ECE 651 – Software Engineering
Week Four: Modeling and Architecture
February 10th, 2016

Ric Telford
Adjunct Associate Professor

Founder, Telford Ventures

Duke | PRATT SCHOOL OF ENGINEERING

02/10/16 – Week 4 Overview

- Recap / Announcements
- Homework Review
- Lecture – System Modeling
- Break
- Android UI (Lee)
- Team Development in Git (Lee)
- Lecture – Architectural Design
- About Week 5

Recap of Week 3

- Agile methods are incremental development methods that focus on rapid software development, frequent releases of the software, reducing process overheads by minimizing documentation and producing high-quality code.
- Agile development practices include
  - User stories for system specification
  - Frequent releases of the software,
  - Continuous software improvement
  - Test-first development
  - Customer participation in the development team.
- Scrum is an agile method that provides a project management framework.
  - It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- Many practical development methods are a mixture of plan-based and agile development.
- Scaling agile methods for large systems is difficult.
  - Large systems need up-front design and some documentation and organizational practice may conflict with the informality of agile approaches.

Announcements

- Upcoming Java Dev lectures
  - Today - UI Development for Android
  - Today - Team-based source control in Git
  - Feb 17th - Java Web UI Development
  - Feb 24th - AWT UI Development
- Mid Term will be March 2nd
  - Covers all material covered to date
  - Focus will be on topics described in “Key Points”, “Recap”, italicized text, bolded text, homework assignment topics.
  - Also one or two high level questions from the Agile Library, a few simple ones on Git and Java
  - Make sure you read the book to reinforce
  - A few questions will be straight from book (maybe not in a lecture), others straight from lectures (maybe not in the book)
  - Multiple Choice, some Fill in the Blank, maybe some “Other”

Homework 3

- Agile Library
- User Stories and Tasks

Chapter 5 – System Modeling

- Introductory Comments
  - Modeling is not a phase of a Software Engineering process, it is a tool to use in the process
    - See http://software-engineering-book.com/web/software-tools for more examples of tools
  - Modeling is used in both the Specification Phase as well as the Development Phase
  - Modeling should only be done as applicable to the project – the more complex the task, the more potential need to model.
  - Detailed modeling is not real popular in the Agile crowd.
  - UML is the most well known documentation of how to model software.
- You should be aware of modeling and UML, but we won't be doing deep or complex modeling in this class (unless you choose to do so for your project).

System modeling

- System modeling is the process (WHAT) 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 (HOW), 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.

System perspectives

- An *external perspective*, where you model the context or environment of the system.
- An *interaction perspective*, where you model the interactions between a system and its environment, or between the components of a system.
- A *structural perspective*, where you model the organization of a system or the structure of the data that is processed by the system.
- A *behavioral perspective*, where you model the dynamic behavior of the system and how it responds to events.

UML diagram types (there are actually 13)

- *Activity diagrams,* which show the activities involved in a process or in data processing .
- *Use case diagrams*, which show the interactions between a system and its environment.
- *Sequence diagrams,* which show interactions between actors and the system and between system components.
- *Class diagrams*, which show the object classes in the system and the associations between these classes.
- *State diagrams*, which show how the system reacts to internal and external events.
- See http://www.tutorialspoint.com/uml/uml_basic_notations.htm for more detailed descriptions

Use of graphical models

- As a means of **facilitating discussion** about an existing or proposed system
  - Incomplete and incorrect models are OK as their role is to support discussion.
- As a way of documenting an **existing** system
  - Models should be an accurate representation of the system but need not be complete.
- As a detailed system description that can be used to **generate** a system implementation
  - Models have to be both correct and complete.

Topics covered

- Context models
- Interaction models
- Structural models
- Behavioral models
- Model-driven engineering

Context models

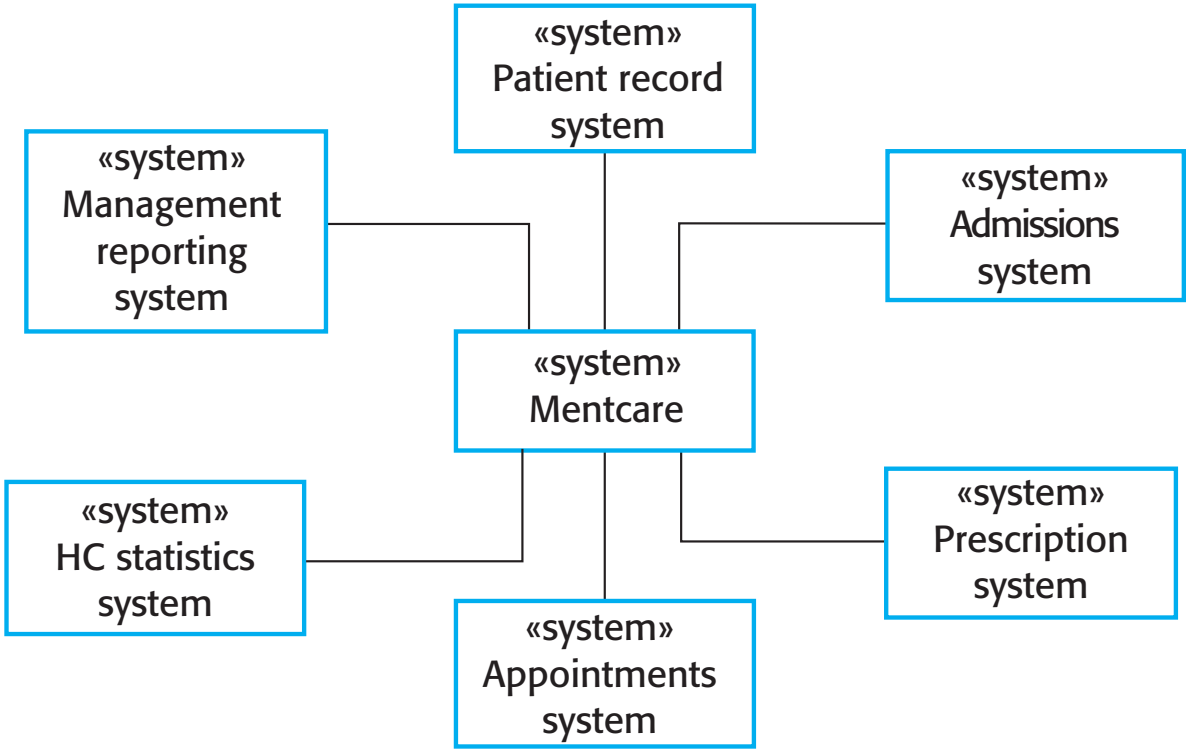
Context models

- Context models are used to illustrate the **operational context of a system** - they show what lies outside the system boundaries.
- Social and organizational 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.

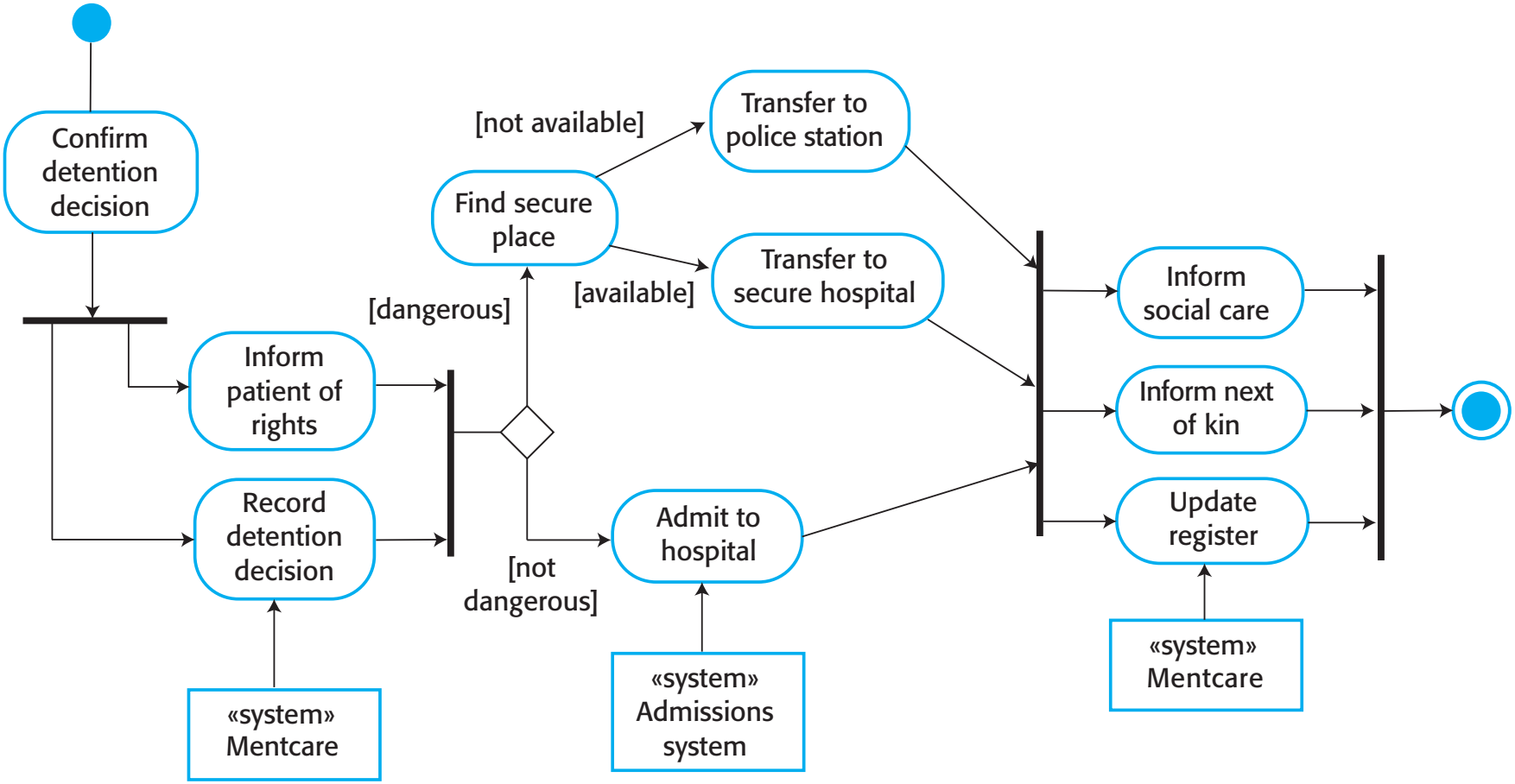
The context of the Mentcare system



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

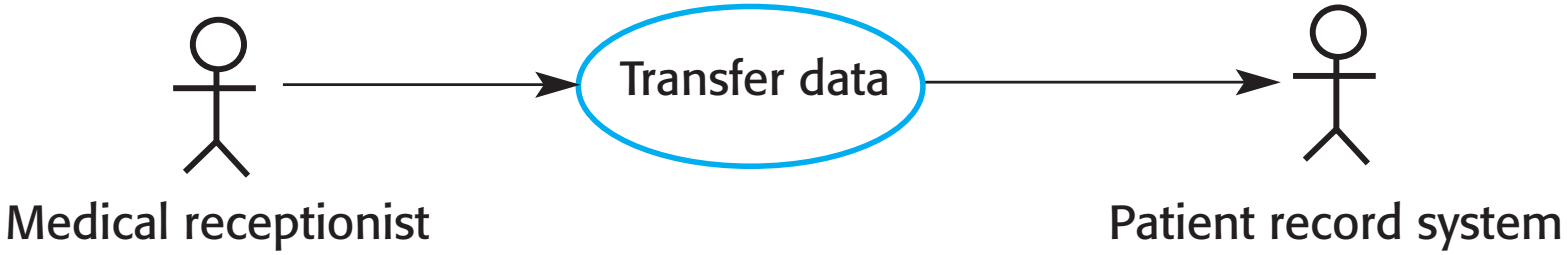
- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- UML **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 diagrammatically to provide an overview of the use case and in a more detailed textual form.

Transfer-data use case

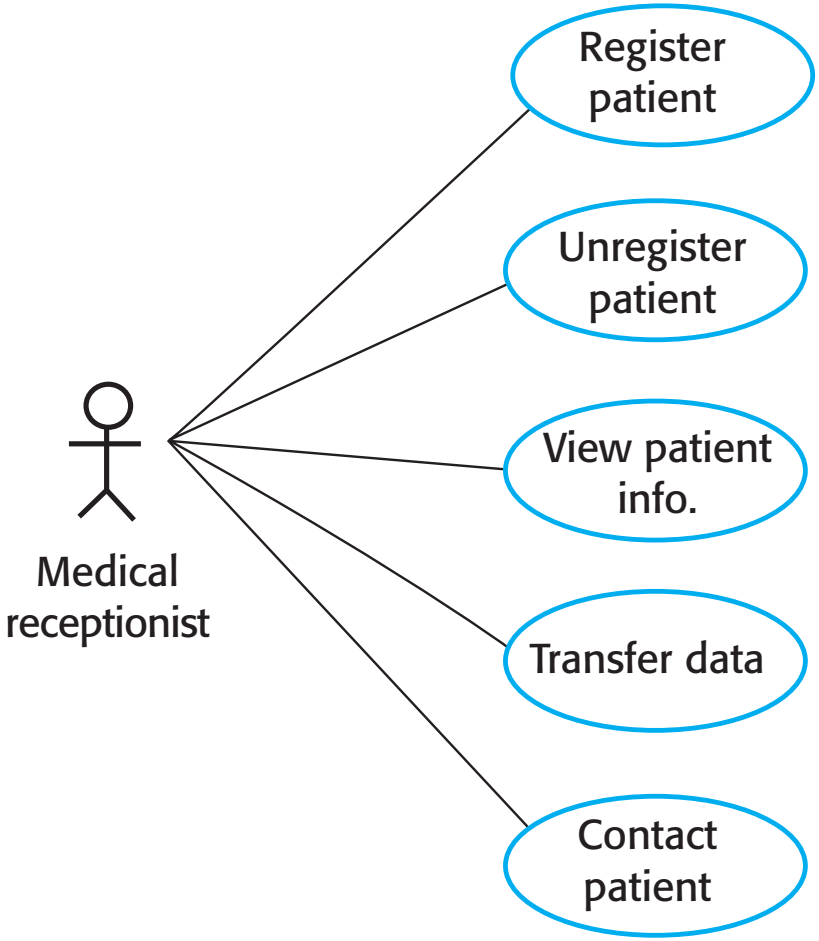
- A use case in the Mentcare system



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.

