# Software Engineering COMP 201

Lecturer: **Sebastian Coope**
*Ashton Building, Room G.18*
*E-mail:* ***coopes@liverpool.ac.uk***

**COMP 201 web-page:**
**http://www.csc.liv.ac.uk/~coopes/comp201**

Lecture 7 – System Models

# Lecture Overview

- **System models** are abstract descriptions of systems whose requirements are being analysed
- Objectives - To explain why the context of a system should be modelled as part of the RE process
- To describe
  - Behavioural modelling (FSM, Petri-nets),
  - Data modelling and
  - Object modelling (Unified Modelling Language, UML)

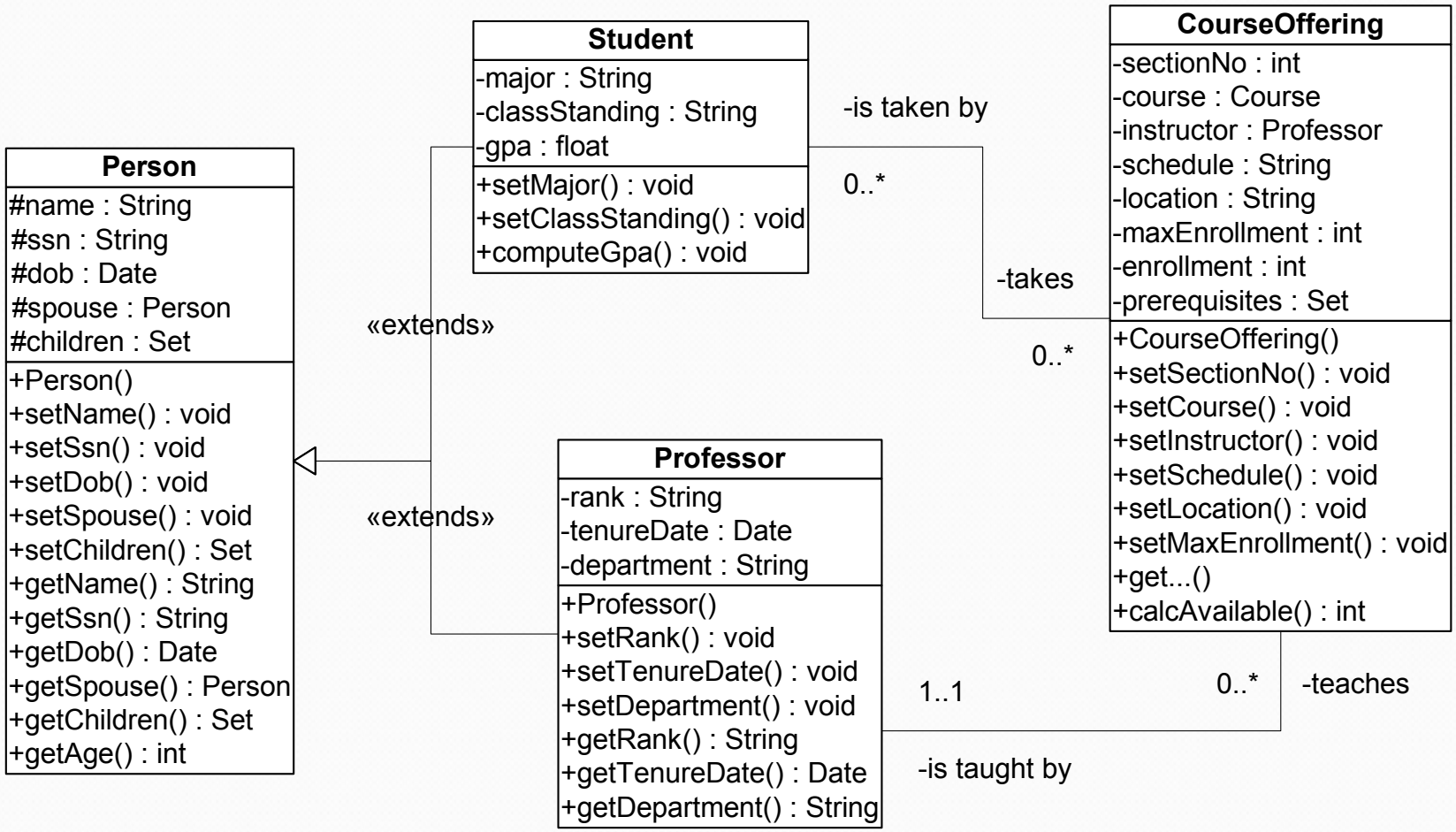
# System Models

- User requirements must be written in such a way that non-technical experts can understand them, e.g., by using natural language
- Detailed system requirements may be expressed in a more technical way however
  - One widely used technique is to document the system specification as a set of system models
  - These are graphical representations which describe business processes and the system to be developed
  - They are an important bridge between the analysis and design processes

# System Modelling

- **System modelling** helps the analyst to understand the functionality of the system and models are used to communicate with customers
- Different models present the system from different perspectives:
  - **External perspective** showing the system's context or environment
  - **Behavioural perspective** showing the behaviour of the system
  - **Structural perspective** showing the system or data architecture

# “A Picture Paints a Thousand Words”



# System Model Advantages

- They can be easier to understand than using a verbose natural language description
- System models can leave out unnecessary details of the system so way may focus on **what is important**
  - A system **representation** should maintain all the information of a system
  - An **abstraction** deliberately simplifies the system and picks out its most salient characteristics
- Different models can focus on different approaches to abstraction

# System Model Weaknesses

- They do not model **non-functional** system requirements
- They do not usually include information about whether a method is appropriate for a given problem
- They may produce too much documentation
- System models are sometimes too detailed and difficult for users to understand

