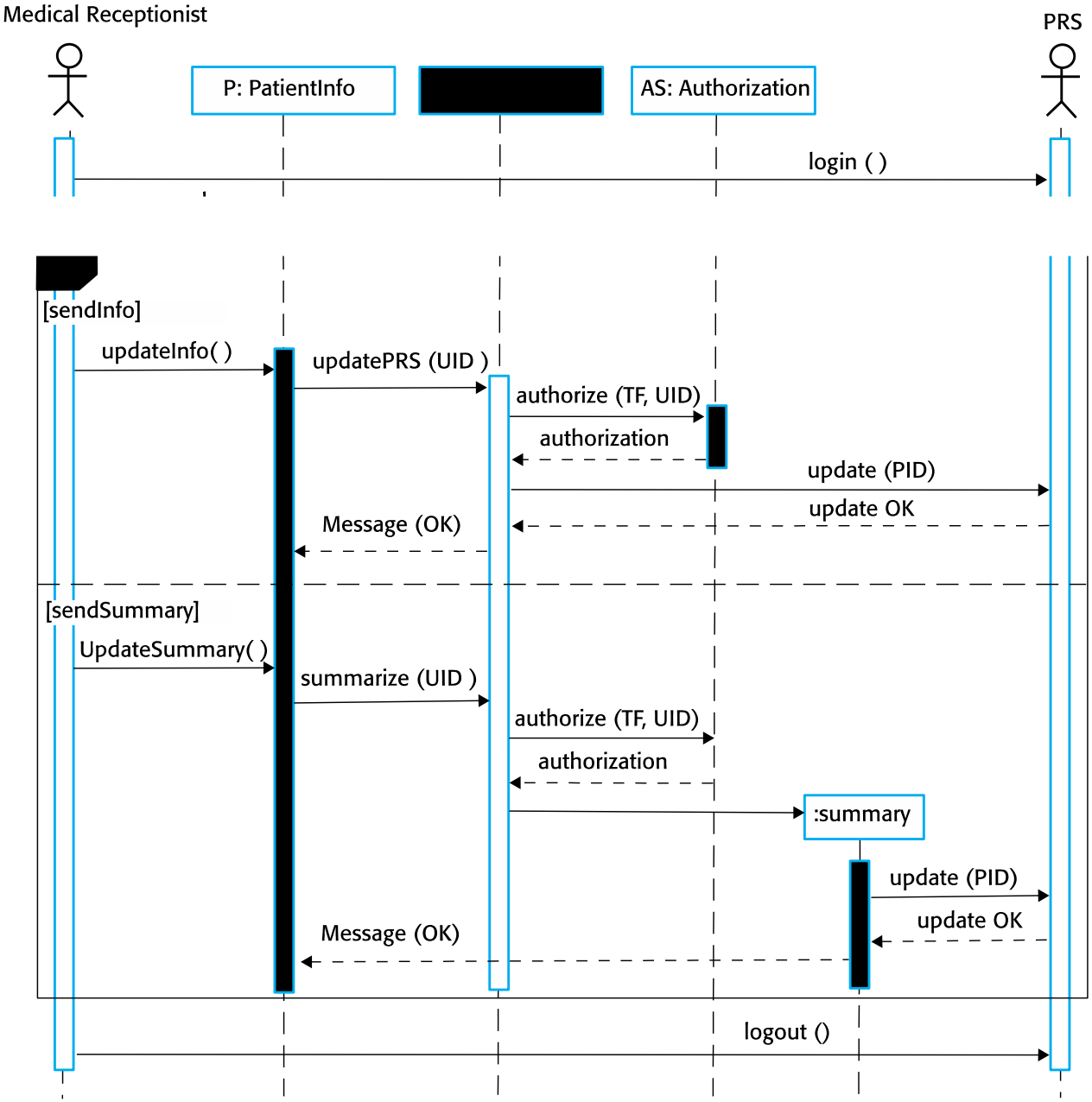


Sequence diagrams

- 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.

Sequence diagram for View patient information





Sequence diagram for Transfer Data

Structural models

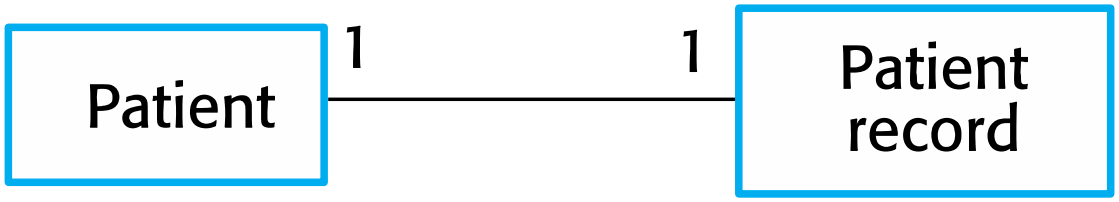
Structural models

- 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.