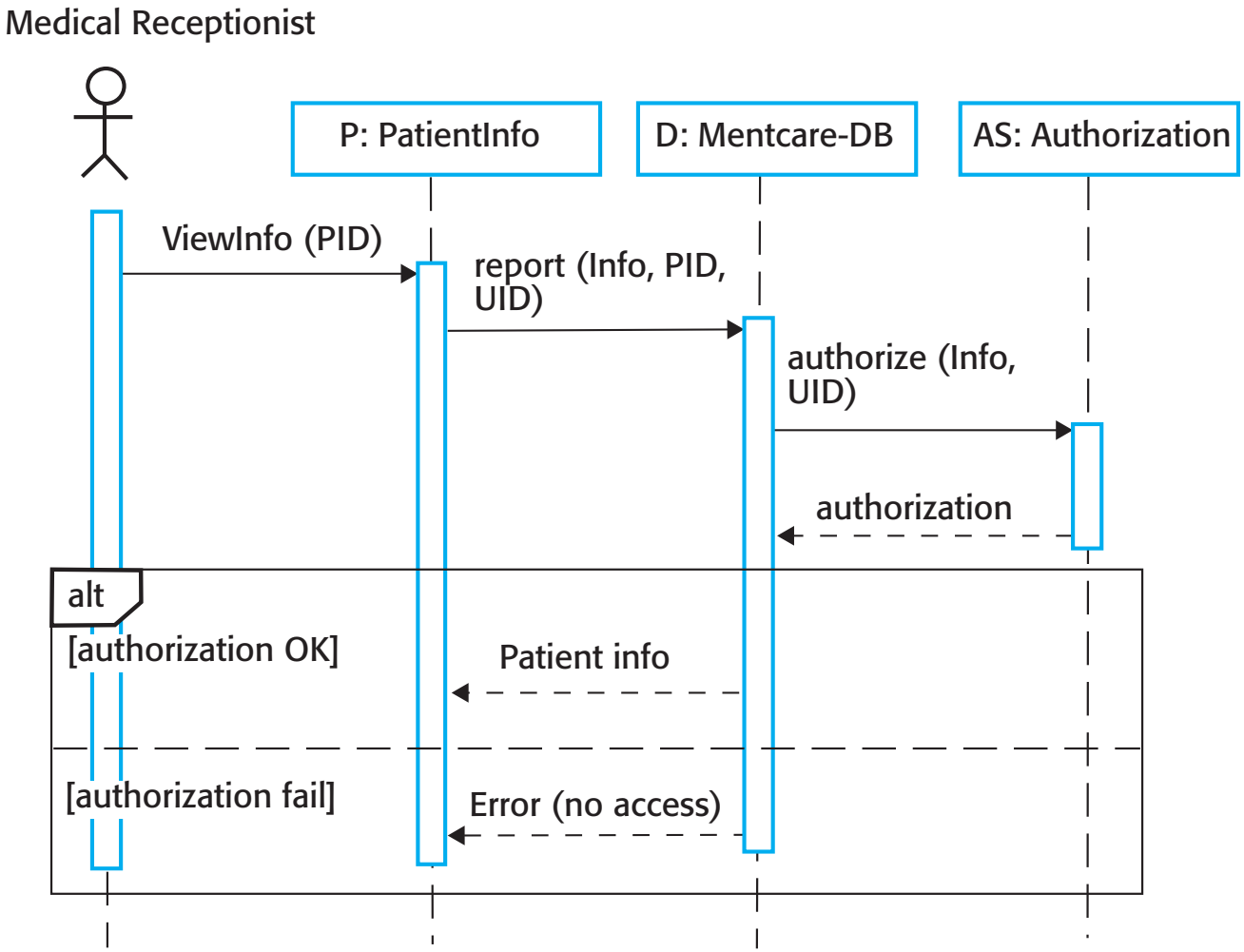
Use cases in the Mentcare system involving the role 'Medical Receptionist'



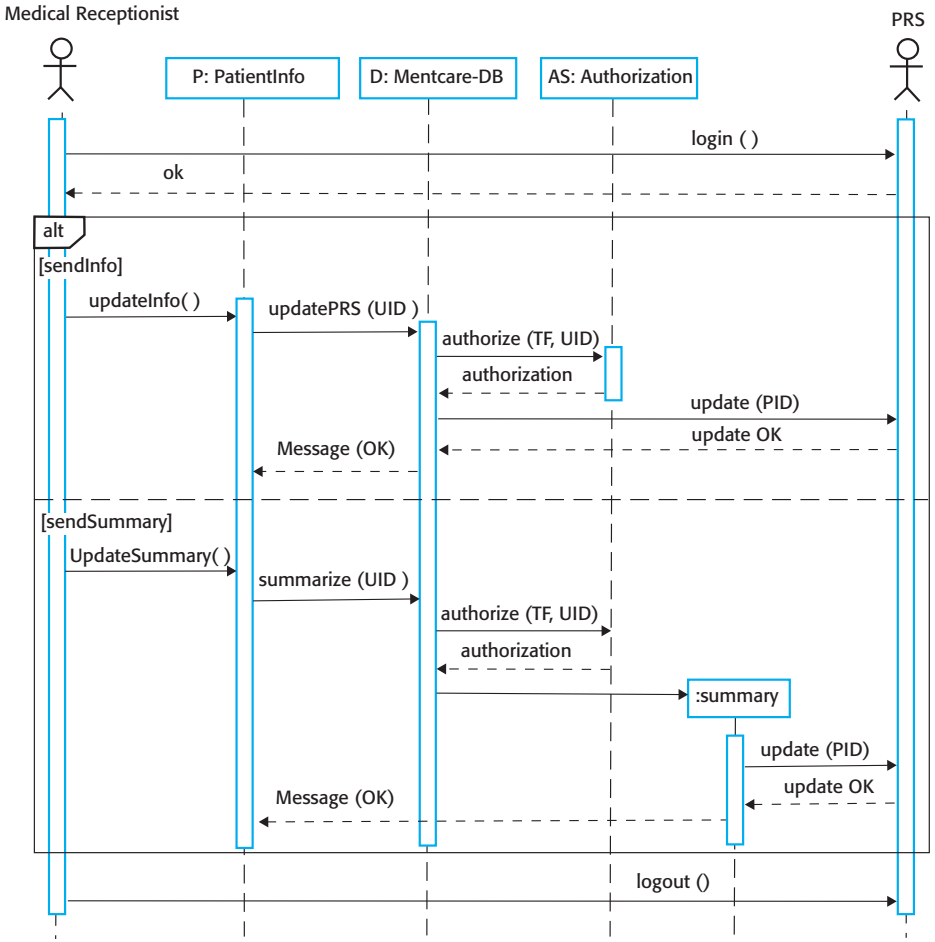
Sequence diagrams

- 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

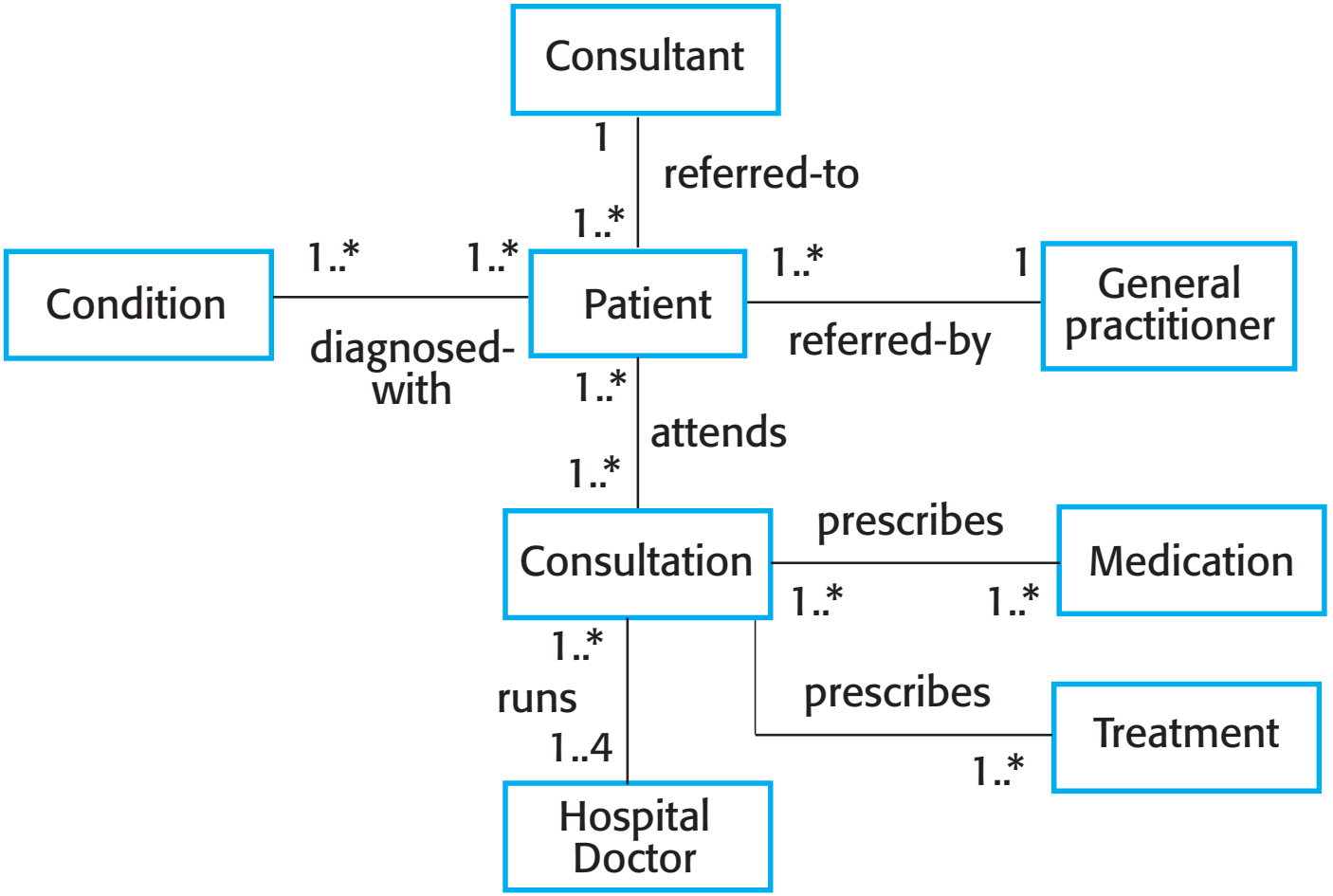
- Structural models of software display the organization of a system in terms of the **components** that make up that system and their relationships.
- Structural models may be **static models**, which show the structure of the system design, or **dynamic models**, which show the organization of the system when it is executing.
- You create structural models of a system when you are discussing and designing the **system architecture**.

Class diagrams

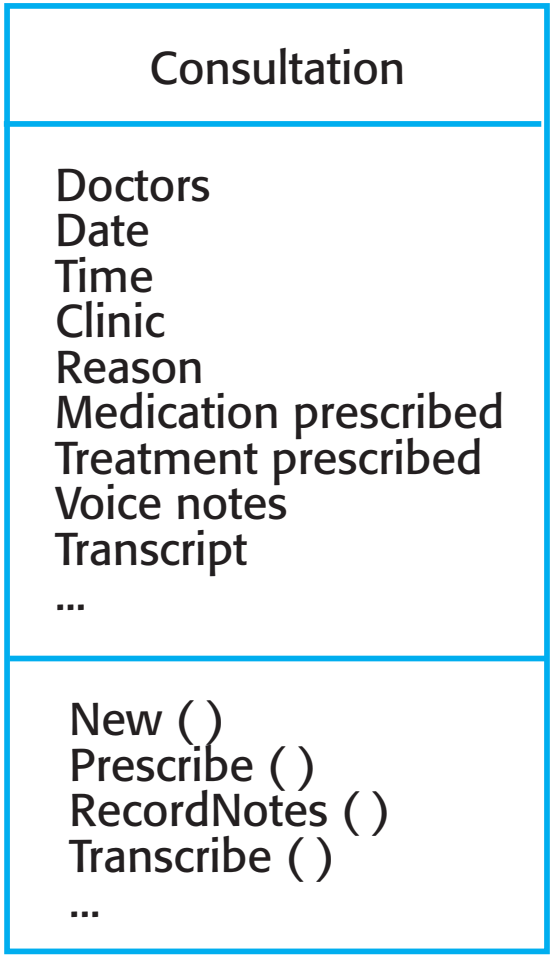
- Class diagrams are used when **developing an object-oriented system model** to show the classes in a system and the associations between these classes.
- An object class can be thought of as a general definition of one kind of system object.
- An association is a link between classes that indicates 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



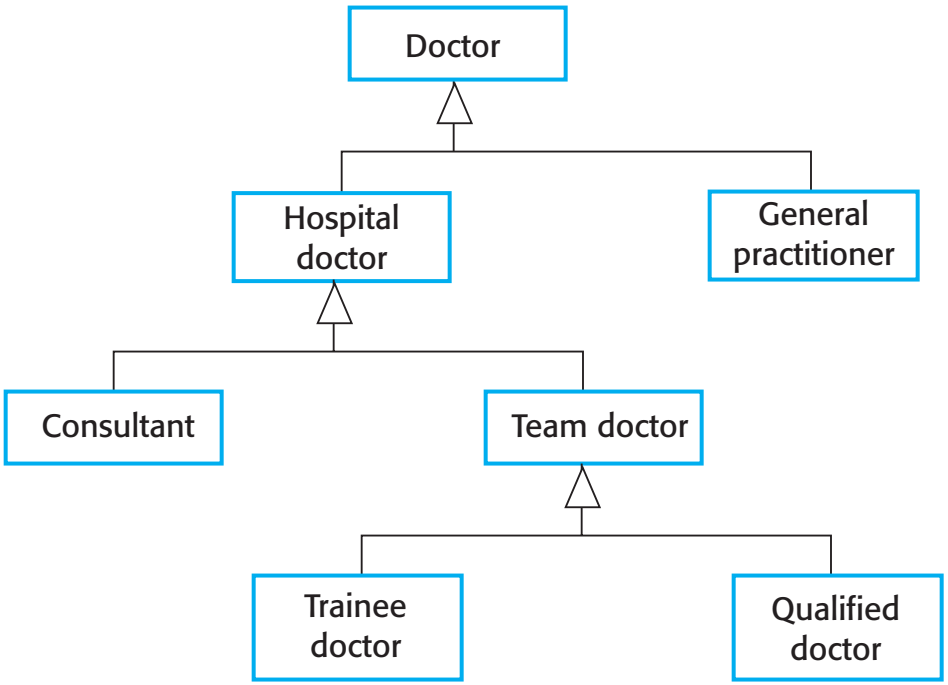
Generalization (aka "building a class hierarchy")

- 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.