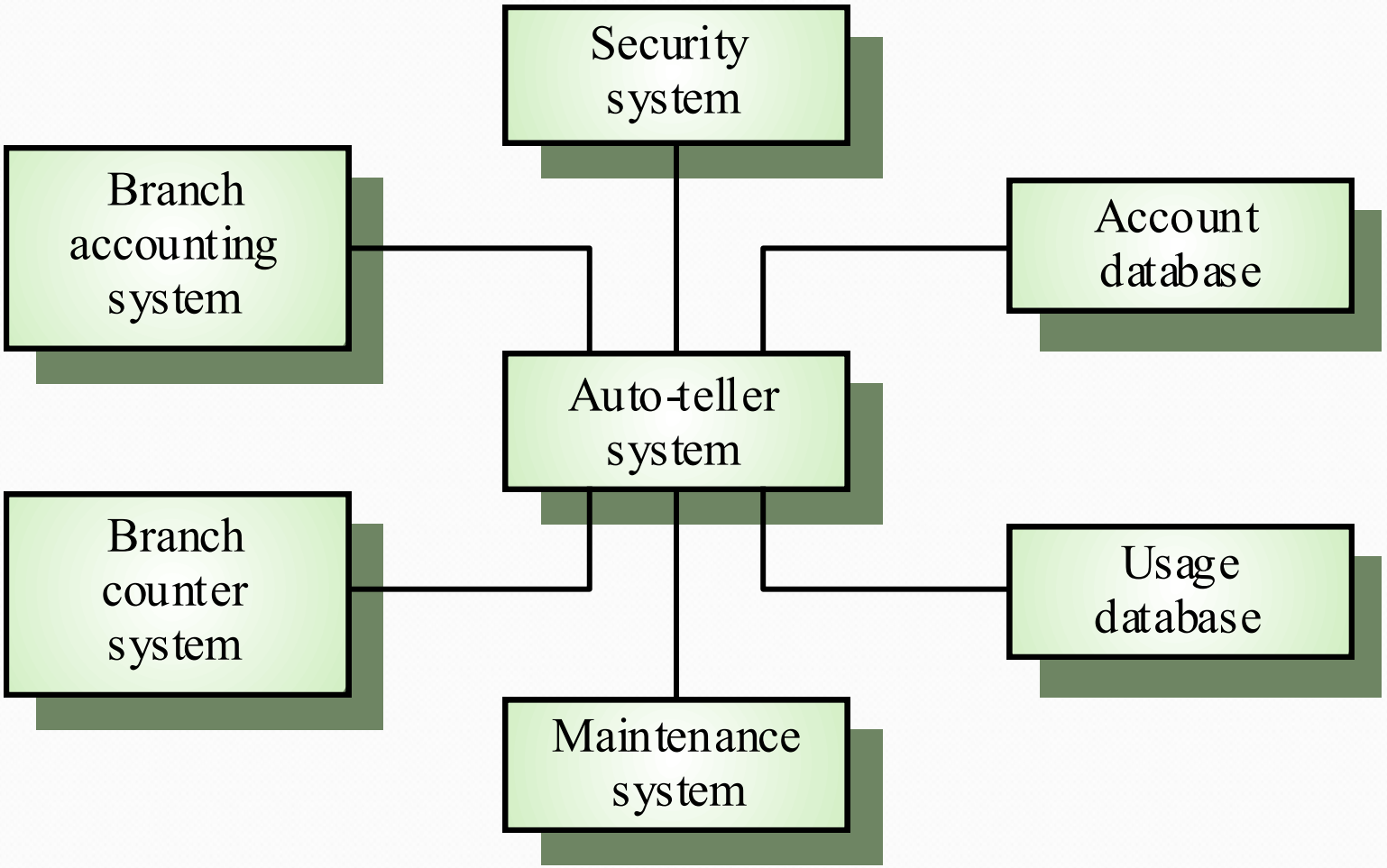
# Model Types

- **Data processing model** - showing how the data is processed at different stages
- **Composition model** - showing how entities are composed of other entities
- **Architectural model** - showing principal sub-systems
- **Classification model** - showing how entities have common characteristics
- **Stimulus/response model** - showing the system's reaction to events

# Context Models

- **Context models** are used to illustrate the boundaries of a system
  - Identifying the boundaries of the system to be developed is not always straightforward
- **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

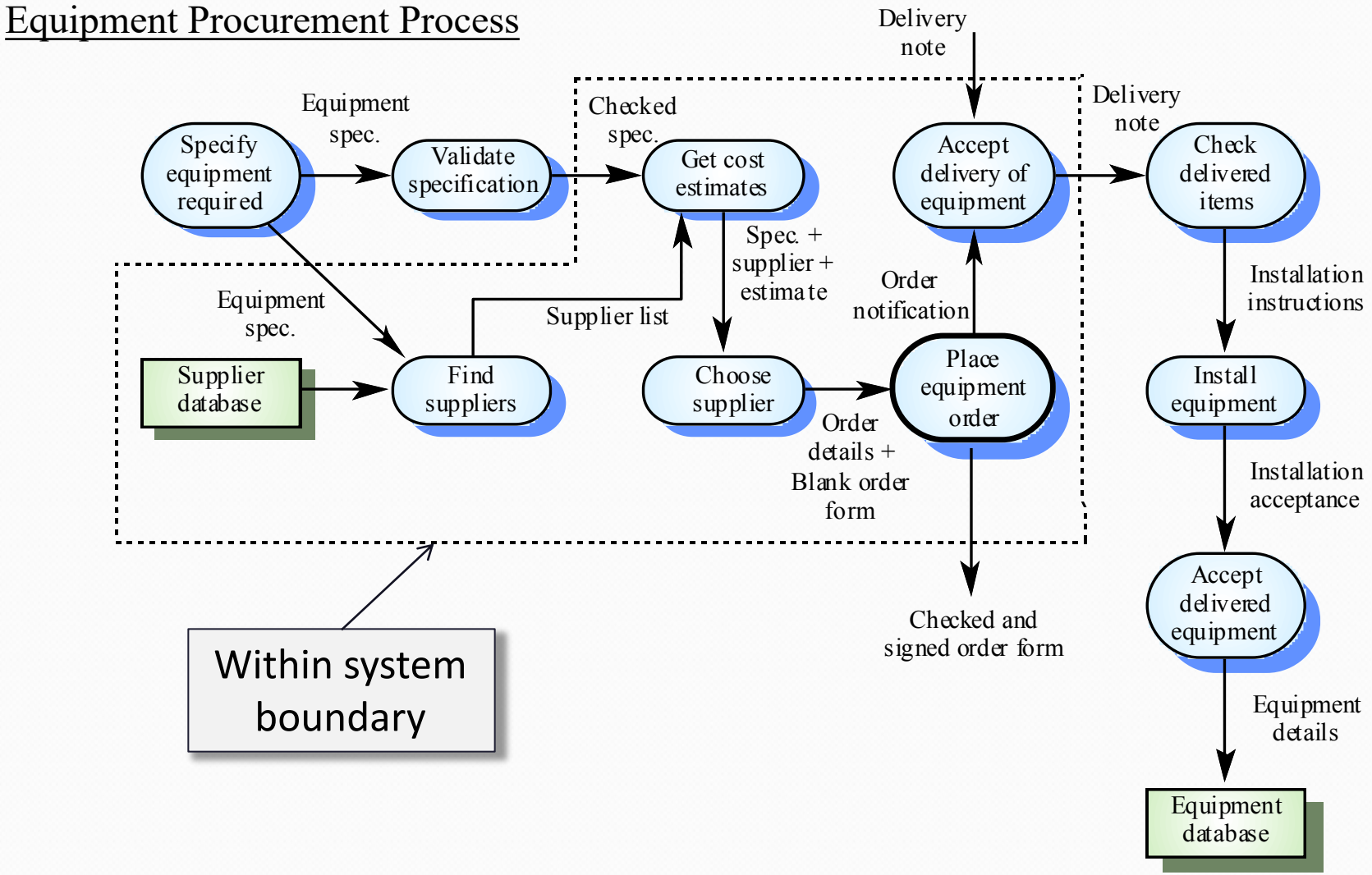
# Example – Architectural Model of an ATM System



# Process Models

- **Process models** show the overall process and the processes that are supported by the system
  - In process models it is implicit 1 process is completed before another process begins
  - Process models are similar to flow charts
- **Data flow models** may be used to show the processes and the **flow of information** from one process to another
  - The data flow models it is implicit that processes will happen in parallel (concurrent processing)