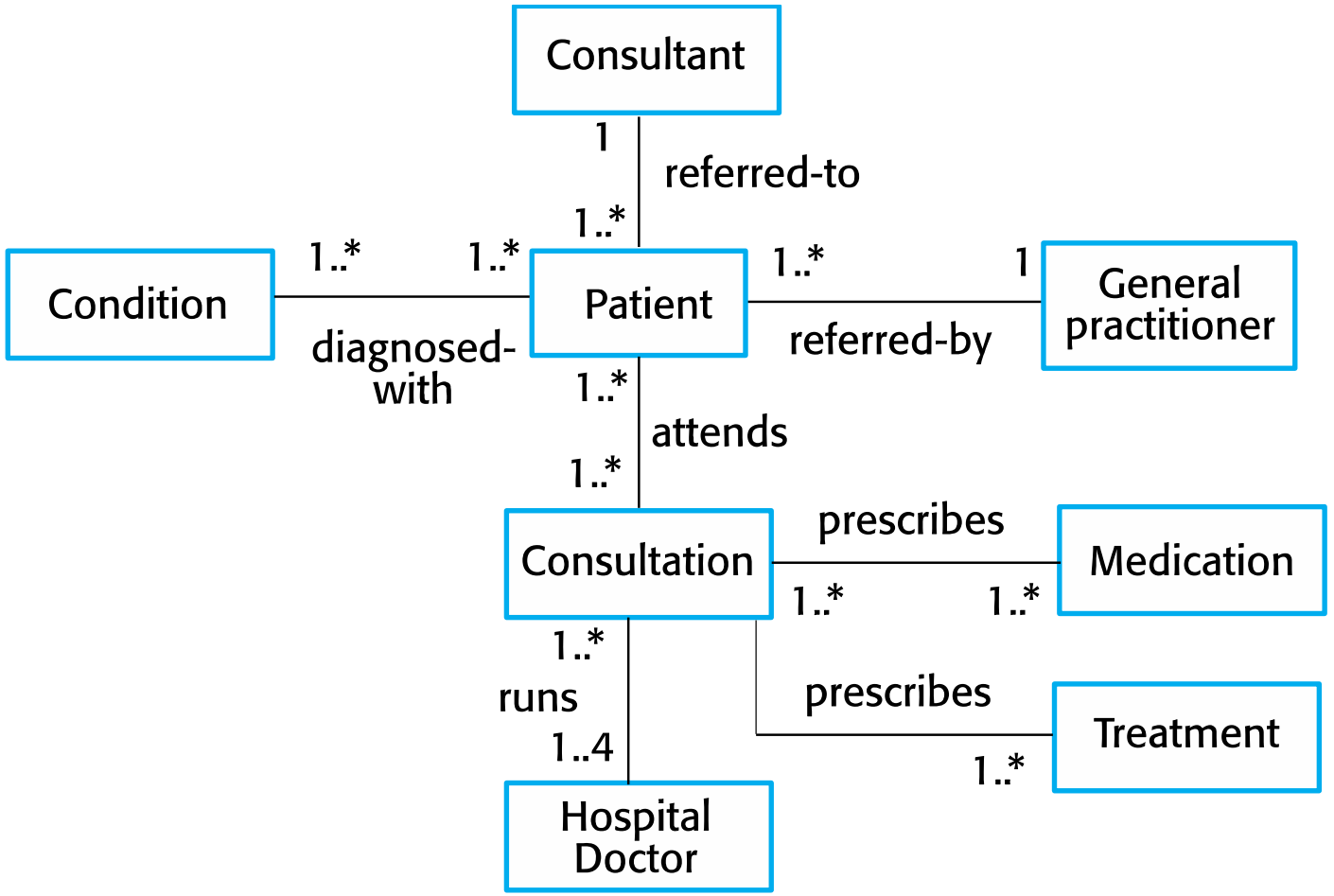
Class diagrams

- 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.