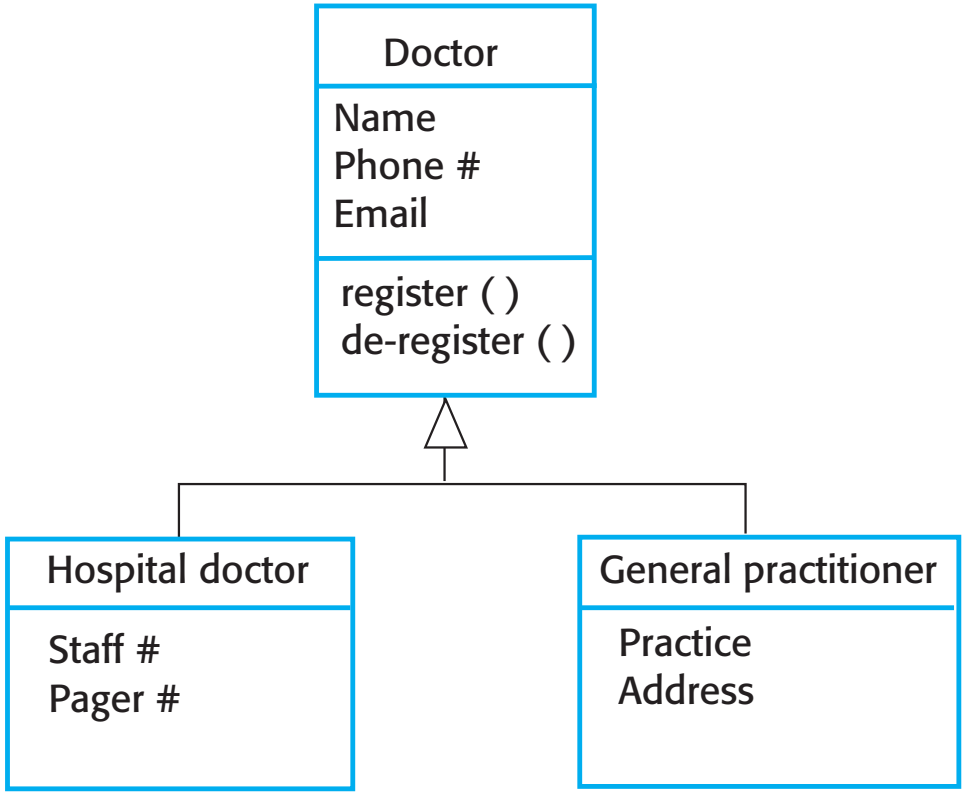
Generalization

- 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



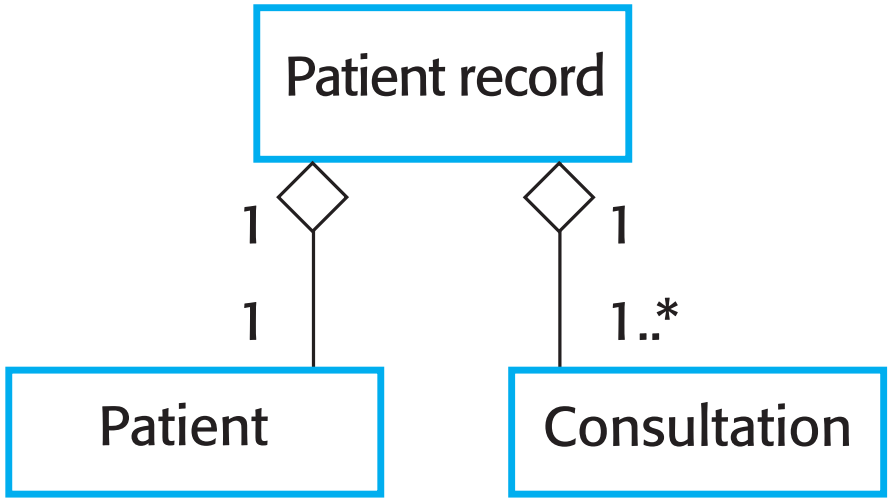
A generalization hierarchy with added detail



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

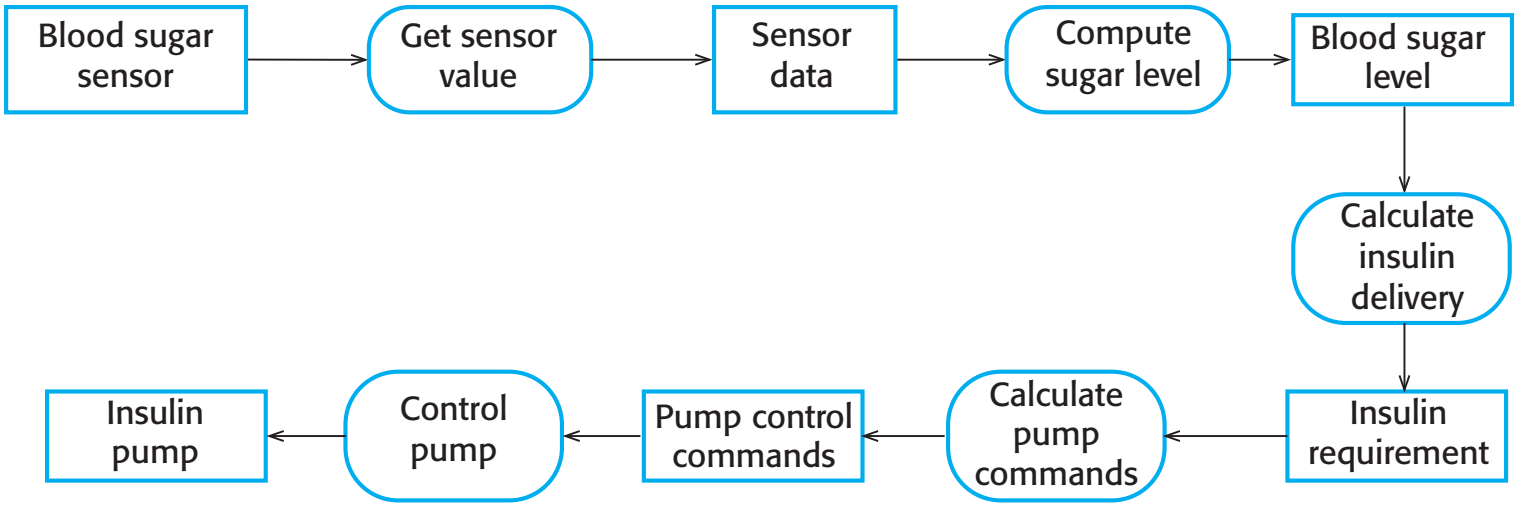
Behavioral models

- 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.

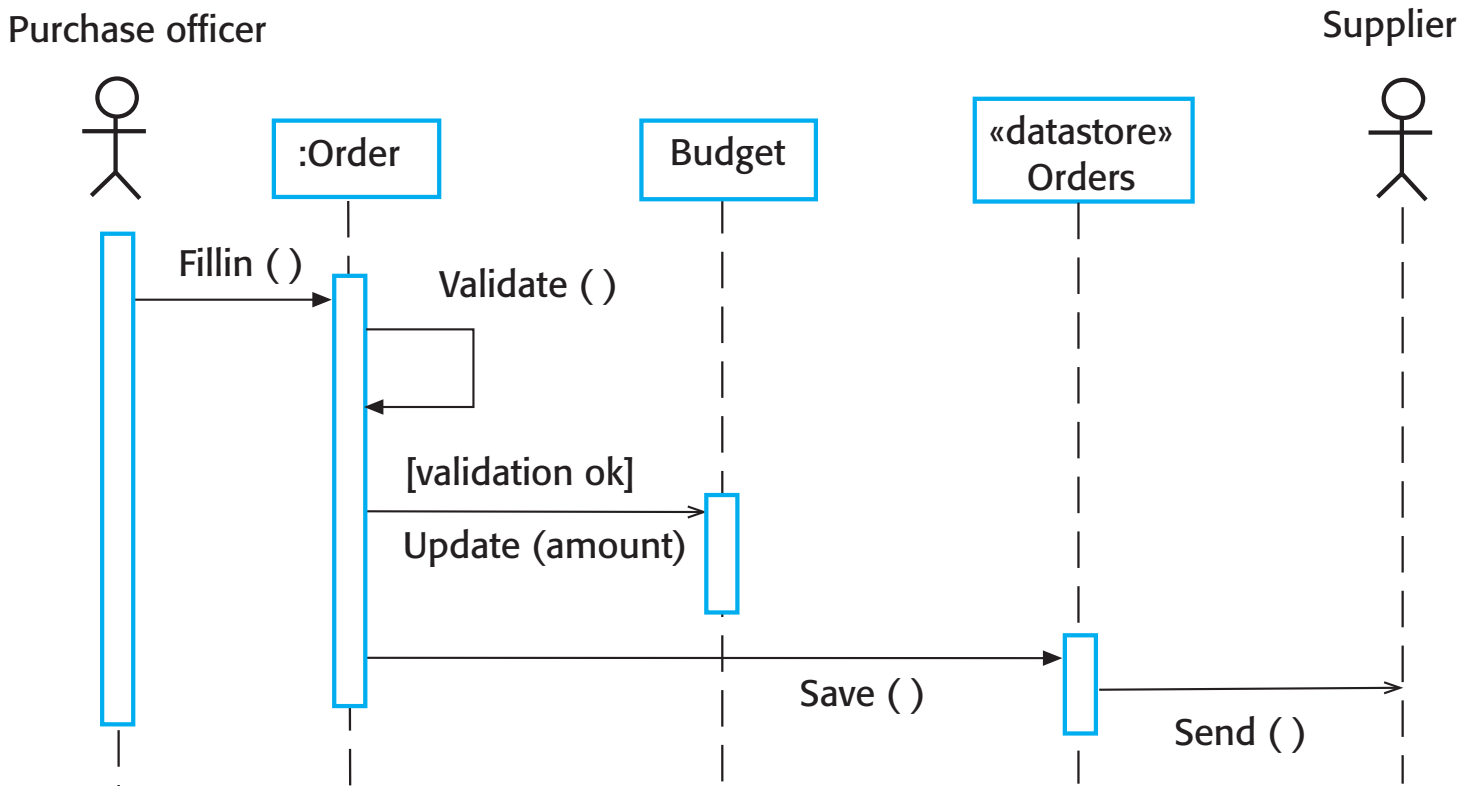
Data-driven modeling

- 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



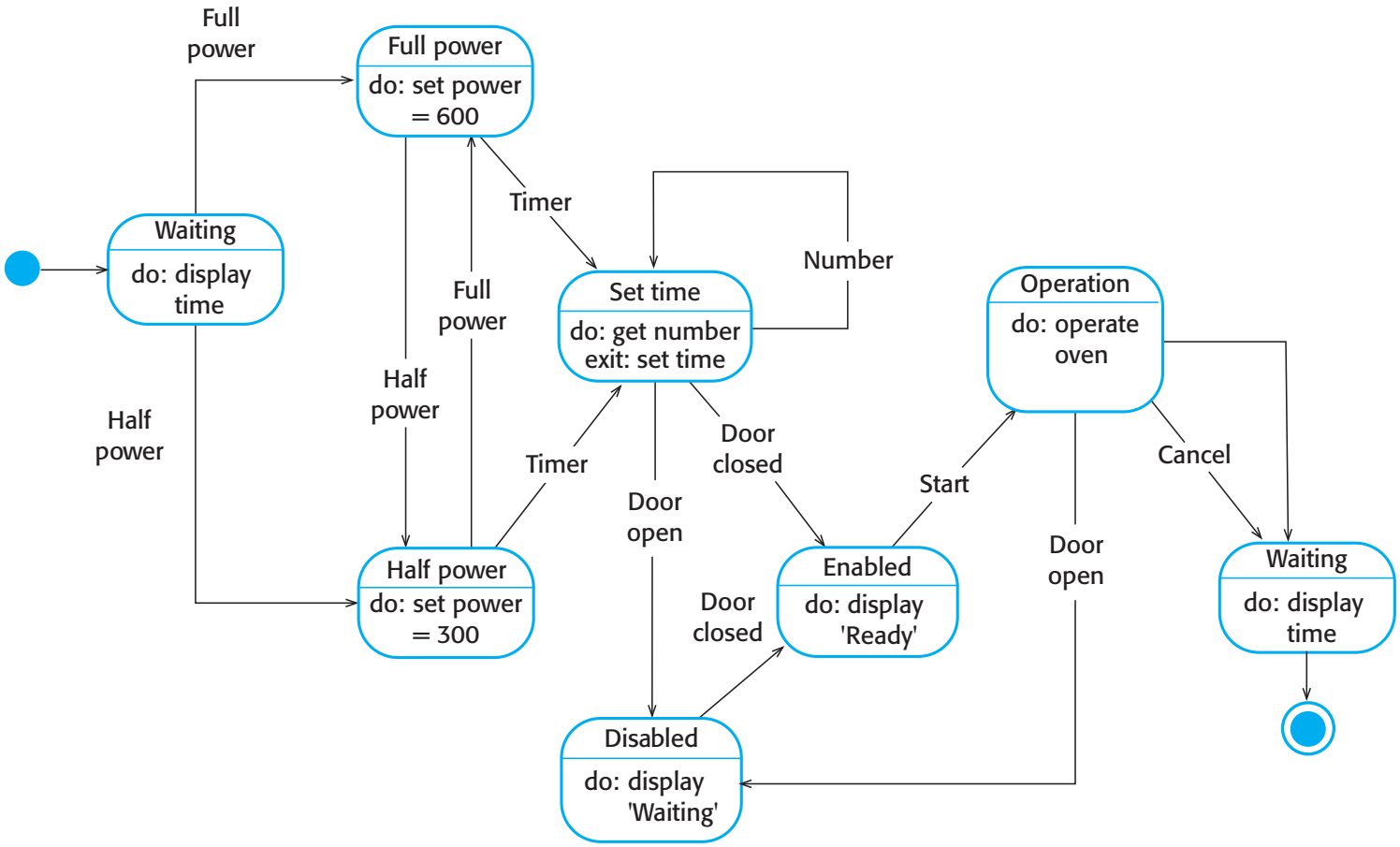
Event-driven modeling

- 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.