# Example Process Model



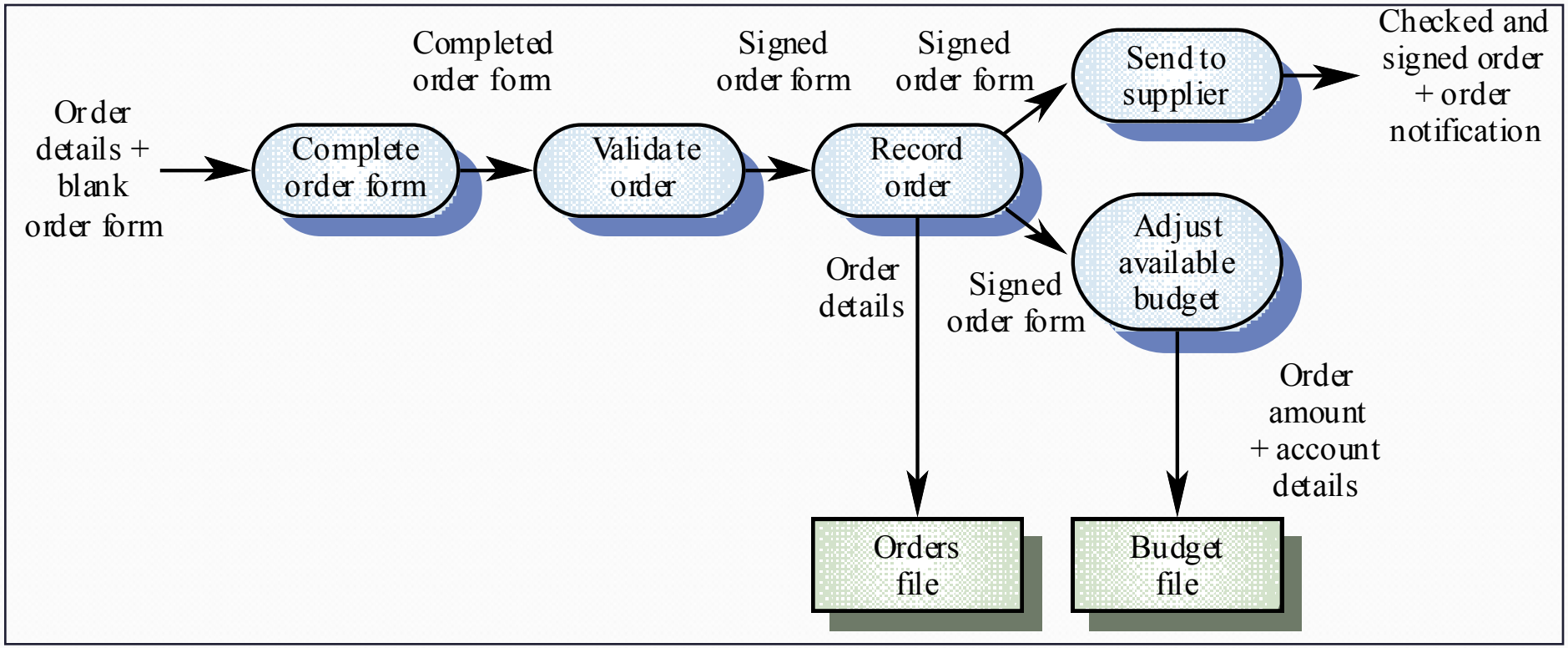
# Behavioural Models

- **Behavioural models** are used to describe the overall behaviour of a system
- Two types of behavioural model
  - Data processing models that show how data is processed as it moves through the system
  - State machine models that show the systems response to events
- **Both** of these models are required for a description of the system's behaviour

# Data-Processing Models

- **Data flow diagrams** are used to model the system's data processing
- These show the processing steps as data flows through a system
- IMPORTANT part of many analysis methods
- **Simple and intuitive notation**
- **Show end-to-end processing of data**

