The Yourdon (Ward-Mellor) Structured Method

Why study YSM?

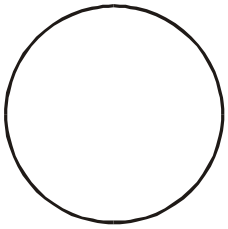
- It has been extended for embedded systems by Ward & Mellor
- It is a popular and well understood method
- It is applicable to both large and small projects
- It is well supported by CASE tools, e.g. Select-Yourdon and EasyCase

Does it have drawbacks?

It is most applicable to the software design stage - its interfaces with the requirements and implementation phases are rather loose

Basic elements of the notation

Terminator



Data transformation process

Data store

Discrete data flow

The terminator represents an item in the system's environment; it acts as a data source or sink.

Data transformation is an input/output process (note that it can report the occurrence of an event, i.e. initiate a control flow).

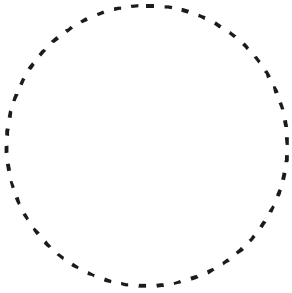
A data store is an abstraction on a file; it acts as a repository for data that is subject to storage delay. Values are modified at discrete points in time and remembered. Readout is not ordered and is non-destructive.

The discrete data flow is an abstraction on a transaction or other data-aggregate sent or received by the system.

Extensions of the method for real-time

For real-time systems, it is necessary to introduce additional elements to handle:

- time continuous data
- event data (logic signals)



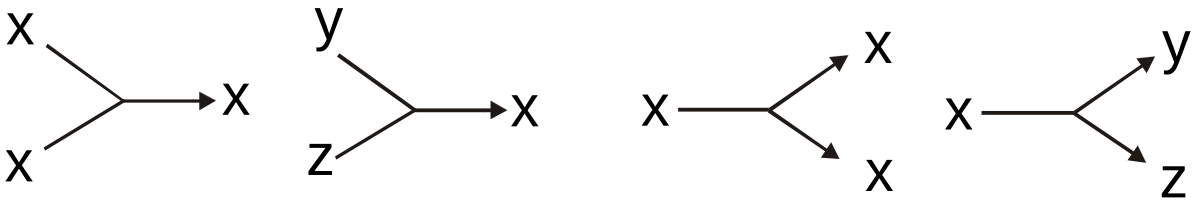
Control transformation process

Control store

Continuous flow

Control flow (signal)

Prompt : Activate/ Enable/ Disable/ Trigger

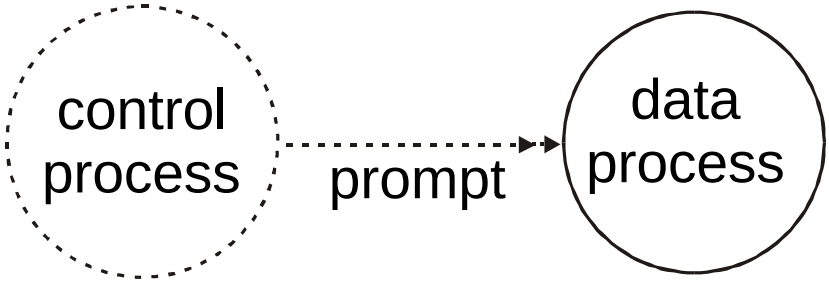


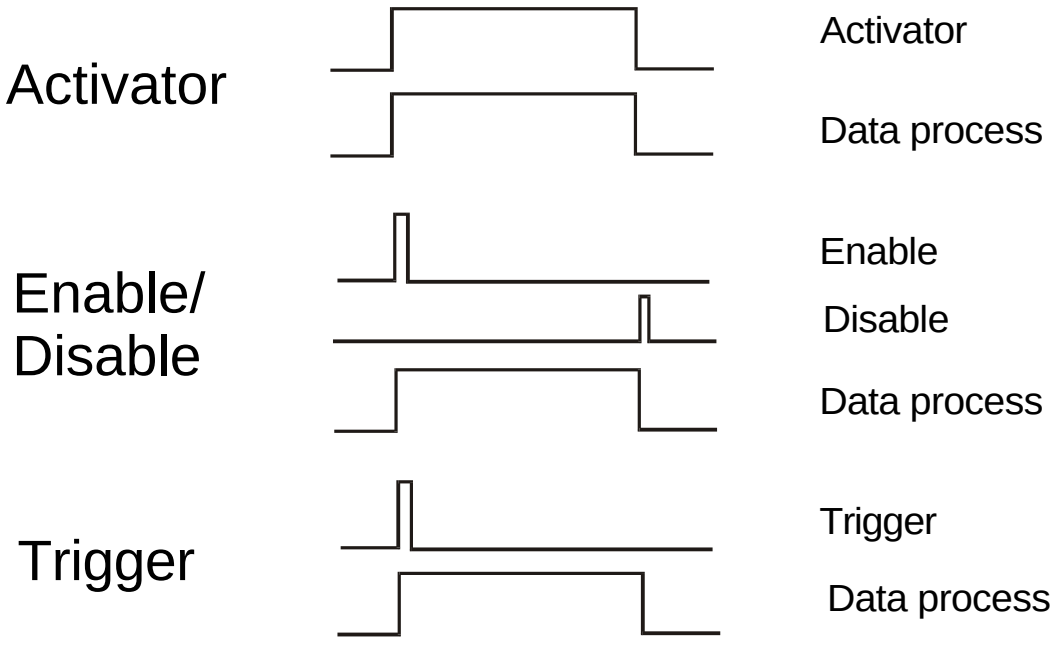
Group nodes

The control transformation controls the behaviour of data transformations by activating or de-activating them; it is an abstraction on some portion of the system's logic.