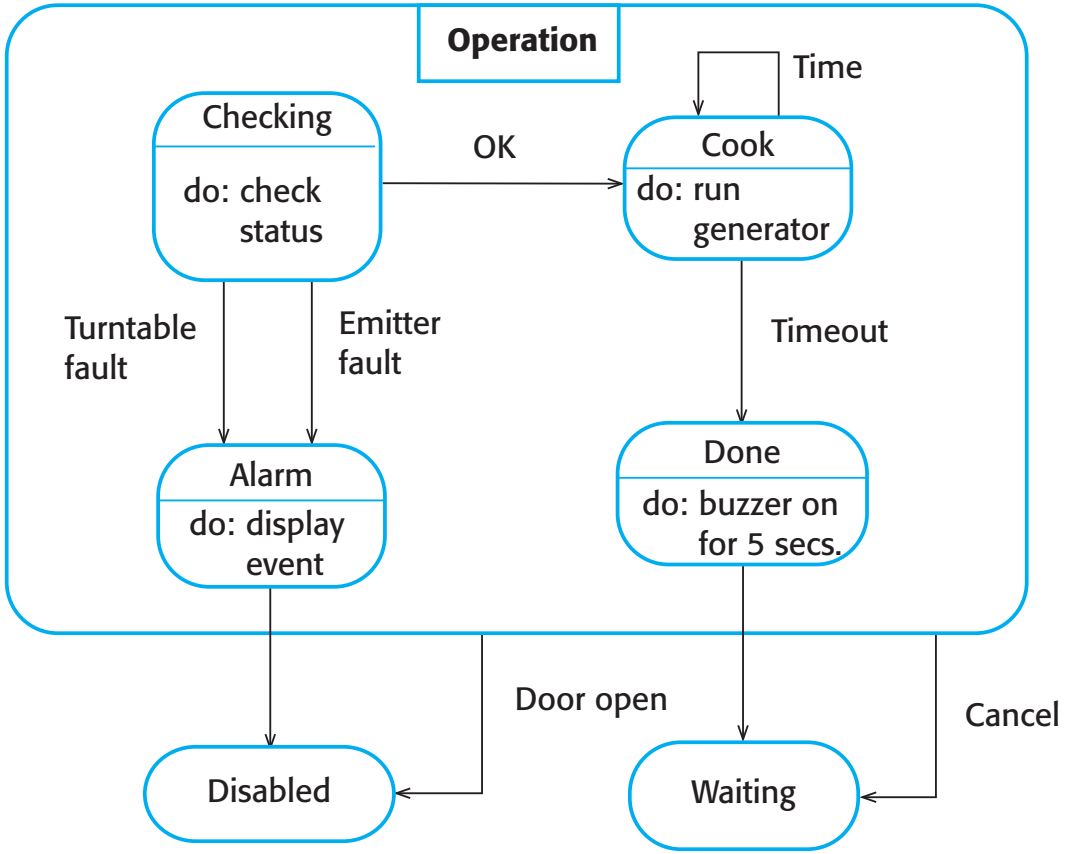
State machine models

- 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.

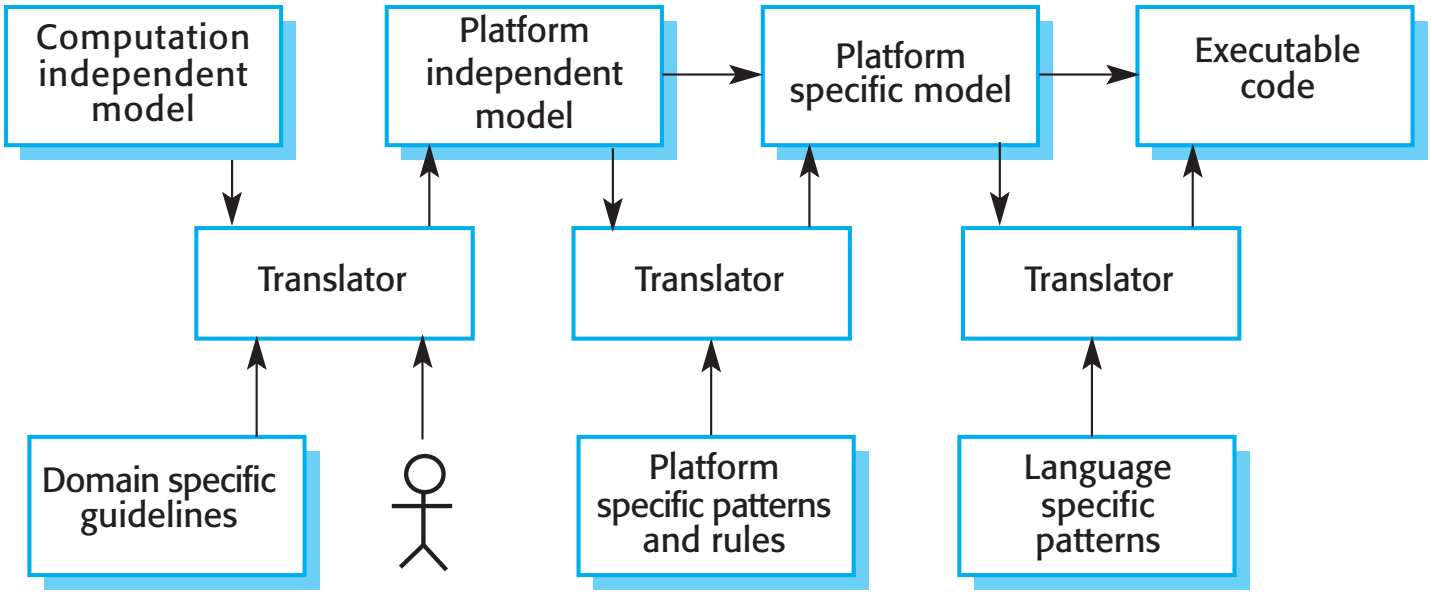
Model driven architecture

- 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.

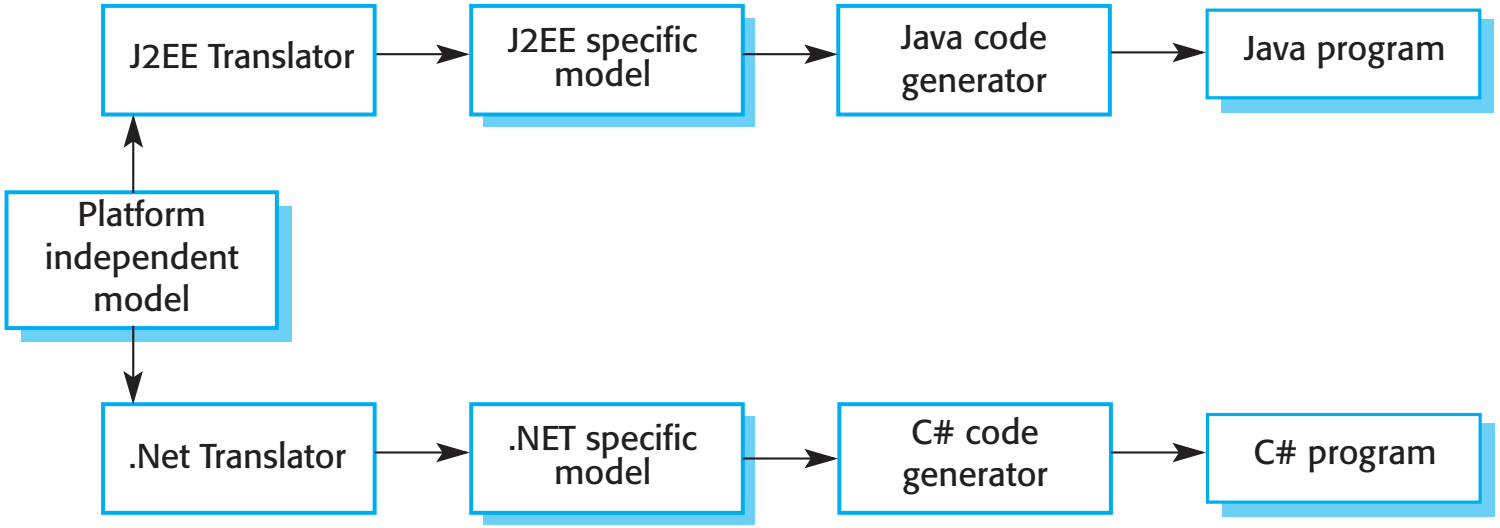
Types of model

- 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 customization 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. **However the 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.

Chapter 6 – Architectural Design

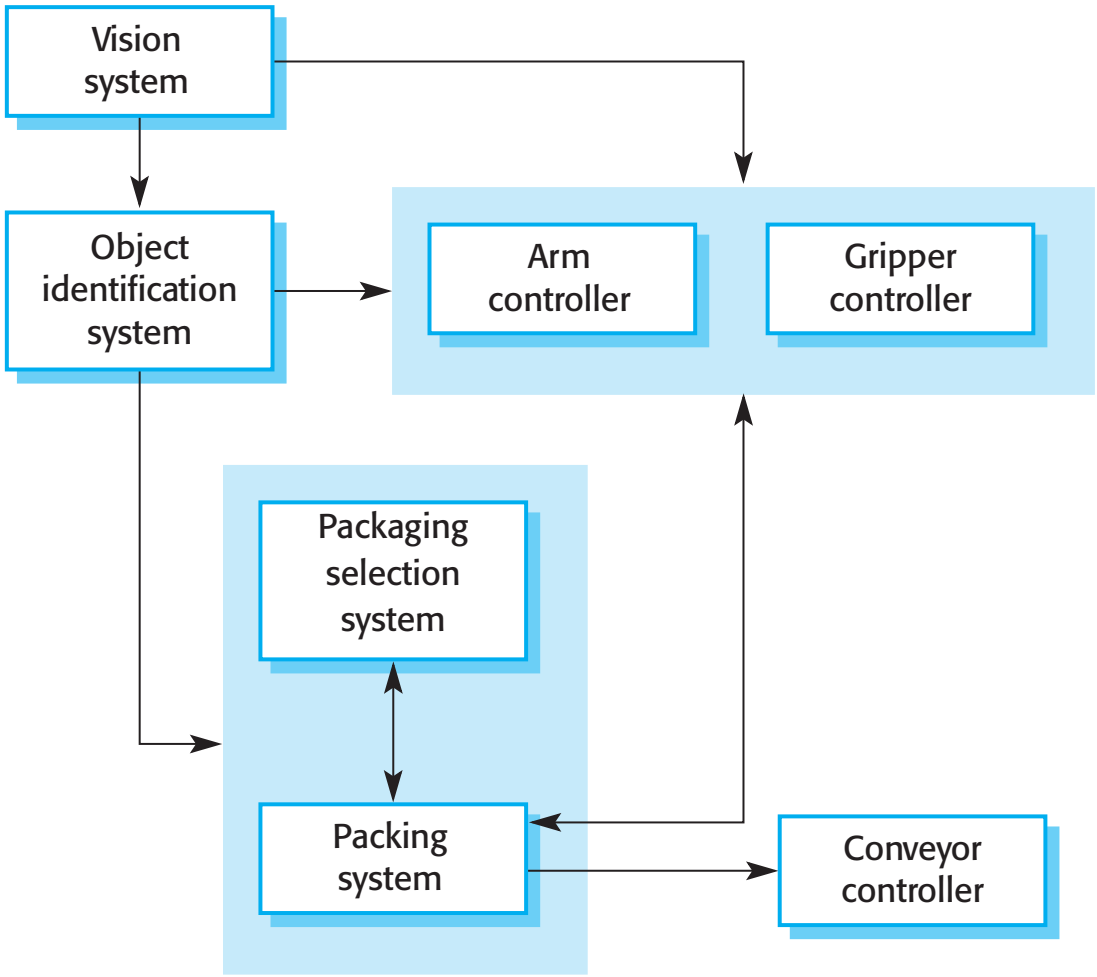
Architectural design

- *Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system.*
- Architectural design is the **critical link between design and requirements engineering**, as it identifies the main structural components in a system and the relationships between them.
- The output of the architectural design process is an **architectural model** that describes how the system is organized as a set of communicating components.

Agility and architecture

- It is generally accepted that an **early stage of agile** processes is to design an overall systems architecture.
- **Refactoring the system architecture is usually expensive** because it affects so many components in the system

The architecture of a packing robot control system



Architectural abstraction

- Architecture in the small is concerned with the **architecture of individual programs.** At this level, we are concerned with the way that an individual program is decomposed into components.
- Architecture in the large is concerned with the **architecture of complex enterprise systems** that include other systems, programs, and program components. These enterprise systems are distributed over different computers, which may be owned and managed by different companies.

Advantages of explicit architecture

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems
  - Product-line architectures may be developed.

Architectural representations

- **Simple, informal block diagrams** showing *entities* and *relationships* are the most frequently used method for documenting software architectures.
- But these have been criticized because they lack semantics, do not show the types of relationships between entities nor the visible properties of entities in the architecture.
- Depends on the use of architectural models. The requirements for model semantics depends on how the models are used.