# Example - Order Processing Data Flow Diagram



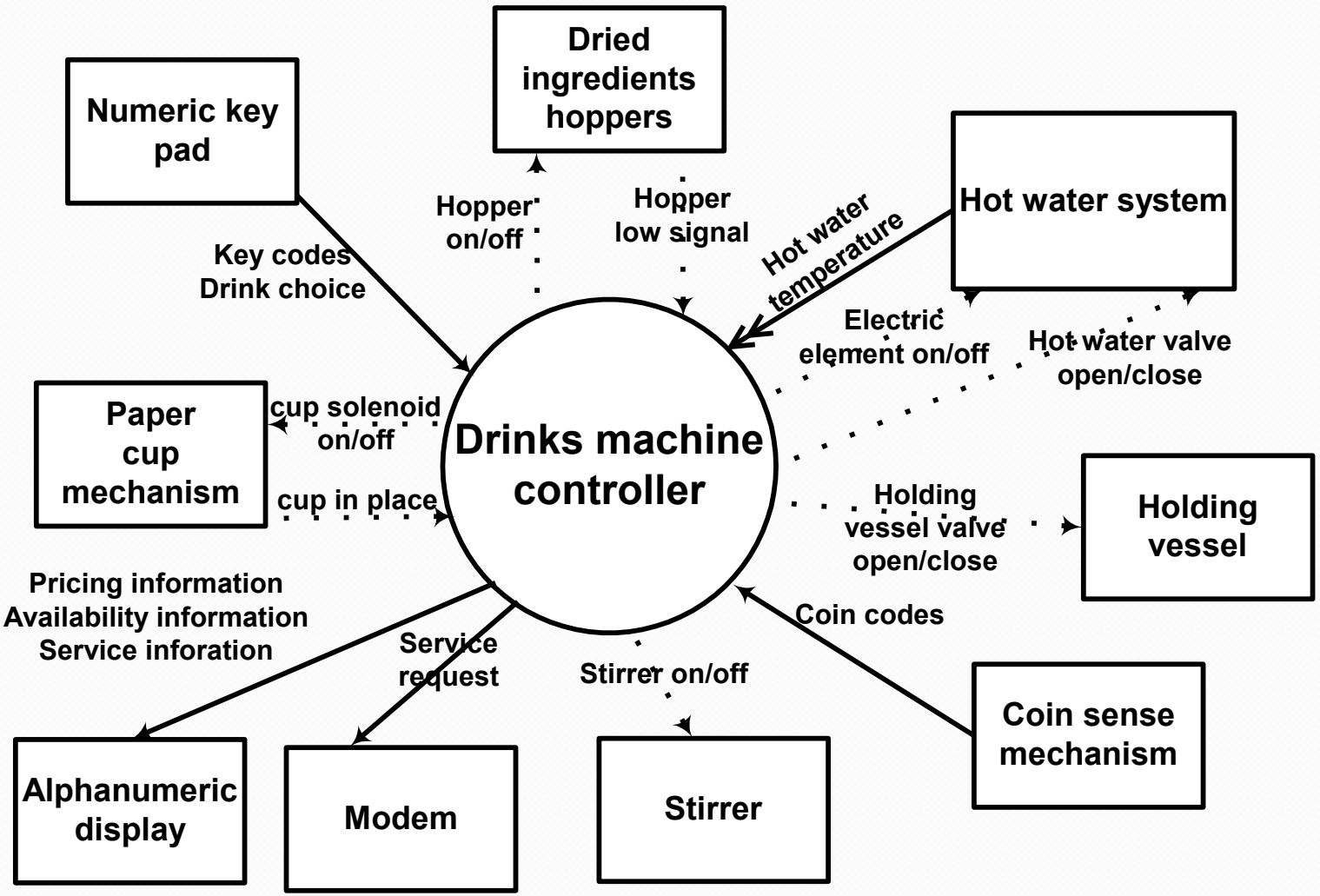
# Data Flow Diagrams

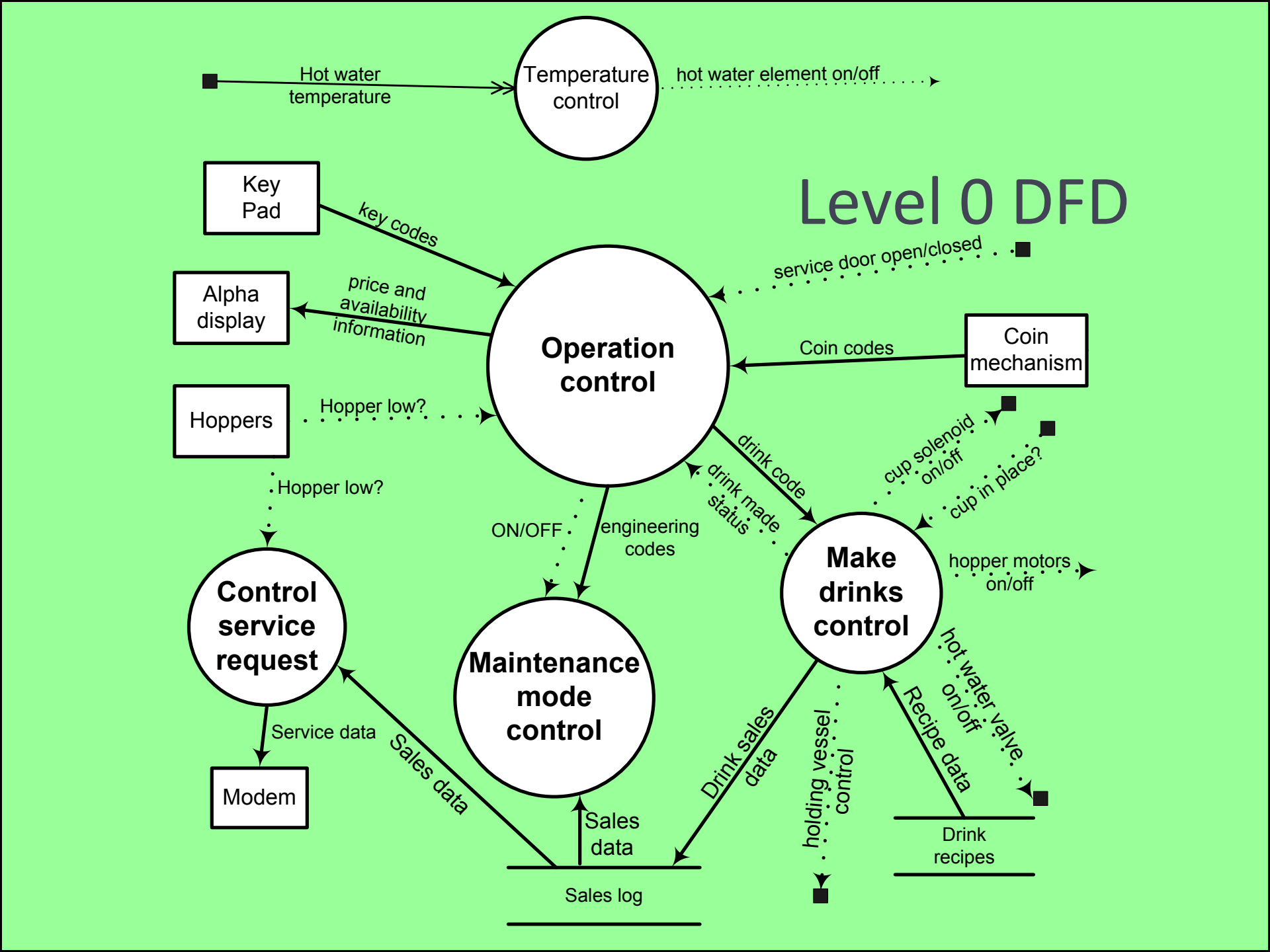
- **Data Flow Diagrams** track and document *how the data associated with a process* is helpful to develop an overall understanding of the system
- Data flow diagrams may also be used in showing the data exchange between a system and *other systems* in its environment

