Viewpoints and Models

- Any notation we use, whether diagrammatic or mathematical/textual, has to try to balance the following (largely conflicting objectives)
  - To provide a description that is sufficiently abstract for the particular task, so that unnecessary detail is omitted
  - To provide enough detail that it is possible to reason about the model, in order to perform the tasks at hand
- The design process generally involves considerable degrees of 'trade-off' between these needs

1 Hierarchy

- Having forms of notation that can be used in a hierarchical manner can help reduce cognitive load
- This permits a description of an element at one level to be expanded into one that employs the same form, but a set of less abstract elements, at a lower level

2 Modelling and Architecture

Architectural ideas present two problems

- It's quite hard to model the forms of different architectural designs themselves, although attempts have been made
- Many modelling notations, particularly those concerned with modelling the structure of a system, do implicitly assume the use of some specific form of architecture. This is particularly true for object-oriented modelling (class diagrams) and call-and-return too
- This is less true when modelling behaviour or state, where our models rarely assume the use of particular types of software elements

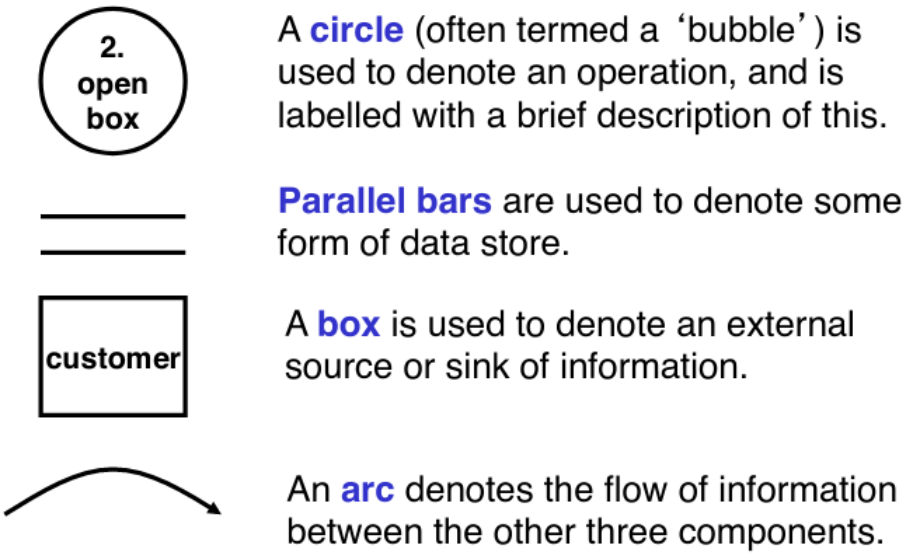
3 The functional viewpoint

- Describes the actions and operations that are to be performed by a system
- The way that this is described may well depend on the architectural style(s) being used
  - In a Dataflow style, would expect to model function in terms of operations and information flow
  - In a Call-and-Return style might expect to be concerned about the sequencing of actions
- In practice, this is one of the hardest viewpoints to capture, partly because of the influence of the architectural style, and also because function can be hard to model with a diagrammatic form

3.1 The Data Flow Diagram (DFD)

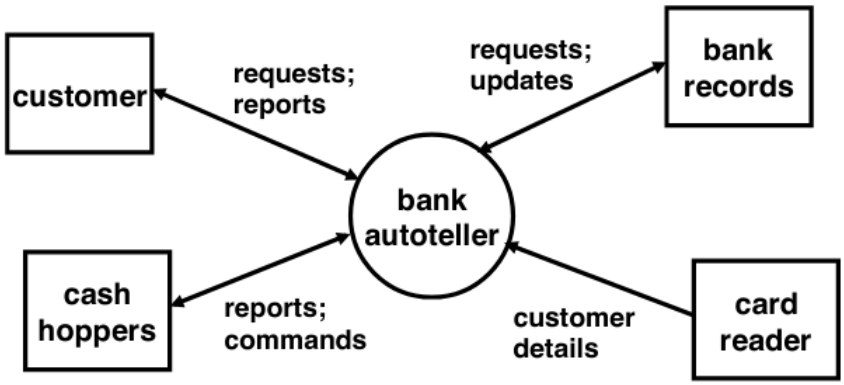
- Essentially a problem-oriented (black box) form used extensively for modelling application functionality
- Traditionally used in the process of the Structured Systems Analysis/Structured Design family of plan-driven design methods, which progress from a Data Flow 'black box' analysis model to a Call-and-Return 'white-box' implementation form