Block diagrams

- Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
- However, **useful for communication with stakeholders and for project planning.**

Use of architectural models

- As a way of facilitating discussion about the system design
  - A high-level architectural view of a system is useful for communication with system stakeholders and project planning because it is not cluttered with detail. Stakeholders can relate to it and understand an abstract view of the system. They can then discuss the system as a whole without being confused by detail.
- As a way of documenting an architecture that has been designed
  - The aim here is to **produce a complete system model that shows the different components in a system, their interfaces and their connections.**

Topics covered

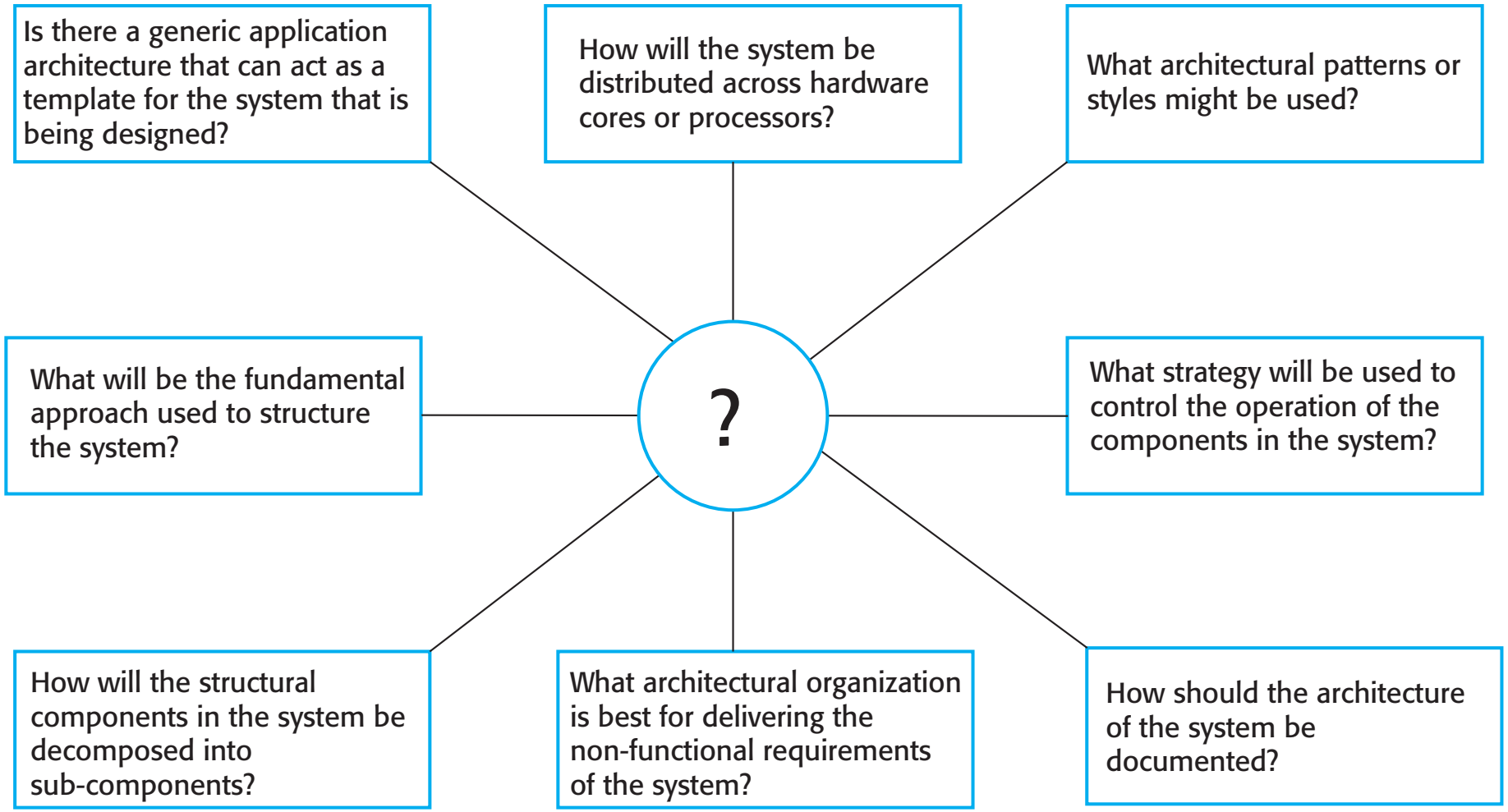
- Architectural design decisions
- Architectural views
- **Architectural patterns**
- Application architectures

Architectural design decisions

Architectural design decisions

- Architectural design is a creative process so the process differs depending on the type of system being developed.
- However, a number of common decisions span all design processes and these decisions affect the non-functional characteristics of the system.

Architectural design decisions



Architecture reuse

- Systems in the same domain often have similar architectures that reflect domain concepts.
- Application product lines are built around a core architecture with variants that satisfy particular customer requirements.
- The architecture of a system may be designed around **one or more architectural patterns or 'styles'**.
  - These capture the essence of an architecture and can be instantiated in different ways.

Architecture and system characteristics

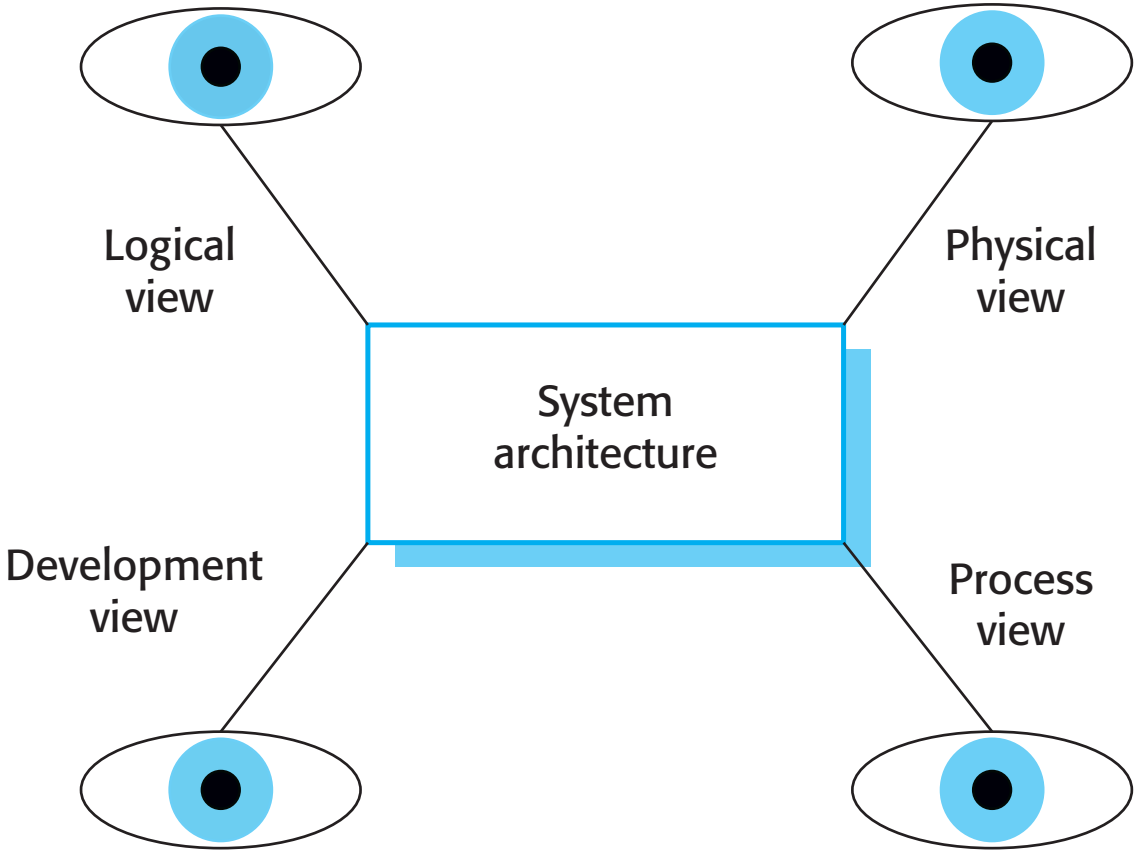
- Performance
  - Localize critical operations and minimize communications. Use large rather than fine-grain components.
- Security
  - Use a layered architecture with critical assets in the inner layers.
- Safety
  - Localize safety-critical features in a small number of sub-systems.
- Availability
  - Include redundant components and mechanisms for fault tolerance.
- Maintainability
  - Use fine-grain, replaceable components.

Architectural views

Architectural views

- What views or perspectives are useful when designing and documenting a system's architecture?
- What notations should be used for describing architectural models?
- **Each architectural model only shows one view or perspective of the system.**
  - It might show how a system is decomposed into modules, how the run-time processes interact or the different ways in which system components are distributed across a network. For both design and documentation, you usually need to present multiple views of the software architecture.

Architectural views



4 + 1 view model of software architecture

- A logical view, which shows the key abstractions in the system as objects or object classes.
- A process view, which shows how, at run-time, the system is composed of interacting processes.
- A development view, which shows how the software is decomposed for development.
- A physical view, which shows the system hardware and how software components are distributed across the processors in the system.
- Related using use cases or scenarios (+1)

Representing architectural views

- Some people argue that the Unified Modeling Language (UML) is an appropriate notation for describing and documenting system architectures
- I disagree with this as I do not think that the UML includes abstractions appropriate for high-level system description.
- Architectural description languages (ADLs) have been developed but are not widely used

Architectural patterns

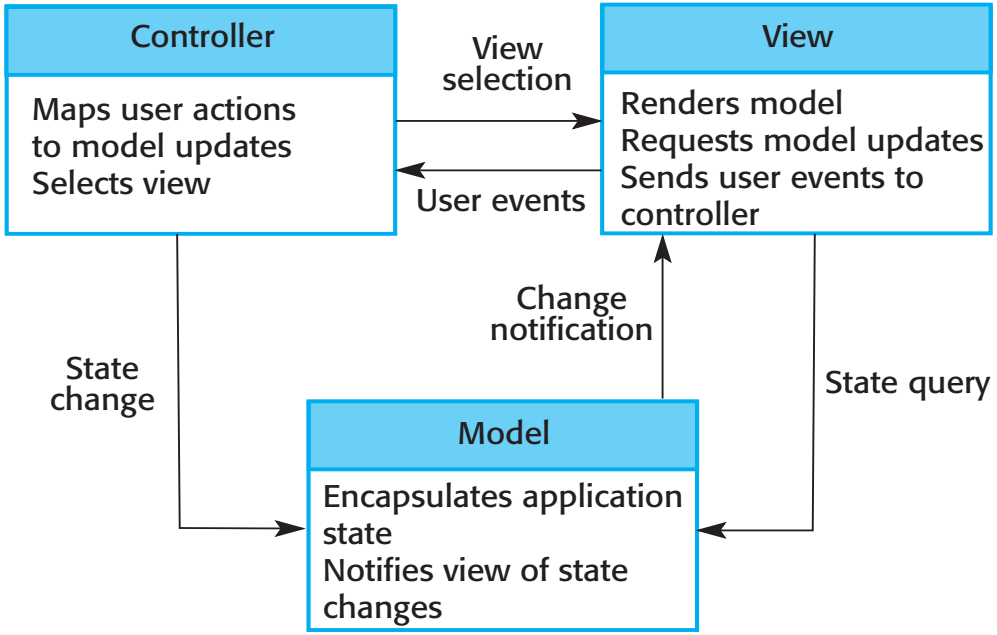
Architectural patterns

- **Patterns are a means of representing, sharing and reusing knowledge.**
- *An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.*
- Patterns should include information about when they are and when the are not useful.
- Patterns may be represented using tabular and graphical descriptions.

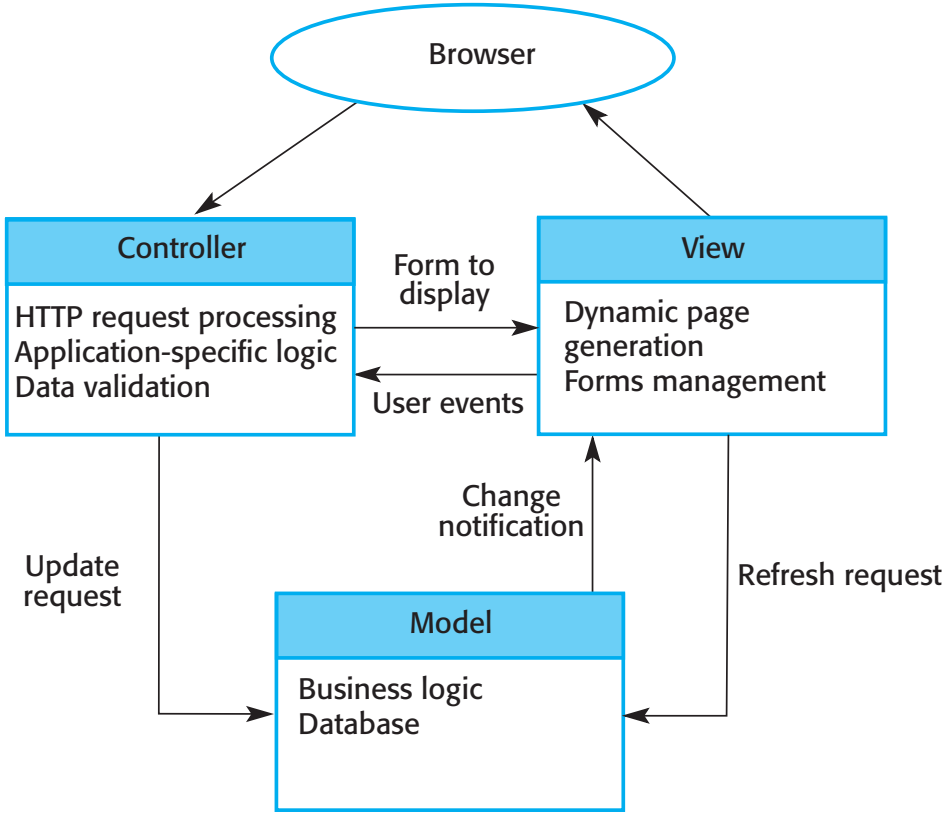
The Model-View-Controller (MVC) pattern

Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. See Figure 6.3.
Example	Figure 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