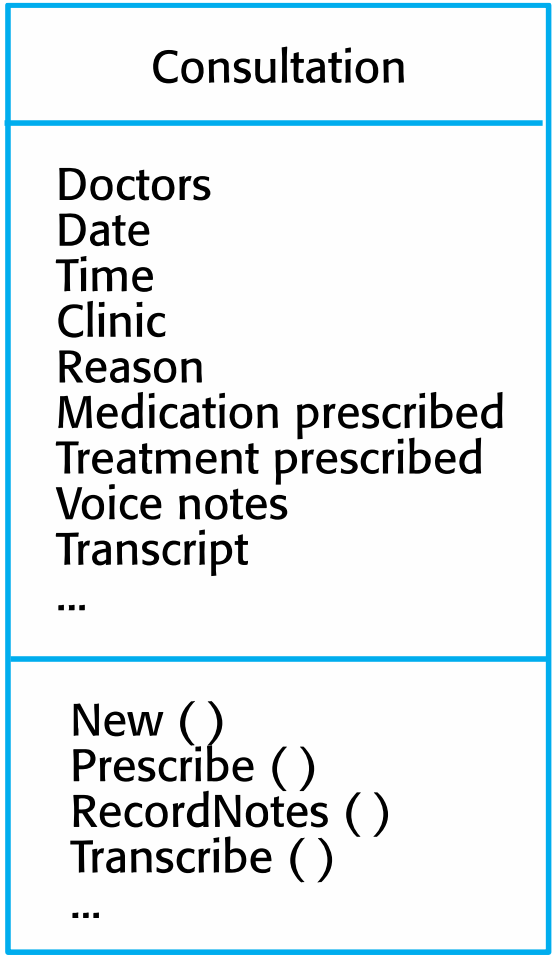
UML classes and association



Classes and associations in the MHC-PMS



The Consultation class



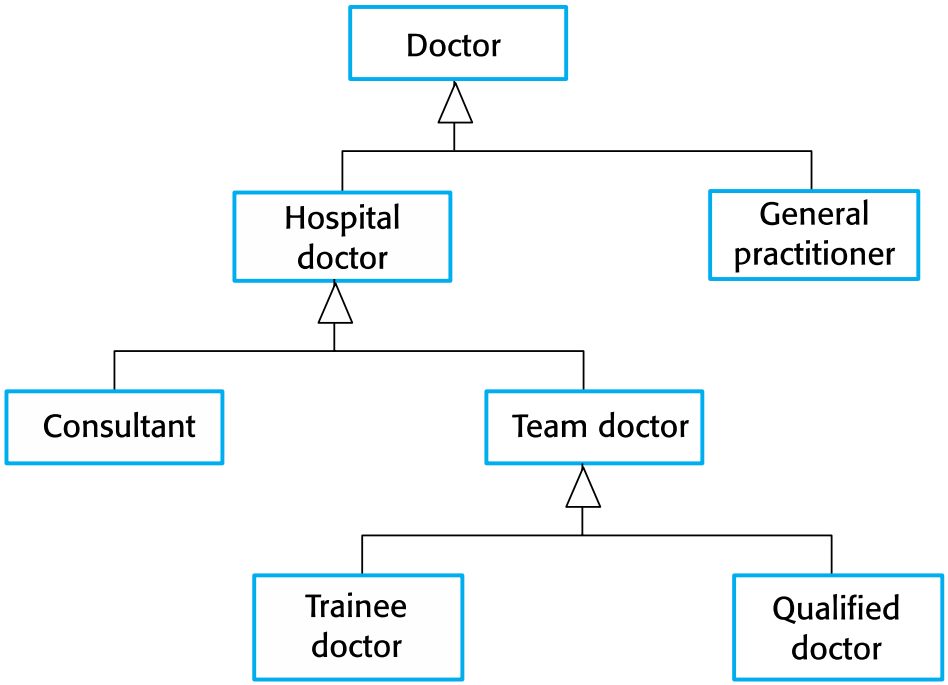
Generalization

- 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.