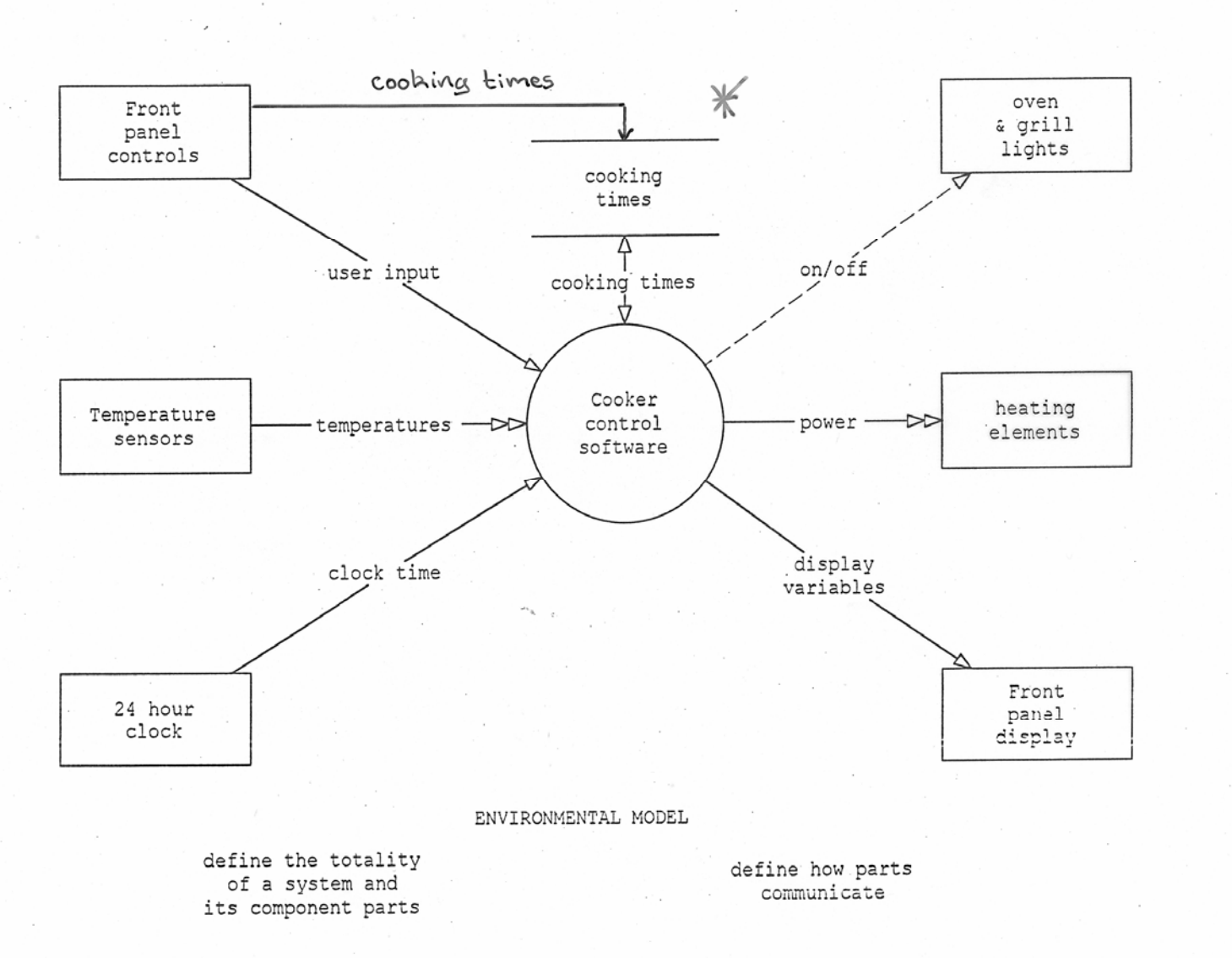
The control store represents a buffer. It is an abstraction on a stack (e.g. LIFO or FIFO). Readout is not ordered but it is destructive. The control store has a capacity.