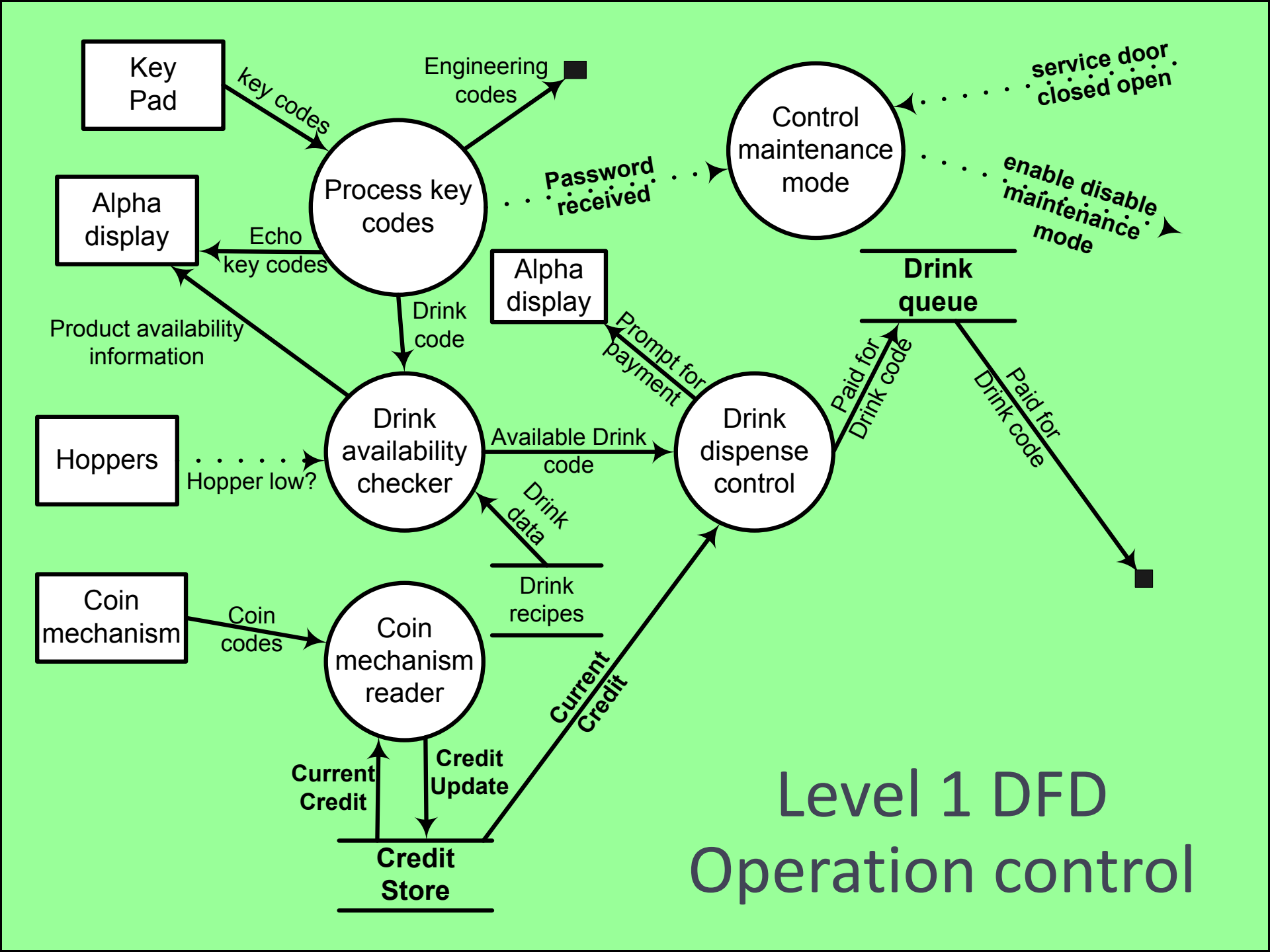
# Data Flow Diagrams

- **Data Flow Diagrams** have an advantage in that they are simple and intuitive and can thus be shown to users who can help in *validating the analysis*
- Developing data flow diagrams is usually a *top-down process*
  - We begin by evaluating the overall process we wish to model before considering sub-processes
- Data flow diagrams show a **functional perspective** where each transformation represents a single function or process which is particularly useful during requirements analysis since it shows end-to-end processing.

# DFD Context diagram




Level 0 DFD
Hot water temperature
Temperature control
hot water element on/off
Key Pad
key codes
Alpha display
price and availability information
Operation control
service door open/closed
Coin mechanism
Coin codes
Hoppers
Hopper low?
Hopper low?
drink code
drink made status
cup solenoid on/off
cup in place?
ON/OFF
engineering codes
Make drinks control
hopper motors on/off
Control service request
Maintenance mode control
hot water valve on/off
Recipe data
Service data
Sales data
Drink sales data
holding vessel control
Modem
Sales data
Drink recipes
Sales log


Key Pad
key codes
Engineering codes
Process key codes
Alpha display
Echo key codes
Product availability information
Drink code
Password received
Control maintenance mode
service door closed open
enable disable maintenance mode
Alpha display
Prompt for payment
Drink queue
Paid for Drink code
Paid for Drink code
Hoppers
Hopper low?
Drink availability checker
Available Drink code
Drink dispense control
Drink data
Drink recipes
Coin mechanism
Coin codes
Coin mechanism reader
Current Credit
Credit Update
Current Credit
Credit Store
Level 1 DFD
Operation control

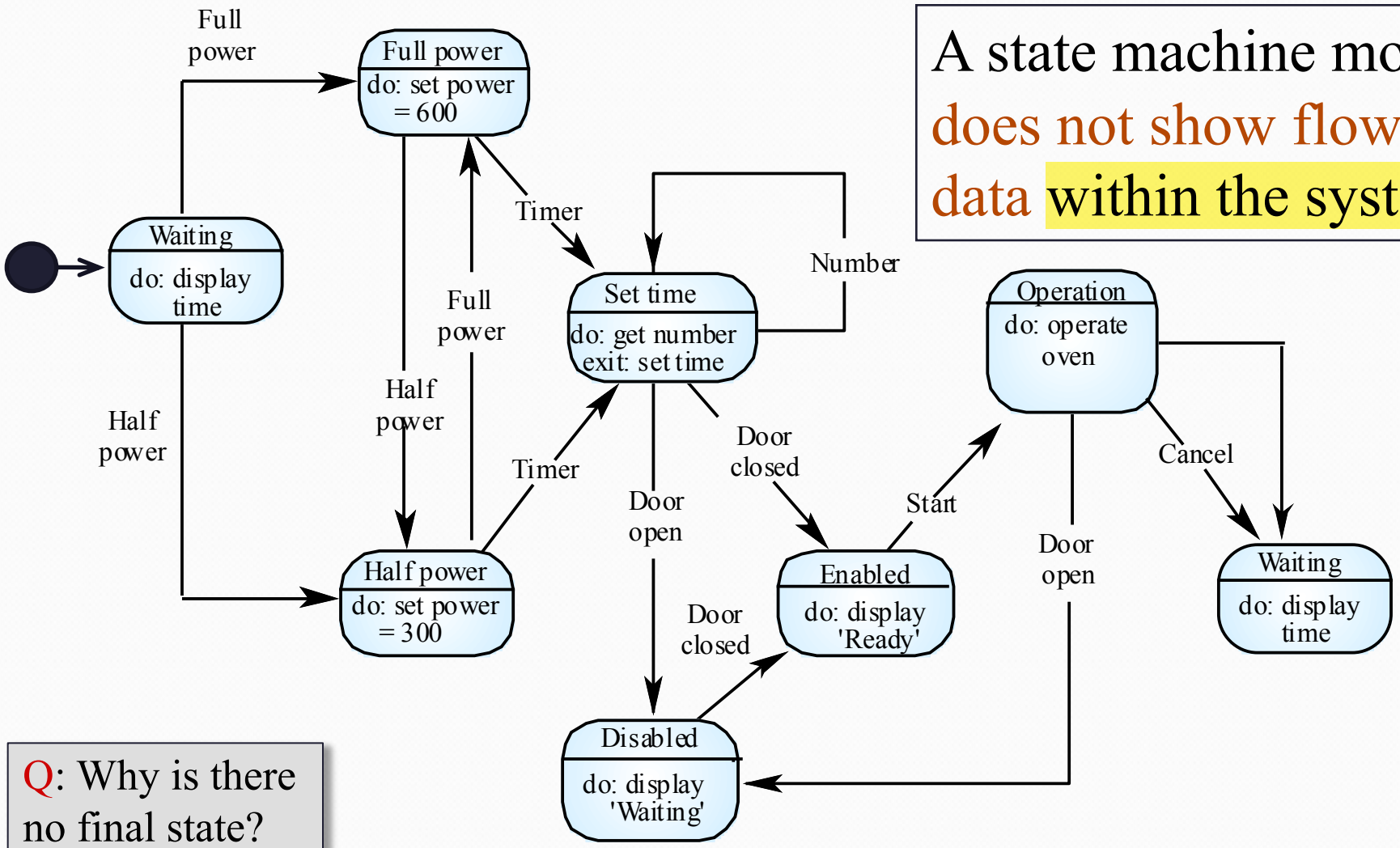
# Statechart Diagrams

- Statechart Diagrams (or State machine models ) show the behaviour of the system in response to external and internal events
- They show the system's responses to stimuli (the event-action paradigm) so are often used for modelling **real-time systems**
- Statechart diagrams 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 **Unified Modeling Language** (UML)

# Statechart Diagrams

- An **initial state** is denoted by a solid circle and is optional (sometimes the system will start in different places and thus the initial state should be omitted).
- If required, a **final state** can also be used; this is denoted by a solid circle with a ring around it.
- We use a level of abstraction so that we can observe the *essential behaviour* of the system we want to model.
- Rounded rectangles are used for states. Each state contains two components, the **state name** and a brief description of the **action performed** in that state (see next slide).

# Example - Microwave Oven Model




A state machine model does not show flow of data within the system

Q: Why is there no final state?