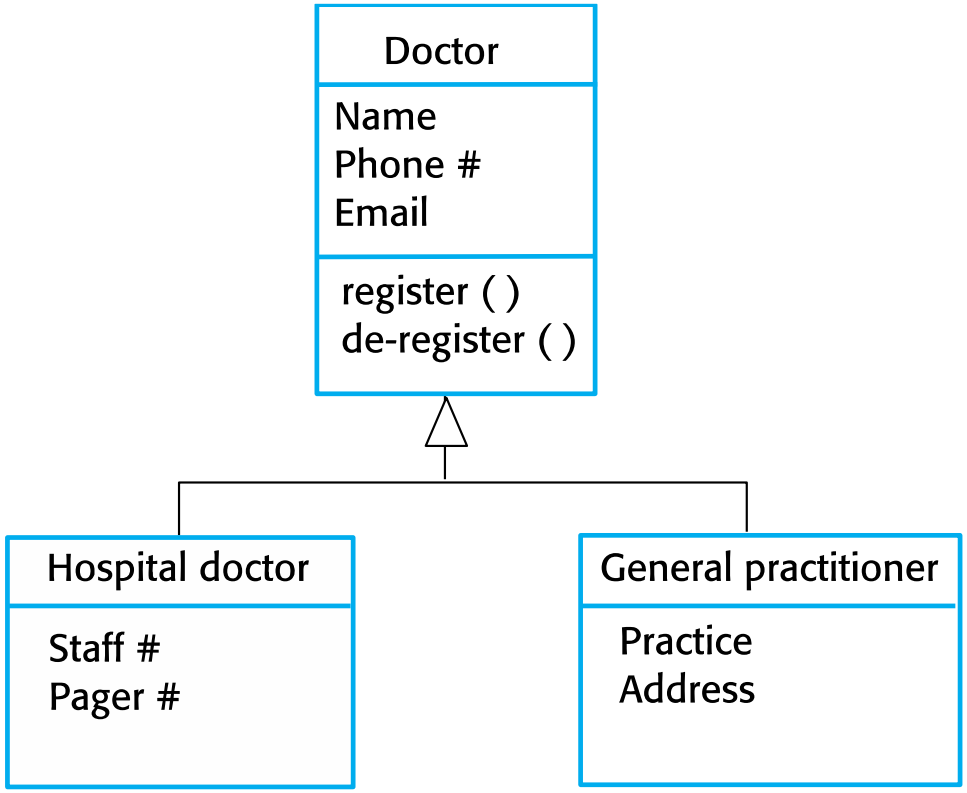
Generalization

- In modeling 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.

A generalization hierarchy



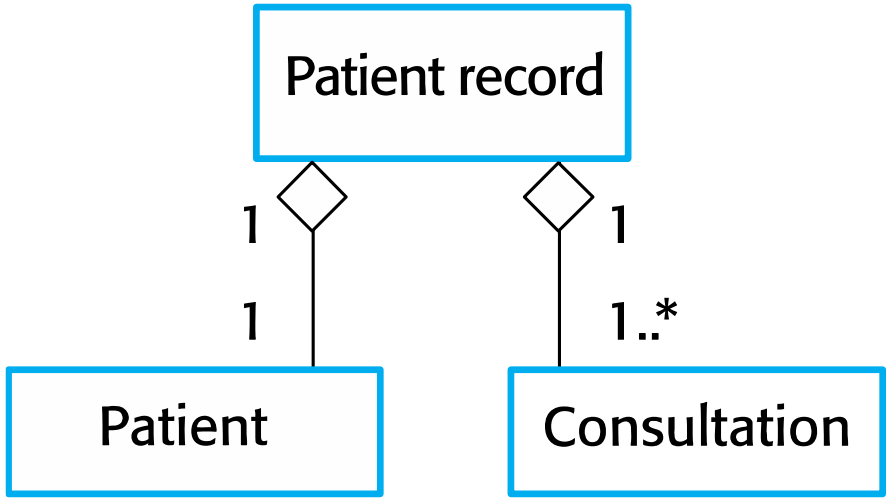
A generalization hierarchy with added detail



Object class aggregation models

- 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.

The aggregation association



Behavioral models

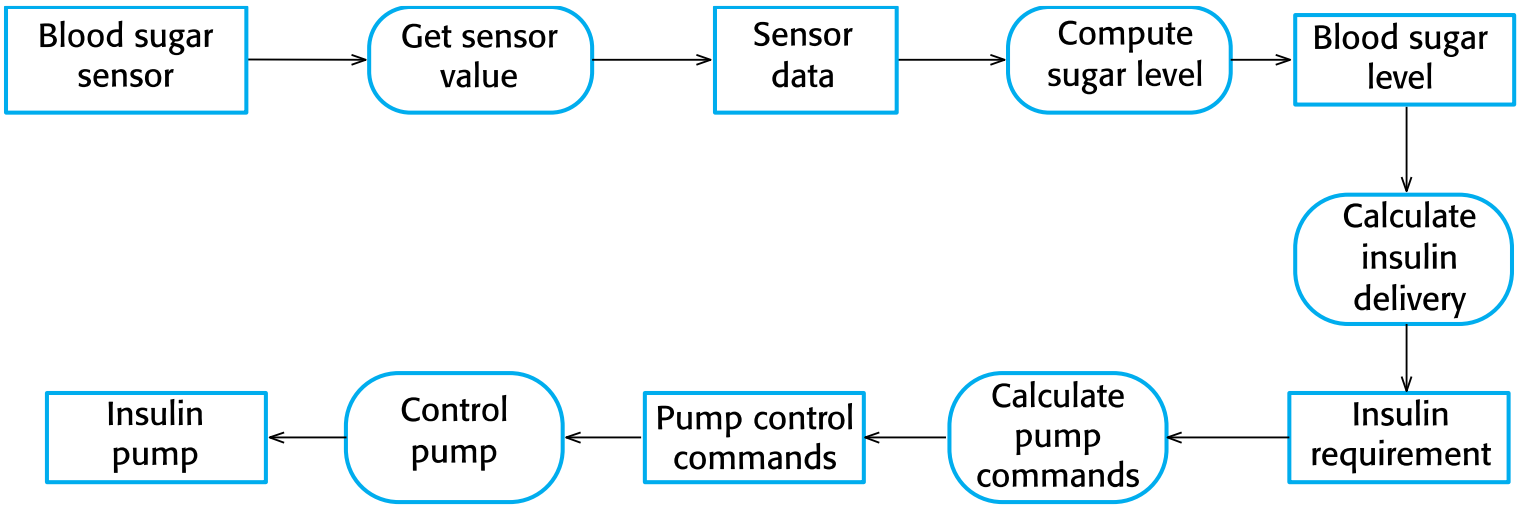
Behavioral models

- 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.