The organization of the Model-View-Controller



Web application architecture using the MVC pattern



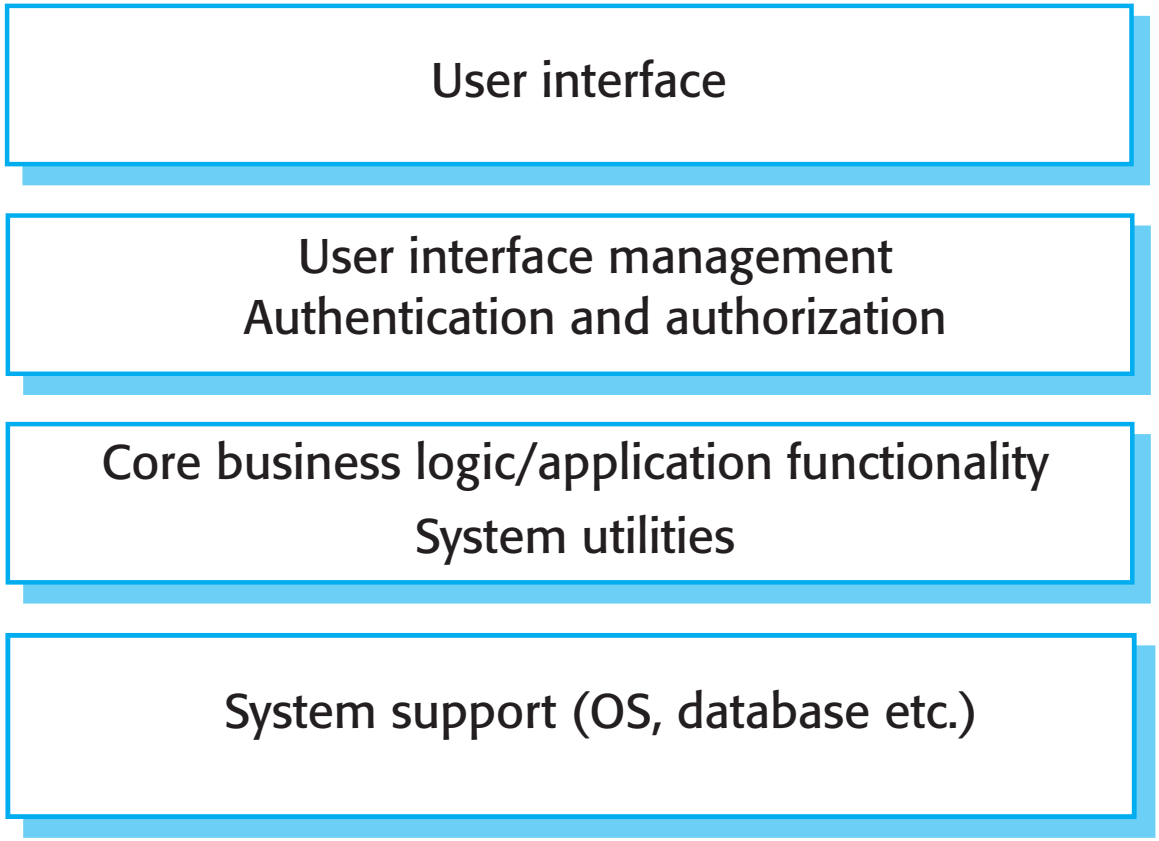
Layered architecture

- Used to model the interfacing of sub-systems.
- Organises the system into a set of layers (or abstract machines) each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- However, often artificial to structure systems in this way.

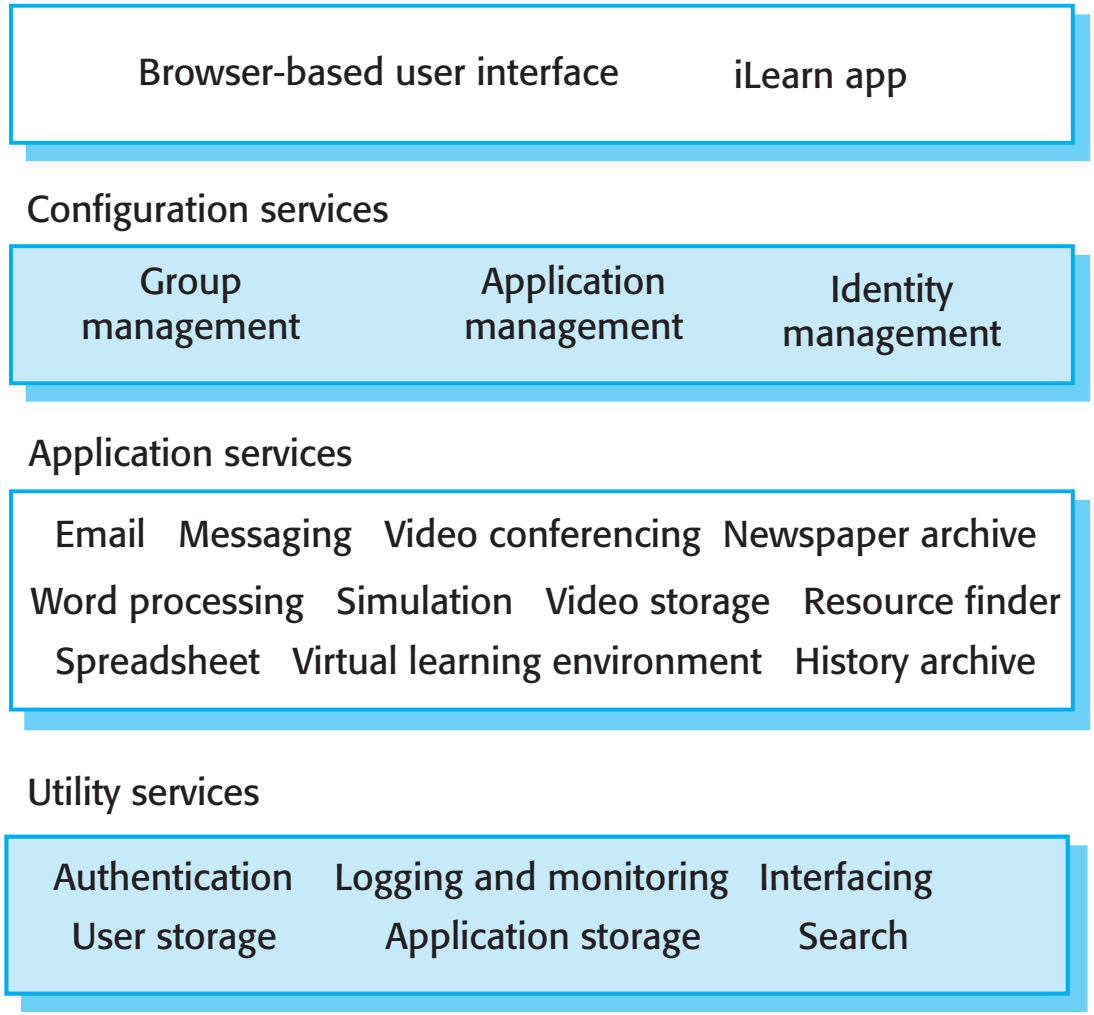
The Layered architecture pattern

Name	Layered architecture
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. See Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in different libraries, as shown in Figure 6.7.
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

A generic layered architecture



The architecture of the iLearn system



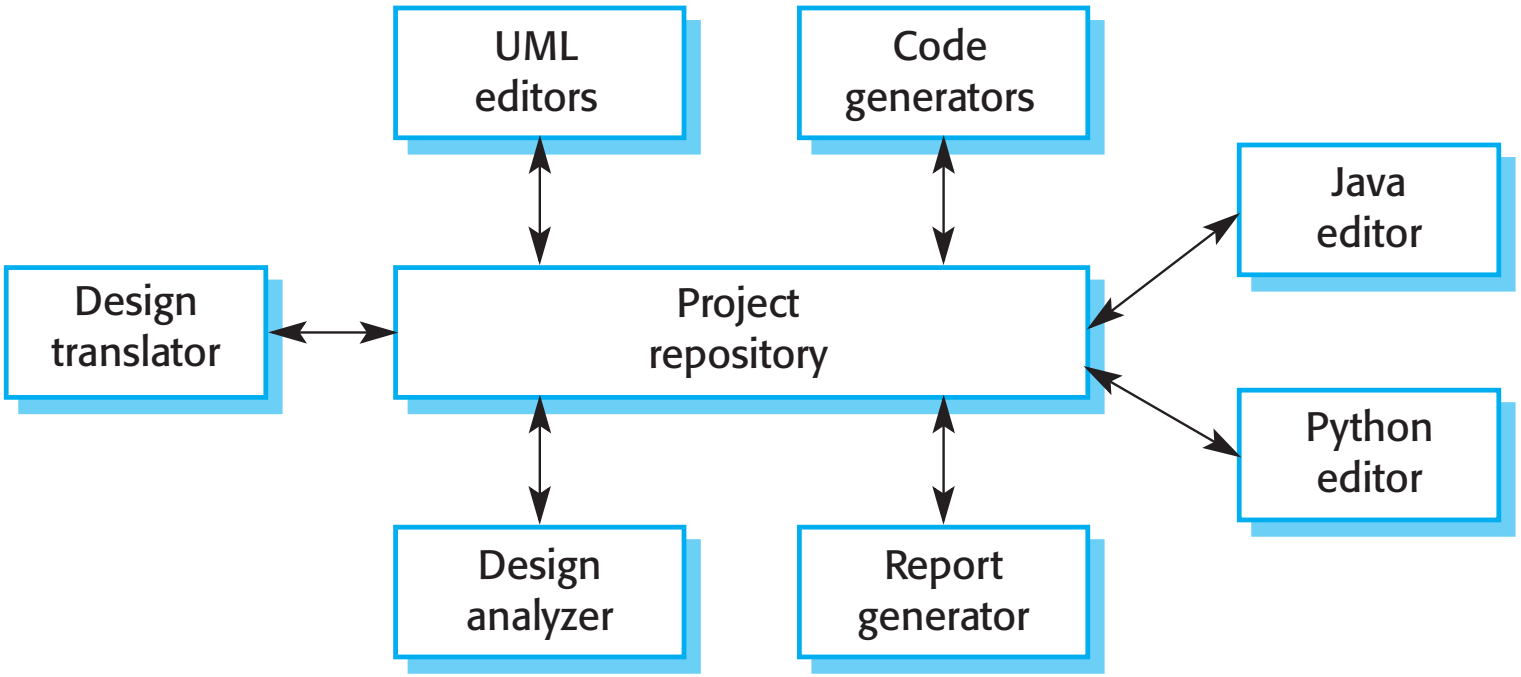
Repository architecture

- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems;
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared, the repository model of sharing is most commonly used a this is an efficient data sharing mechanism.

The Repository pattern

Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	Figure 6.9 is an example of an IDE where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools.
When used	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

A repository architecture for an IDE



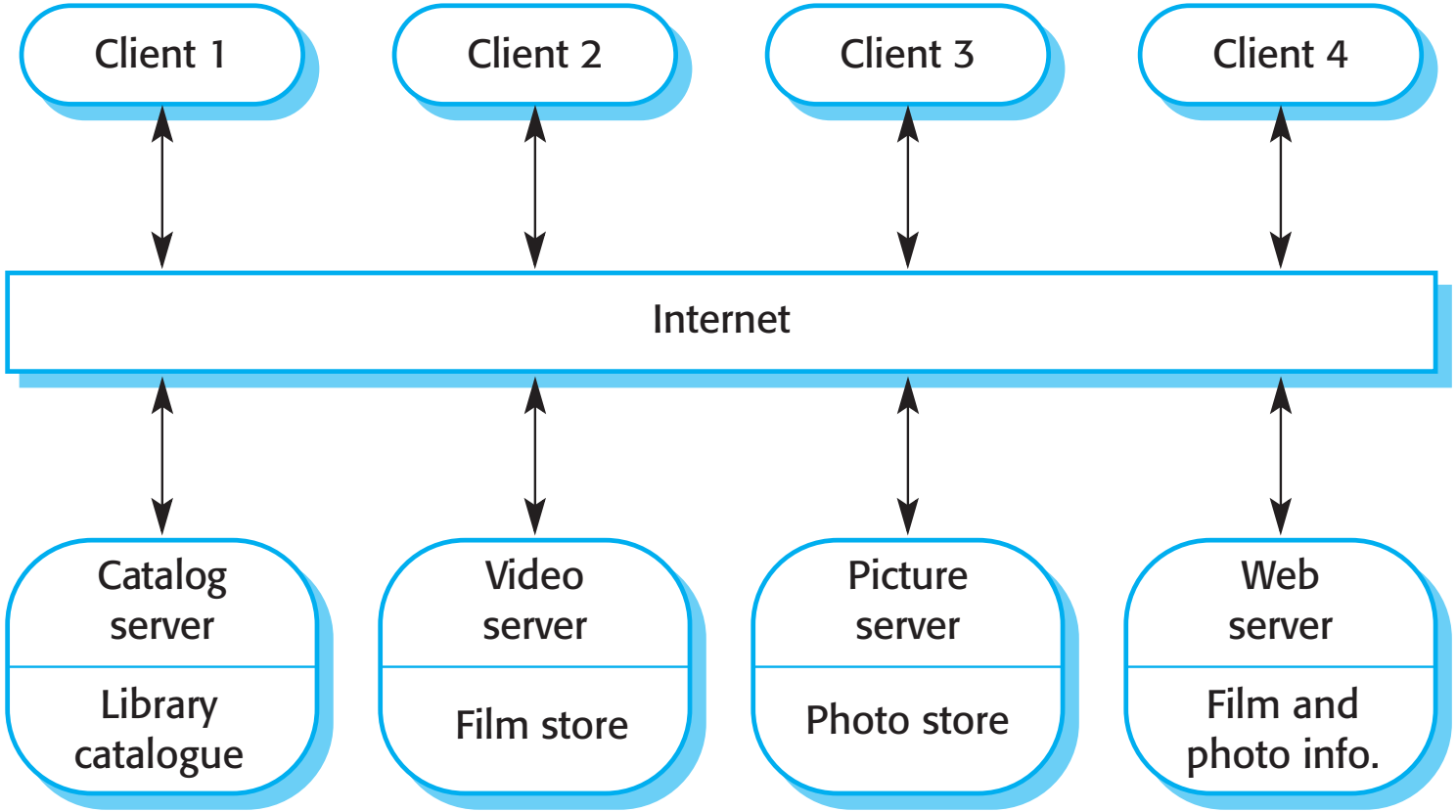
Client-server architecture

- Distributed system model which shows how data and processing is distributed across a range of components.
  - Can be implemented on a single computer.
- Set of stand-alone servers which provide specific services such as printing, data management, etc.
- Set of clients which call on these services.
- Network which allows clients to access servers.

The Client–server pattern

Name	Client-server
Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
Example	Figure 6.11 is an example of a film and video/DVD library organized as a client–server system.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.