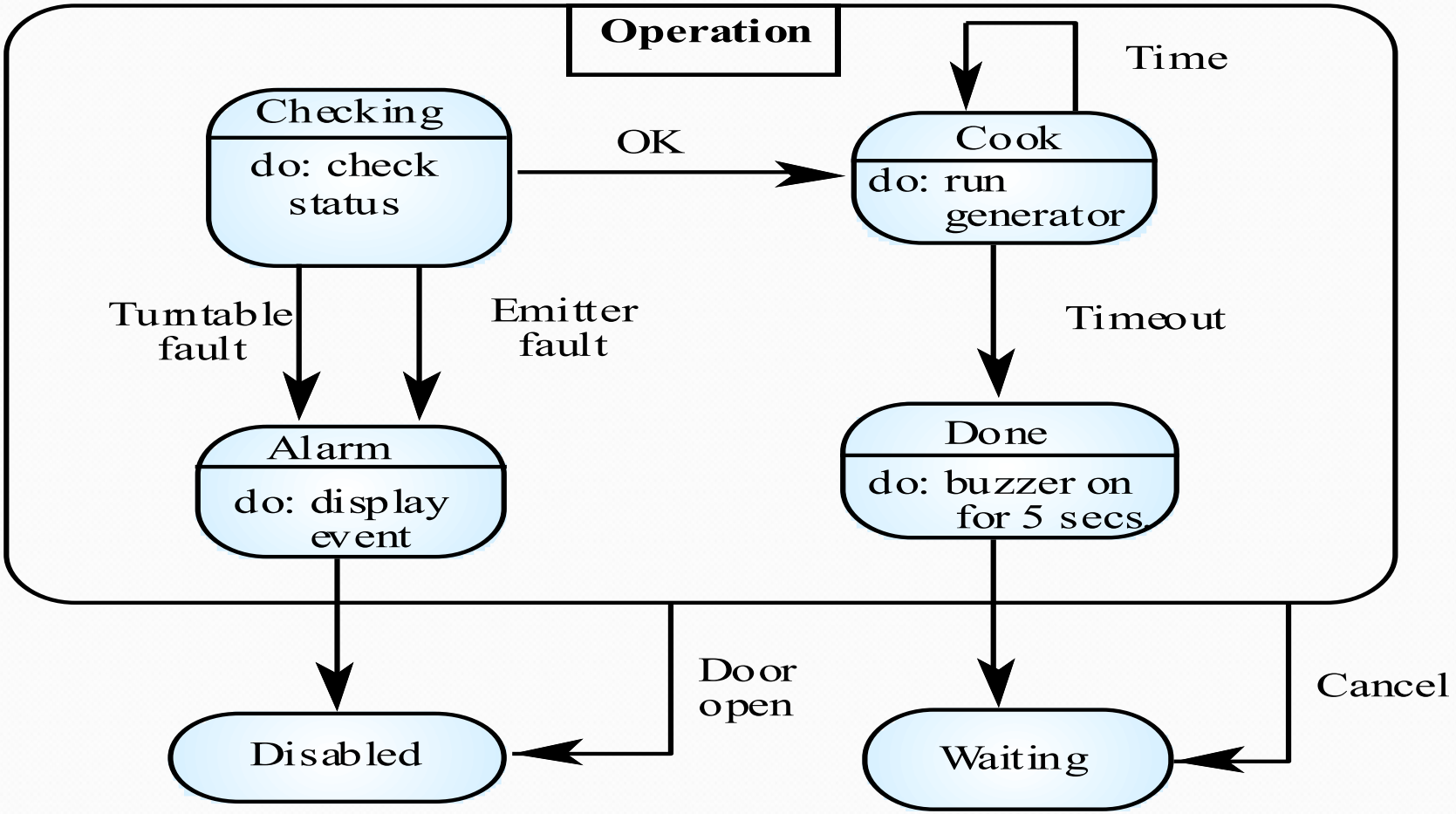
# Microwave Oven Stimuli

| 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 |

# Statecharts

- Statecharts also allow the decomposition of a model into sub-models (see figure on next slide).
- A brief description of the actions is included following the ‘do’ in each state (the word “do” is optional).
- Can be complemented by tables describing the states and the stimuli.