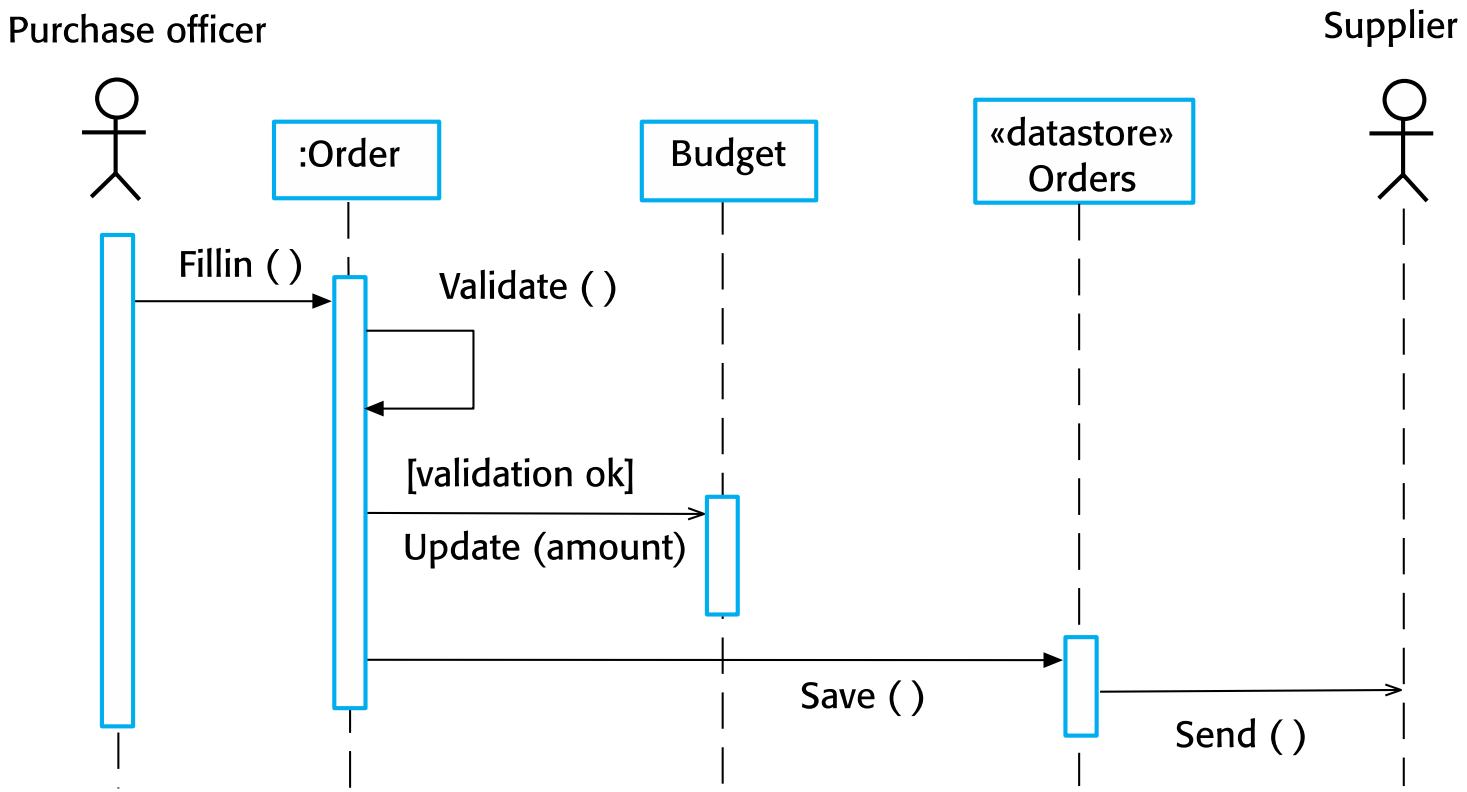
Data-driven modeling

- 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.