A client–server architecture for a film library



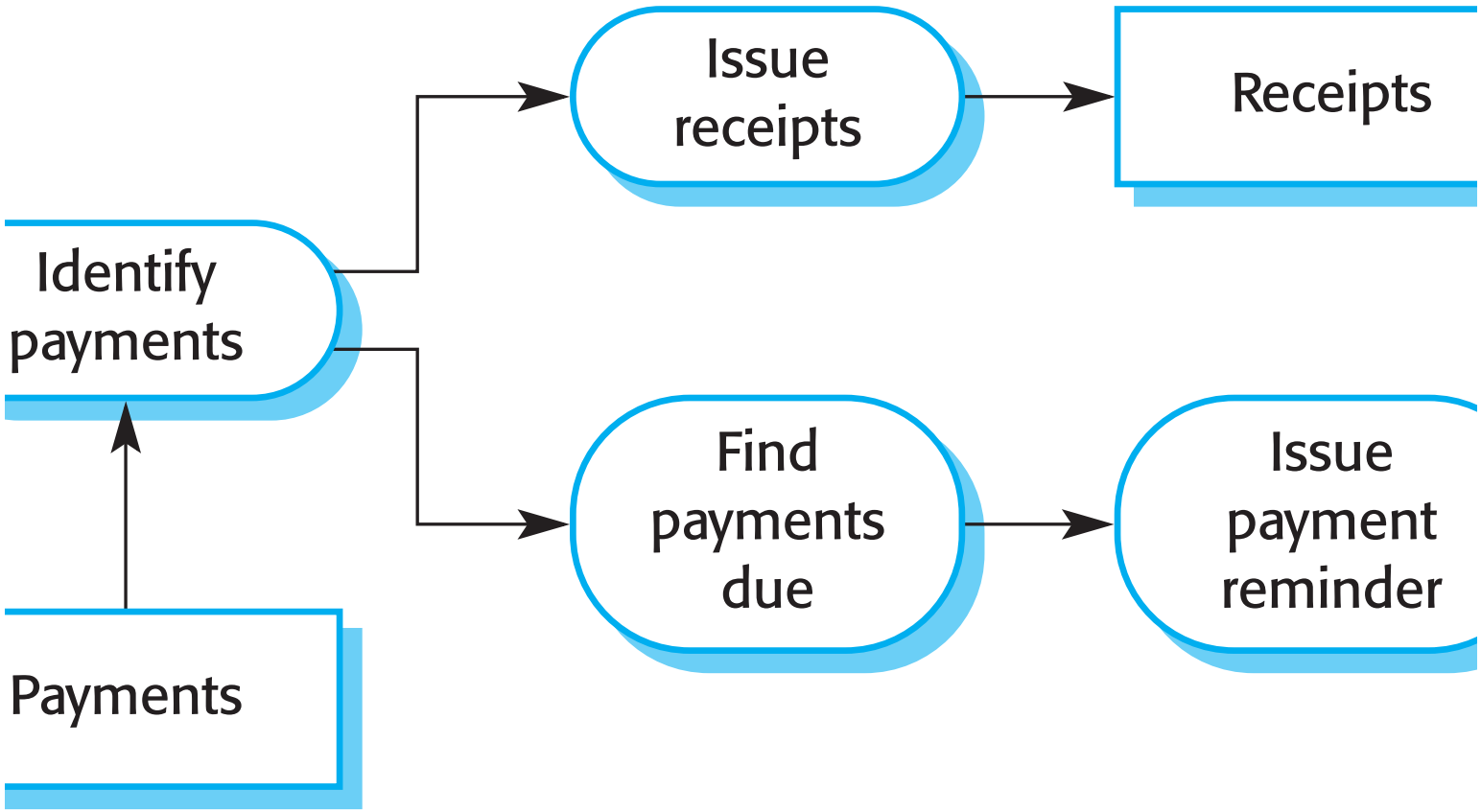
Pipe and filter architecture

- Functional transformations process their inputs to produce outputs.
- May be referred to as a pipe and filter model (as in UNIX shell).
- Variants of this approach are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems.
- Not really suitable for interactive systems.

The pipe and filter pattern

Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	Figure 6.13 is an example of a pipe and filter system used for processing invoices.
When used	Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.

An example of the pipe and filter architecture used in a payments system



Application architectures

Application architectures

- Application systems are designed to meet an organizational need.
- **As businesses have much in common, their application systems also tend to have a common architecture that reflects the application requirements.**
- A *generic application architecture is an architecture for a type of software system that may be configured and adapted* to create a system that meets specific requirements.

Use of application architectures

- As a starting point for architectural design.
- As a design checklist.
- As a way of organising the work of the development team.
- As a means of assessing components for reuse.
- As a vocabulary for talking about application types.

Examples of application types

- Data processing applications
  - Data driven applications that process data in batches without explicit user intervention during the processing.
- Transaction processing applications
  - Data-centred applications that process user requests and update information in a system database.
- Event processing systems
  - Applications where system actions depend on interpreting events from the system's environment.
- Language processing systems
  - Applications where the users' intentions are specified in a formal language that is processed and interpreted by the system.

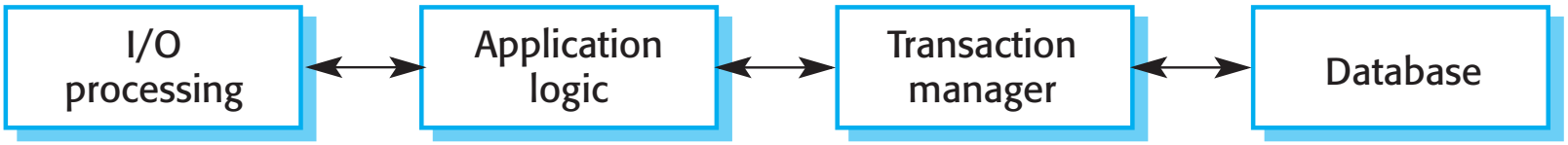
Application type examples

- Two very widely used generic application architectures are transaction processing systems and language processing systems.
- Transaction processing systems
  - E-commerce systems;
  - Reservation systems.
- Language processing systems
  - Compilers;
  - Command interpreters.

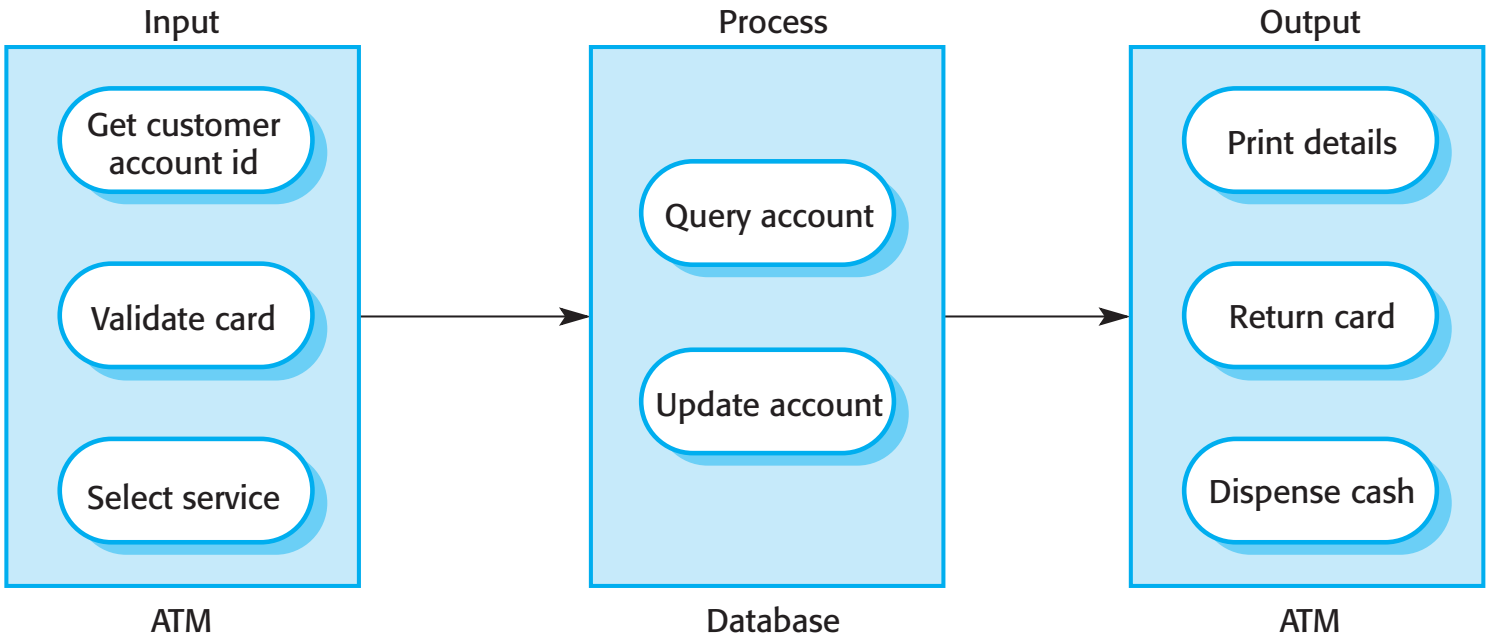
Transaction processing systems

- Process user requests for information from a database or requests to update the database.
- From a user perspective a transaction is:
  - Any coherent sequence of operations that satisfies a goal;
  - For example - find the times of flights from London to Paris.
- Users make asynchronous requests for service which are then processed by a transaction manager.

The structure of transaction processing applications



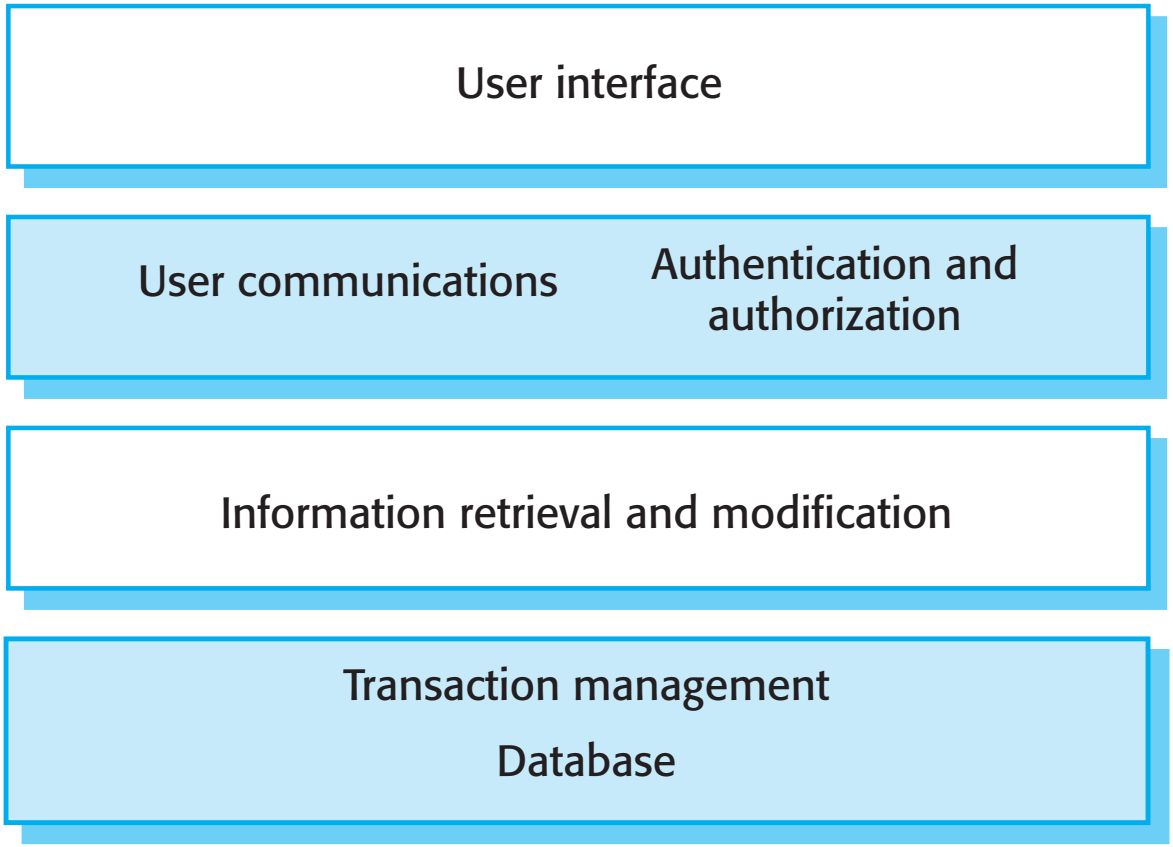
The software architecture of an ATM system