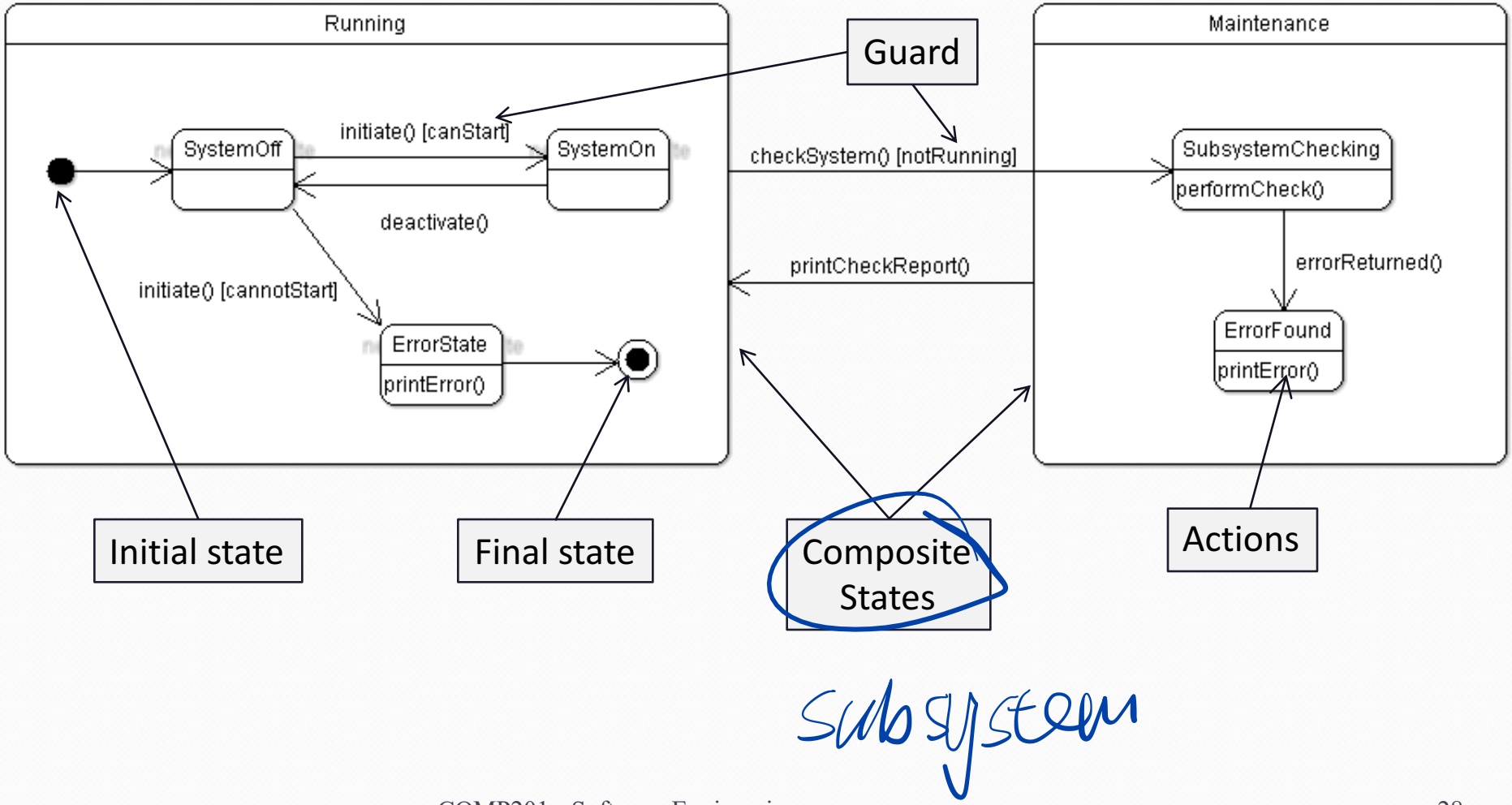
# Statechart Diagram



# Statechart Diagrams

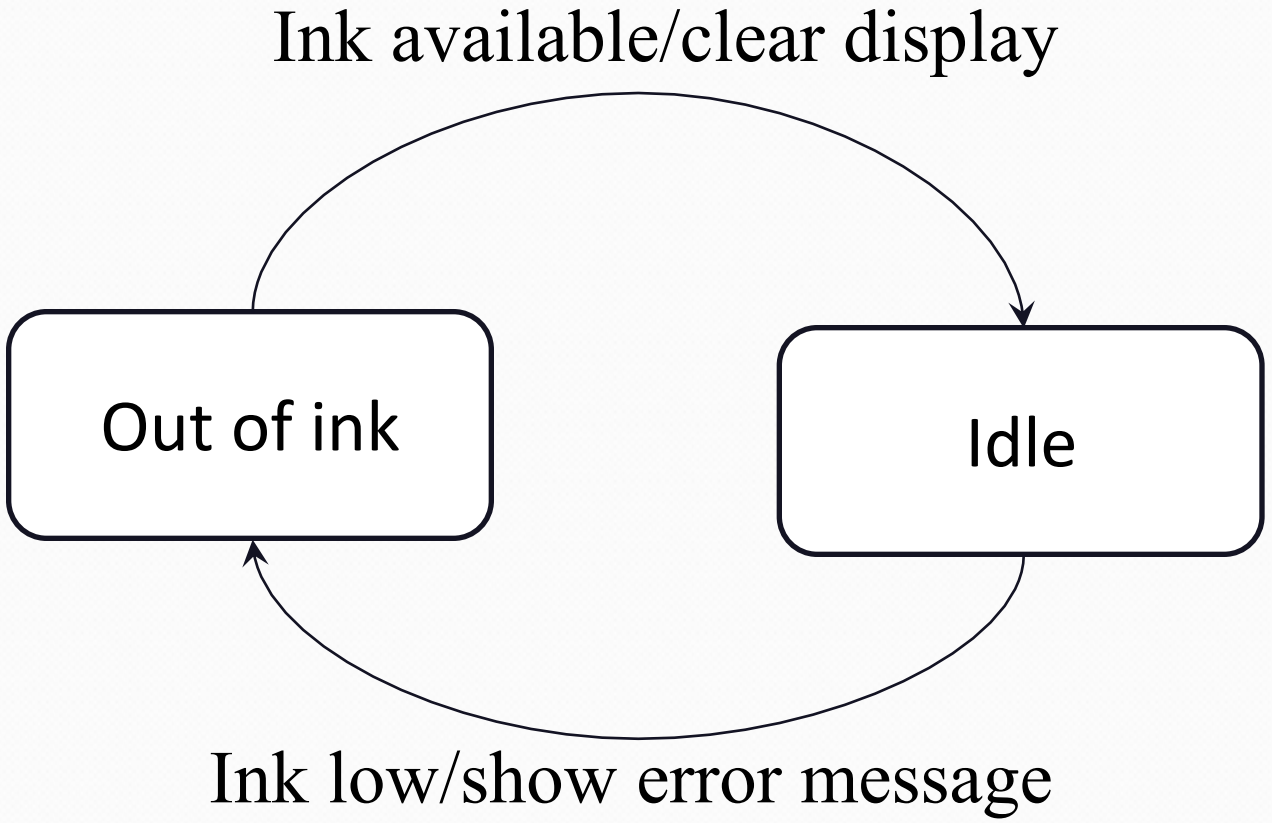
- The **label** on an arc can denote the method called to move from one state to the next (the *event*).
- A **guard** is used to ensure that the system only moves from one state to the other if the expression is satisfied.
- A state can contain a subdiagram within it (also called a **composite state**). This is useful for example when we wish to model a subsystem or substates.
- On the next slide, we can see all these elements of a UML statechart diagram