An activity model of the insulin pump's operation



Order processing



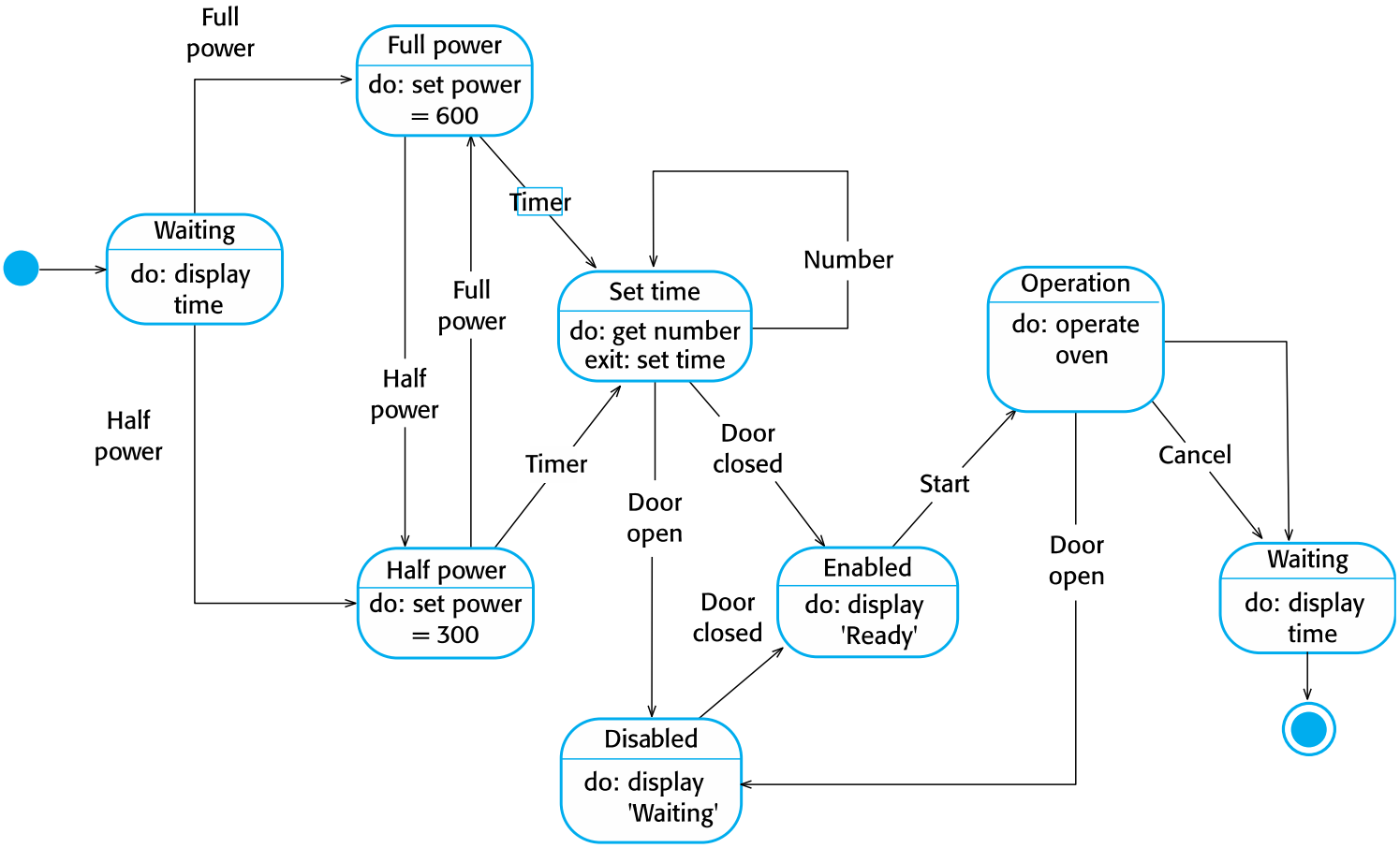
Event-driven modeling

- 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.