3.1.1 The DFD Symbols



3.1.2 The context diagram

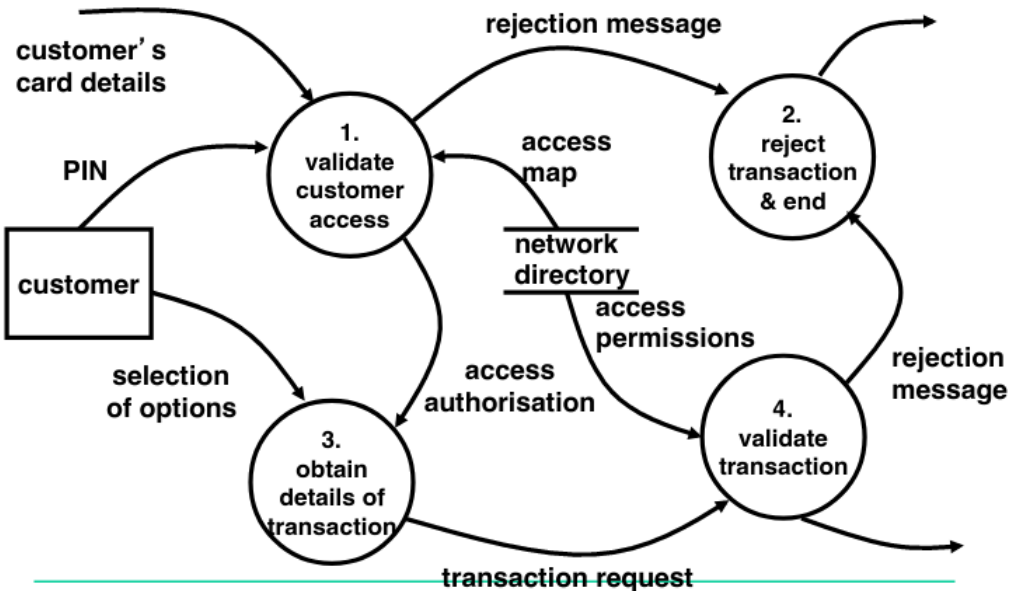
- This is the very top level of a DFD, consisting of a single bubble (to represent the system) and a set of sinks and sources



3.1.3 Hierarchy in DFDs

- The Context Diagram is then expanded into a top level DFD that describes the overall system operation
- DFDs are hierarchical in form, in that bubbles can be expanded by using a further diagram that shows the details of an operation as a DFD
- Convention is to use a numbering system with 1 at the top 1.1... at the next level and 1.1.1... at the next

3.2 A top-level DFD



3.3 What not to include

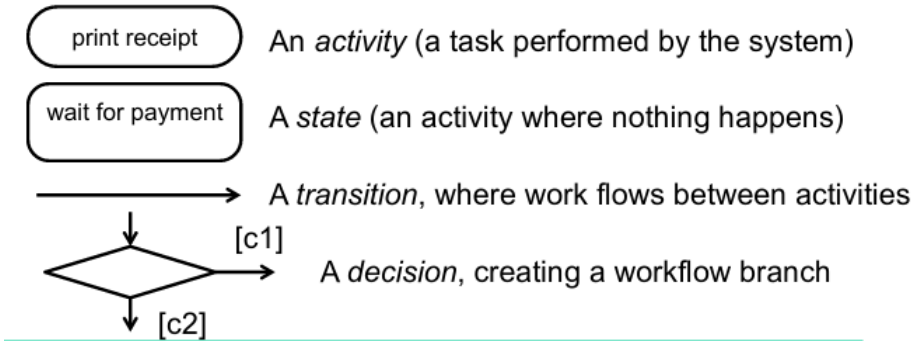
- A DFD does not include details of the control flow that may be involved in operations
- Also, we try to limit the descriptions to be brief

3.3.1 Usefulness

- Regardless of the form of 'solution' we are seeking DFDs can still be useful for modelling some of the aspects of how an application is to work
- In particular, they have the benefit that they can quite easily be explained to customers, and used to help ensure that there is a common understanding of what an application is to do
- Can be useful for identifying user stories, which we often employ during agile development

3.4 The activity diagram

Activity diagrams are often used to model the way that a business operates. Like DFDs, they focus on describing how information flows through a system and gets transformed, but do so by modelling events. Some key notational elements are:



4 The behavioural viewpoint

- Used to capture casual issues through the use of the notion of a finite state machine
- Models for this viewpoint tend to be abstractions that are concerned with states and events and with identifying the transitions that can occur between the different states

4.1 State Transition Diagram (STD)

Can be used fo ”finite state machine” modelling of

- ”system” behaviour at the top level (black box)
- the detailed behaviour of design entities

Various notations - this has four main elements

- Labelled box to denote state
- Arc to denote a transition
- Text above line for the transition condition
- Text below line for transition action

Does not readily scale up for large systems, as the lack of any hierarchical element means that there is no means of expressing groupings of states

4.2 State Transition Table (STT)

Particularly useful when:

- Checking for a completeness of a design model
- Developing the state model, as it may be useful to tabulate the information first, and then draw the diagram

Lacks the ready visualisation of an STD, but handles scale much better, although still non-hierarchical

Provokes thinking about the ”non-events” such as when a user does something unexpected