# Statechart Diagrams



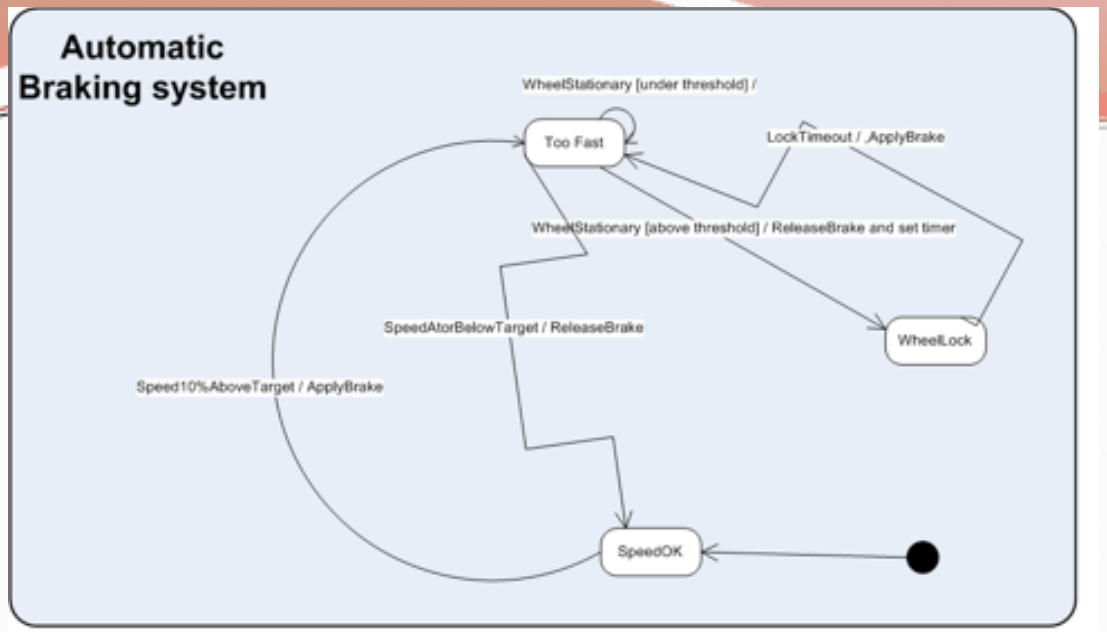
# Actions

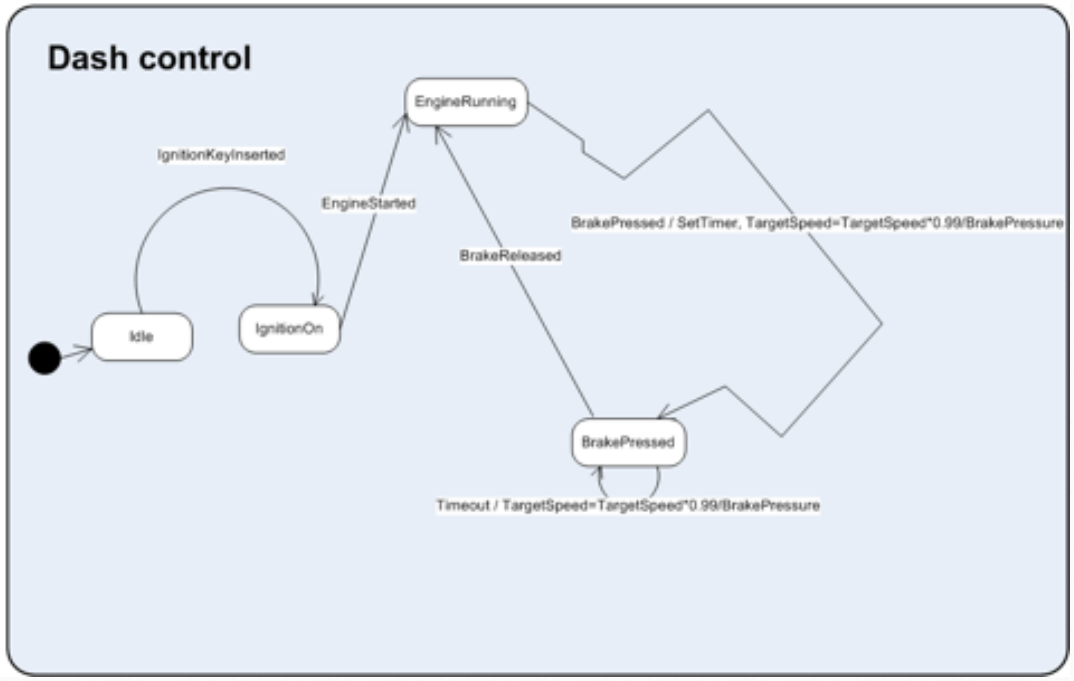
- You can put actions after the event using a /



# More hints on state charts

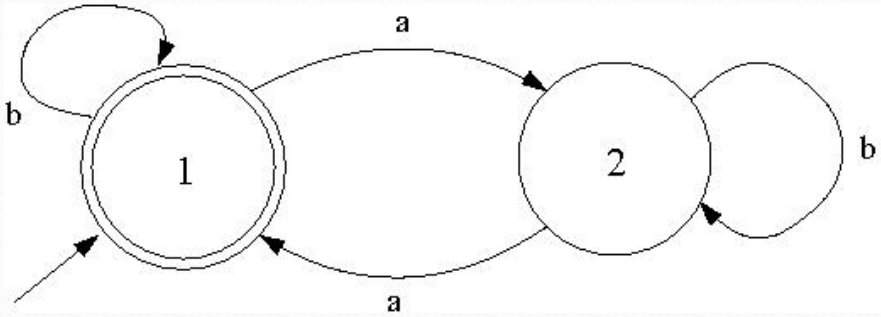
- Often have an Idle state where the process is not active
- All states need some exit (no deadlock, even in error conditions)
- Use multiple state charts to keep the design simple
- Do NOT need to have a state chart as sub state of other state chart
  - System can be described by multiple state machines running concurrently


Automatic
Braking system
WheelStationary [under threshold] /
Too Fast
LockTimeout / ,ApplyBrake
WheelStationary [above threshold] / ReleaseBrake and set timer
SpeedAtorBelowTarget / ReleaseBrake
WheelLock
Speed10%AboveTarget / ApplyBrake
SpeedOK


Dash control
EngineRunning
IgnitionKeyInserted
EngineStarted
BrakePressed / SetTimer, TargetSpeed=TargetSpeed*0.99/BrakePressure
BrakeReleased
Idle
IgnitionOn
BrakePressed
Timeout / TargetSpeed=TargetSpeed*0.99/BrakePressure

# Finite State Machines

- Finite State Machines (FSM), also known as Finite State Automata (FSA) are models of the behaviours of a system or a complex object, with a limited number of defined conditions or modes, where mode transitions change with circumstance.

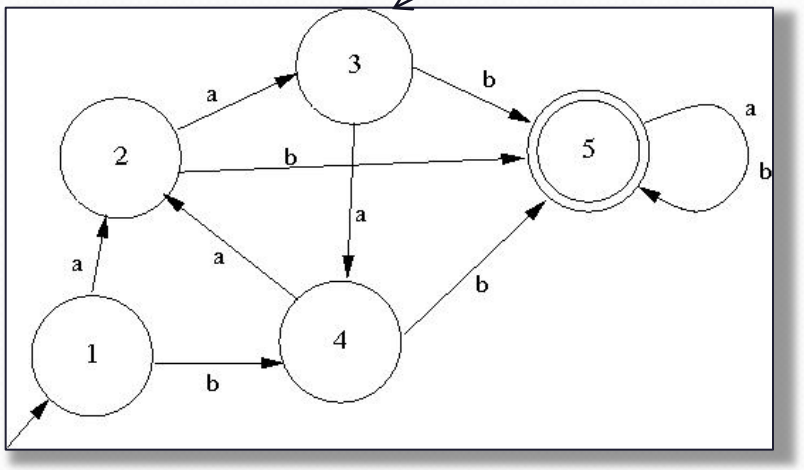



**Question**: What language does this FSA recognise?

# Finite State Machines - Definition

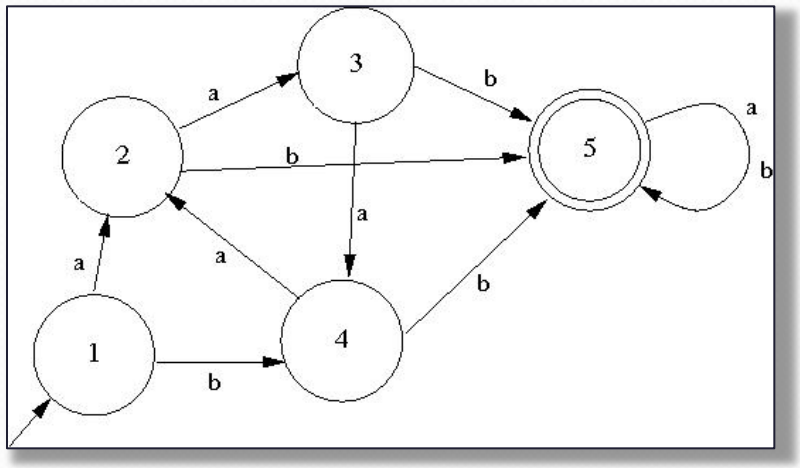
- A *model of computation* consisting of
  - a set of *states*,
  - a *start (initial) state*,
  - an input *alphabet*, and
  - a *transition function* that maps input symbols and current states to a *next state*
- *You may recall finite state machines (or automata) from COMP209.*

What language is recognised by this FSA?



# Finite State Machines - Definition

- Computation begins in the start state with an input string. It changes to new states depending on the transition function.
  - ***states*** *define behaviour and may produce actions*
  - ***state transitions*** *are movement from one state to another*
  - ***rules or conditions*** *must be met to allow a state transition*
  - ***input events*** *are either externally or internally generated, which may possibly trigger rules and lead to state transitions.*



# Lecture Key Points

- A **model** is an abstract system view. Complementary types of model provide different system information.
- Context models show the position of a system in its environment with other systems and processes.
- Data flow models may be used to model the data processing in a system.
- State machine models model the system's behaviour in response to internal or external events