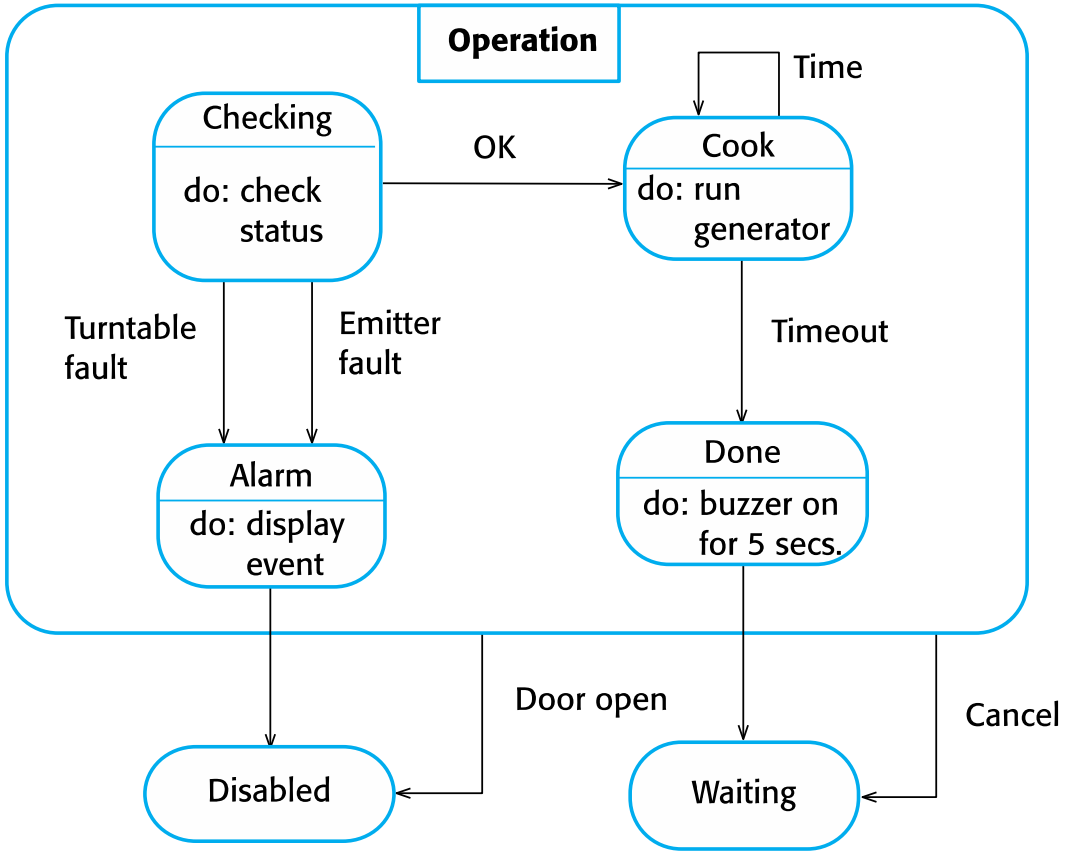
State machine models

- These 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.
- Statecharts are an integral part of the UML and are used to represent state machine models.

State diagram of a microwave oven



Microwave oven operation



States and stimuli for the microwave oven (a)

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.

States and stimuli for the microwave oven (b)

Stimulus	Description
Half power	The user has pressed the half-power button.
Full power	The user has pressed the full-power button.
Timer	The user has pressed one of the timer buttons.
Number	The user has pressed a numeric key.
Door open	The oven door switch is not closed.
Door closed	The oven door switch is closed.
Start	The user has pressed the Start button.
Cancel	The user has pressed the Cancel button.

Model-driven engineering

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.

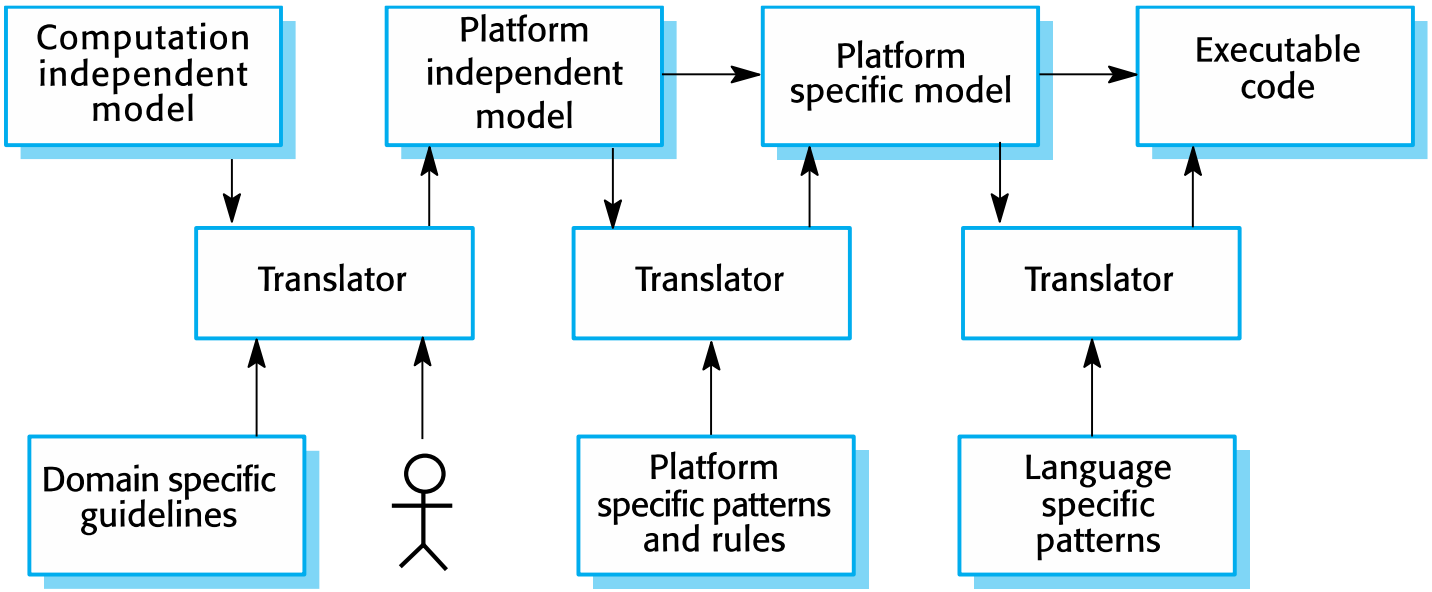
Model driven architecture

- 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.