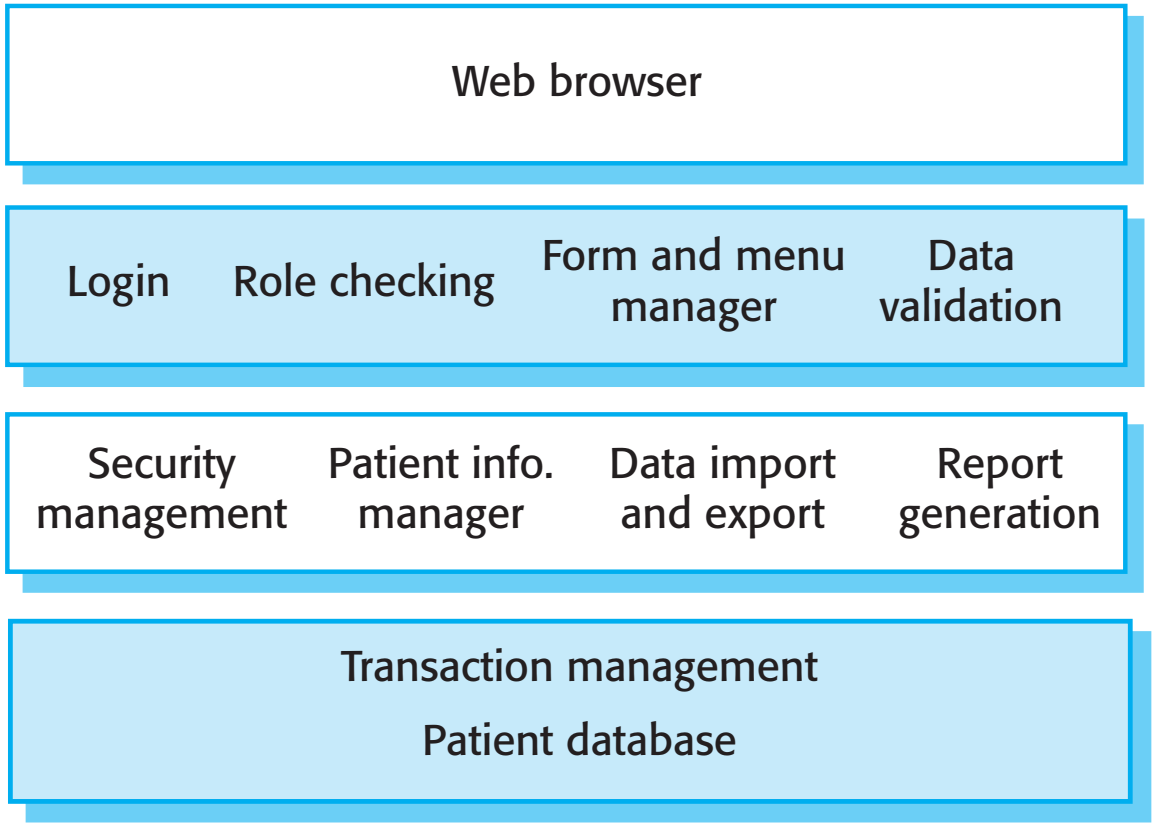
Information systems architecture

- Information systems have a generic architecture that can be organized as a layered architecture.
- These are transaction-based systems as interaction with these systems generally involves database transactions.
- Layers include:
  - The user interface
  - User communications
  - Information retrieval
  - System database

Layered information system architecture



The architecture of the Mentcare system



Web-based information systems

- Information and resource management systems are now usually web-based systems where the user interfaces are implemented using a web browser.
- For example, e-commerce systems are Internet-based resource management systems that accept electronic orders for goods or services and then arrange delivery of these goods or services to the customer.
- In an e-commerce system, the application-specific layer includes additional functionality supporting a ‘shopping cart’ in which users can place a number of items in separate transactions, then pay for them all together in a single transaction.

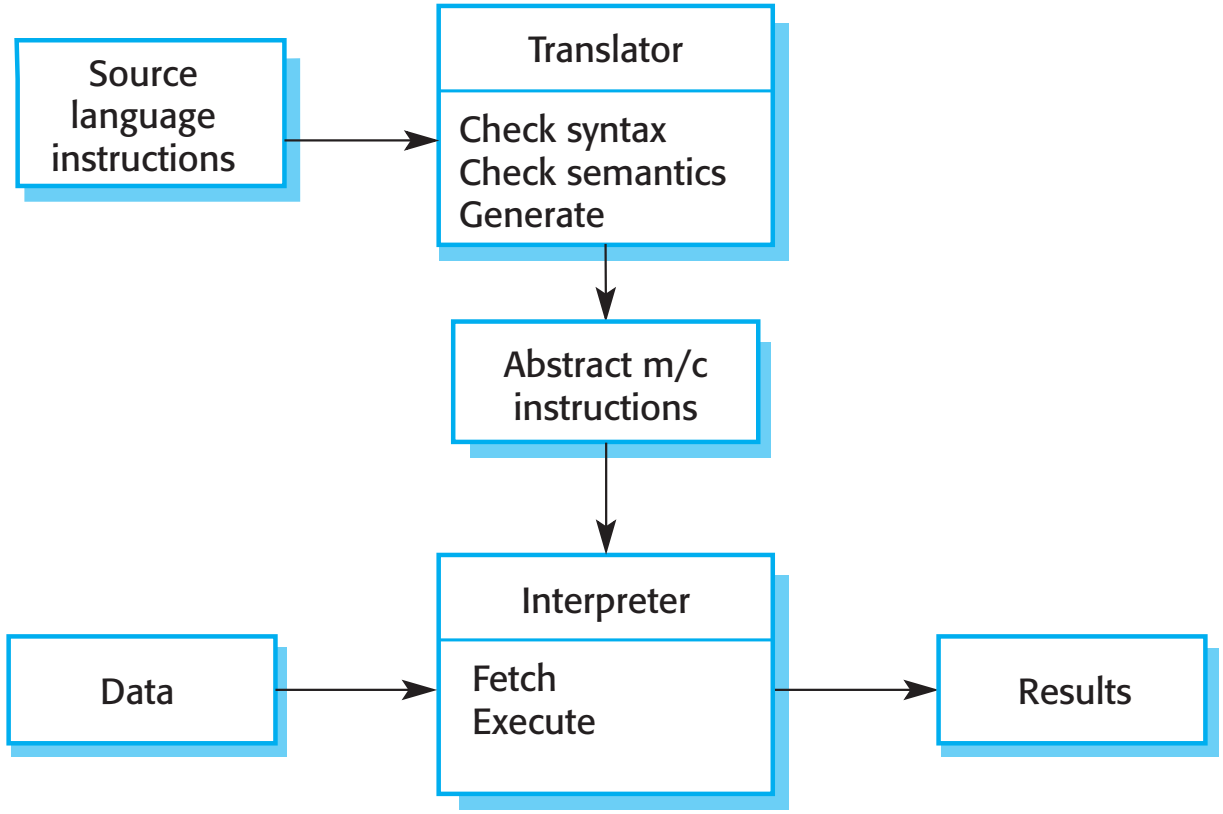
Server implementation

- These systems are often implemented as multi-tier client server/architectures (discussed in Chapter 17)
  - The web server is responsible for all user communications, with the user interface implemented using a web browser;
  - The application server is responsible for implementing application-specific logic as well as information storage and retrieval requests;
  - The database server moves information to and from the database and handles transaction management.

Language processing systems

- Accept a natural or artificial language as input and generate some other representation of that language.
- May include an interpreter to act on the instructions in the language that is being processed.
- Used in situations where the easiest way to solve a problem is to describe an algorithm or describe the system data
  - Meta-case tools process tool descriptions, method rules, etc and generate tools.

The architecture of a language processing system



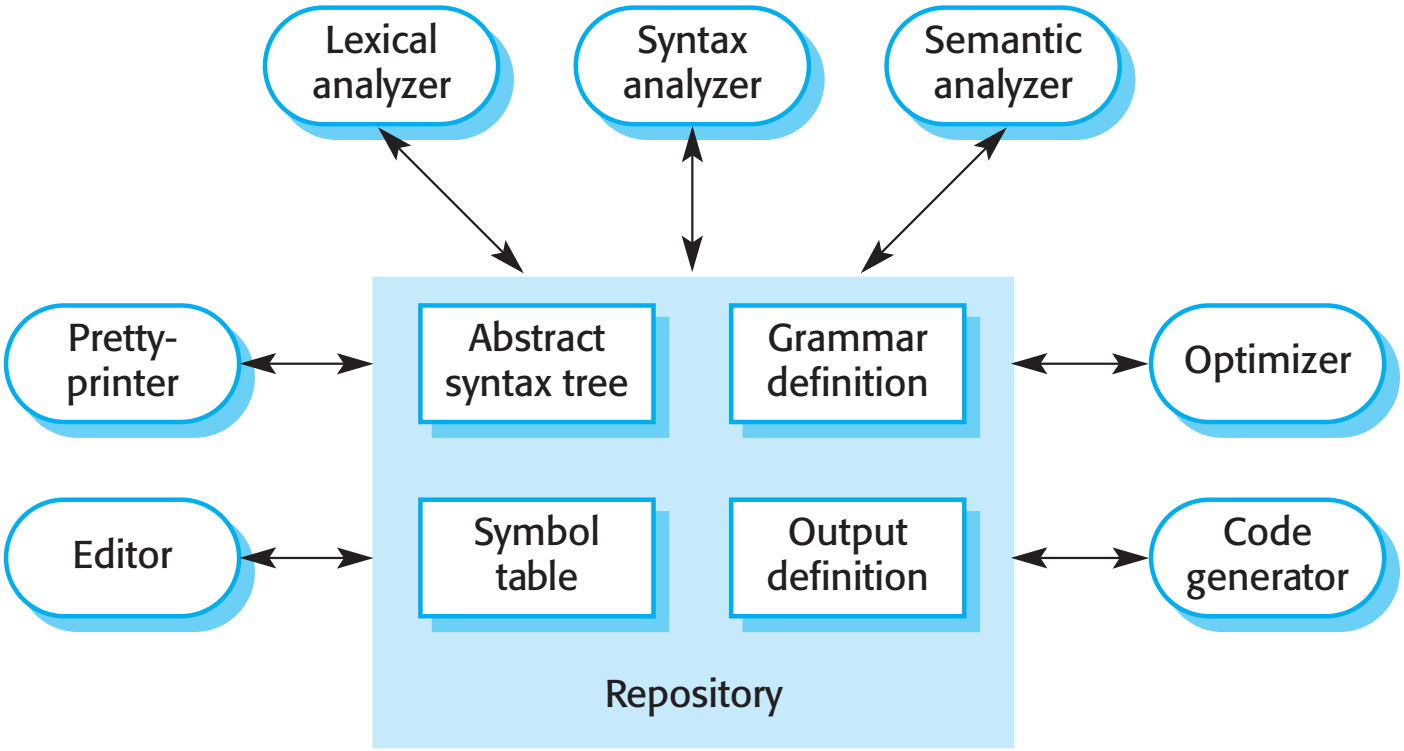
Compiler components

- A lexical analyzer, which takes input language tokens and converts them to an internal form.
- A symbol table, which holds information about the names of entities (variables, class names, object names, etc.) used in the text that is being translated.
- A syntax analyzer, which checks the syntax of the language being translated.
- A syntax tree, which is an internal structure representing the program being compiled.

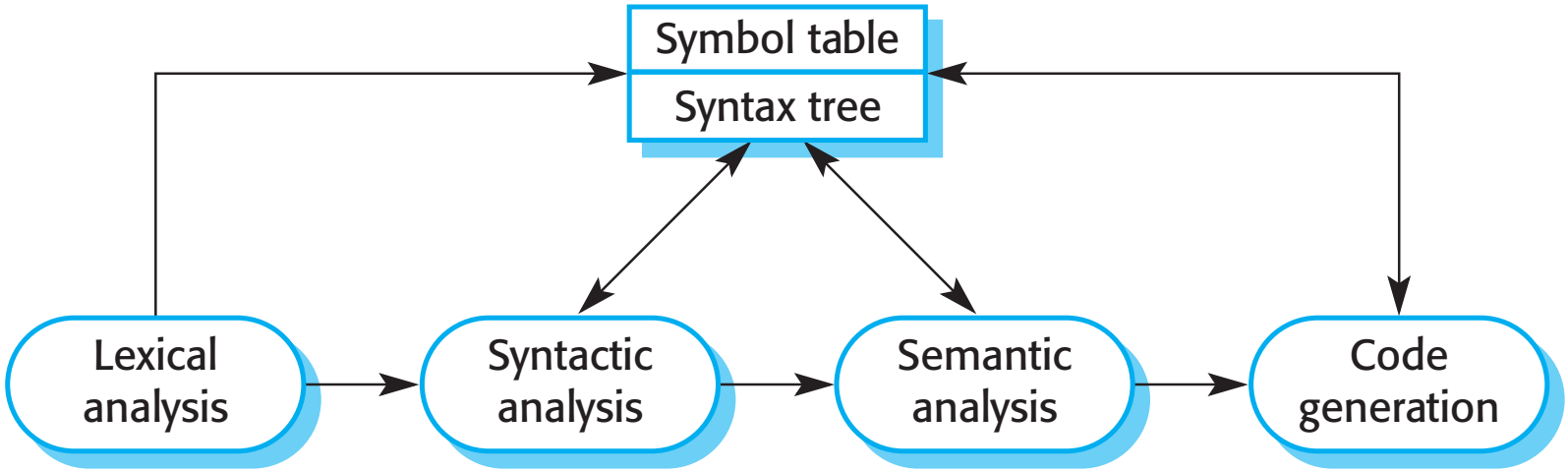
Compiler components

- A semantic analyzer that uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.
- A code generator that 'walks' the syntax tree and generates abstract machine code.

A repository architecture for a language processing system



A pipe and filter compiler architecture



Key points

- A software architecture is a description of how a software system is organized.
- Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used.
- Architectures may be documented from several different perspectives or views such as a conceptual view, a logical view, a process view, and a development view.
- Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used and describe its advantages and disadvantages.

Key points

- Models of application systems architectures help us understand and compare applications, validate application system designs and assess large-scale components for reuse.
- Transaction processing systems are interactive systems that allow information in a database to be remotely accessed and modified by a number of users.
- Language processing systems are used to translate texts from one language into another and to carry out the instructions specified in the input language. They include a translator and an abstract machine that executes the generated language.

About Week 5 – February 17th

- TEAM Homework Assignments
  - HW4 - Create a set of Models (team of 4 does all 4, everyone else do the System Architecture + 2 others)
    - **System Architecture Block** diagram – an overarching box / block diagram of all the major components you envision for your system. Components and relationships.
    - **Architectural Pattern** diagram – decide which pattern is most applicable to your project and model accordingly (MVC, Layered, Client/Server, etc)
    - **Use Case** diagram with Tabular Description of the Use Case. Something not too simple.
    - **Class** diagram – first pass at what you think the key objects are in your project and their properties / methods. A class hierarchy would be a nice addition.
    - Submit as a .pdf attachments to the Sakai Assignment (doesn't matter what tools you use to create, just save as a .pdf) If you have Acrobat, please COMBINE into one .pdf file.
    - Submit as a TEAM, but each individual deliverable needs to be assigned to one person to own.
    - Let me know in the document and/or in Sakai who worked on what
    - Grade based on completeness, detail, complexity. 40 points / 34 base, DUE Feb 16th
  - HW5 - Create your first code drop or "sprint"
    - Based on your Homework 3 user story / tasks
    - Running code in Git – we will create the Git projects for you!
    - Reply in Sakai when complete, NOT DUE UNTIL FEB 23rd!
    - 60 points / 52 base
- Reading Assignment
  - Software Engineering, 10th Edition, Ian Sommerville
    - Chapters 7 – "Design and Implementation"