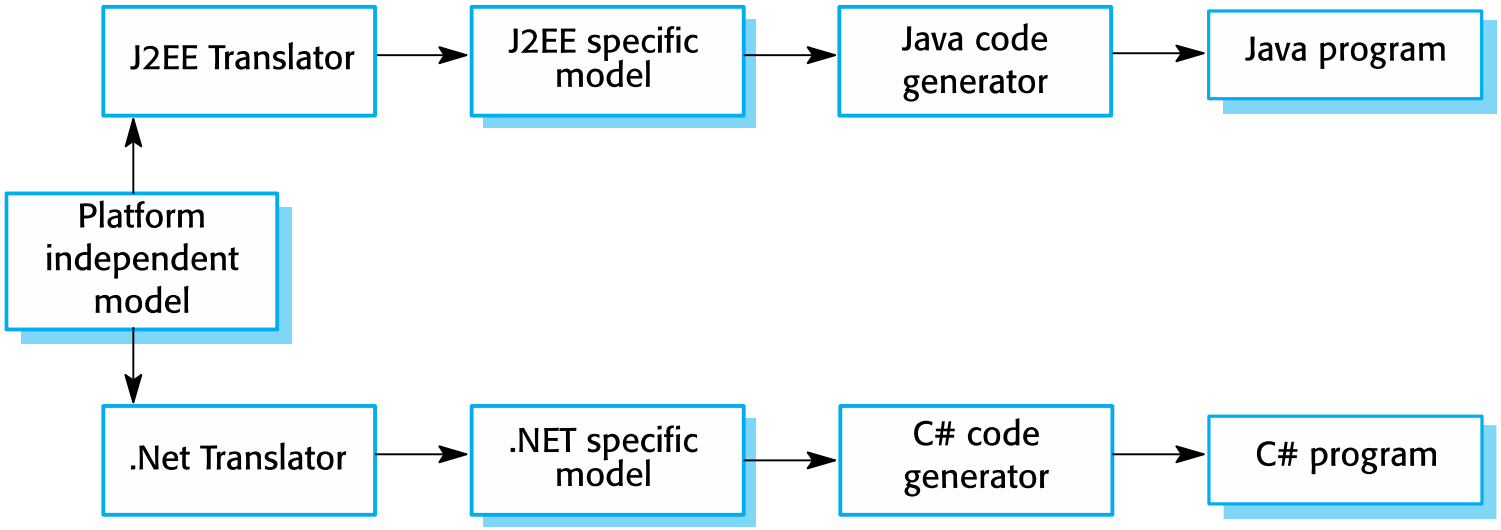
Types of model

- A computation independent model (CIM)
  - These 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.

MDA transformations



Multiple platform-specific models



Agile methods and MDA

- The developers of MDA claim that it is intended to support an iterative approach to development and so can be used within agile methods.
- The notion of extensive up-front modeling contradicts the fundamental ideas in the agile manifesto and I suspect that few agile developers feel comfortable with model-driven engineering.
- If transformations can be completely automated and a complete program generated from a PIM, then, in principle, MDA could be used in an agile development process as no separate coding would be required.

Adoption of MDA

- A range of factors has limited the adoption of MDE/MDA
- Specialized tool support is required to convert models from one level to another
- There is limited tool availability and organizations may require tool adaptation and customisation to their environment
- For the long-lifetime systems developed using MDA, companies are reluctant to develop their own tools or rely on small companies that may go out of business

Adoption of MDA

- Models are a good way of facilitating discussions about a software design. Howeverthe abstractions that are useful for discussions may not be the right abstractions for implementation.
- For most complex systems, implementation is not the major problem – requirements engineering, security and dependability, integration with legacy systems and testing are all more significant.

Adoption of MDA

- The arguments for platform-independence are only valid for large, long-lifetime systems. For software products and information systems, the savings from the use of MDA are likely to be outweighed by the costs of its introduction and tooling.
- The widespread adoption of agile methods over the same period that MDA was evolving has diverted attention away from model-driven approaches.

Key points

- A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system's context, interactions, structure and behavior.
- Context models show how a system that is being modeled is positioned in an environment with other systems and processes.
- Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.
- Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.

Key points

- Behavioral models are used to describe the dynamic behavior of an executing system. This behavior can be modeled from the perspective of the data processed by the system, or by the events that stimulate responses from a system.
- Activity diagrams may be used to model the processing of data, where each activity represents one process step.
- State diagrams are used to model a system's behavior in response to internal or external events.
- Model-driven engineering is an approach to software development in which a system is represented as a set of models that can be automatically transformed to executable code.