Prompts are initiated by control processes and activate data transformations. There are various types :



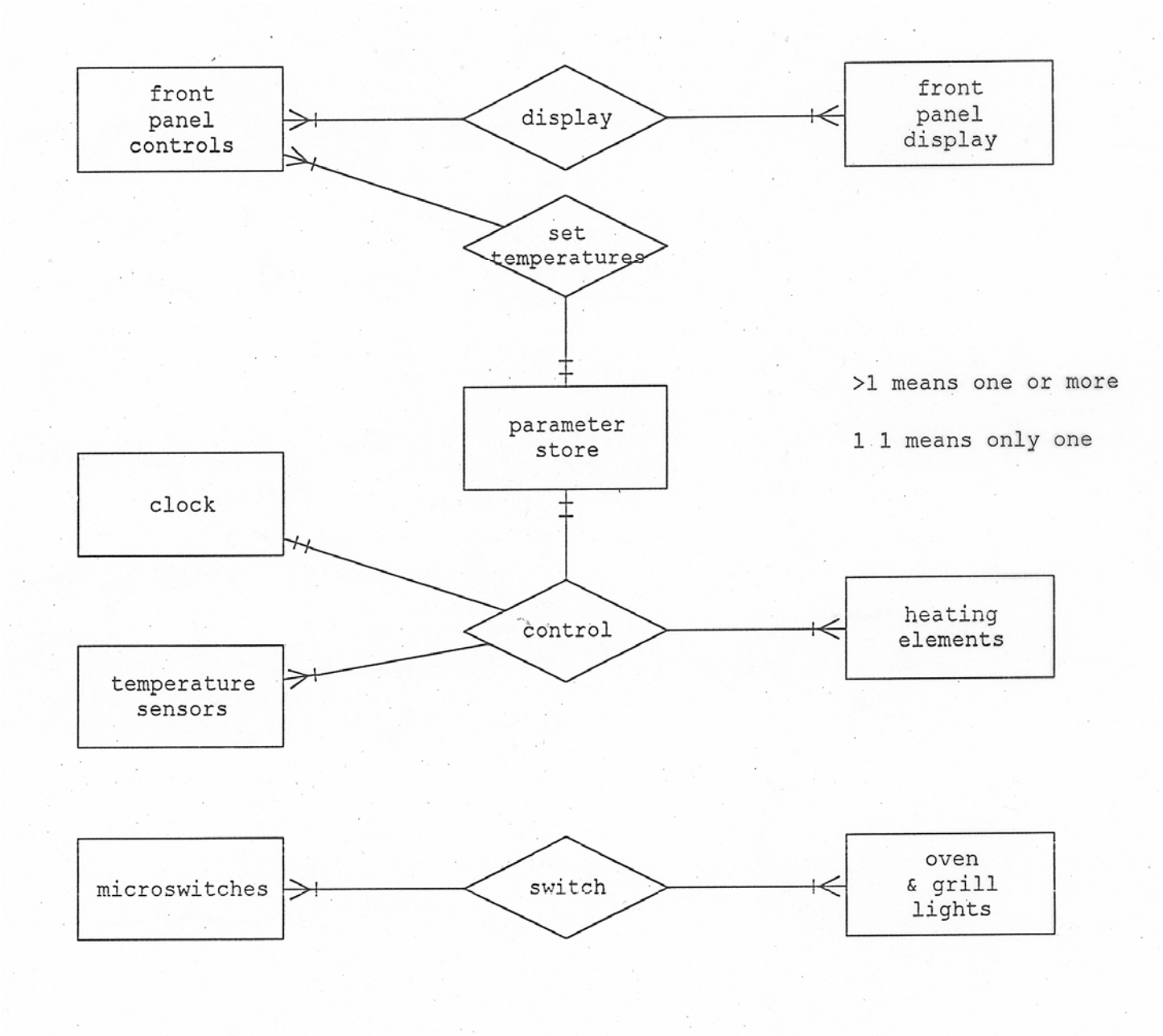


A more complex Environmental Model.



Context Diagram – Cooker Control Software

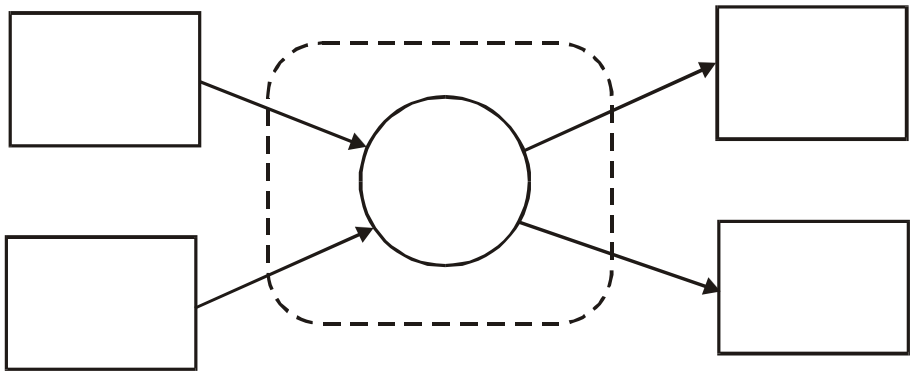
* This is known as an ‘access’ flow and indicates that the system uses stored data that is shared between it and its environment. Any store shown on the context diagram must be accessed by the system and at least one terminator.



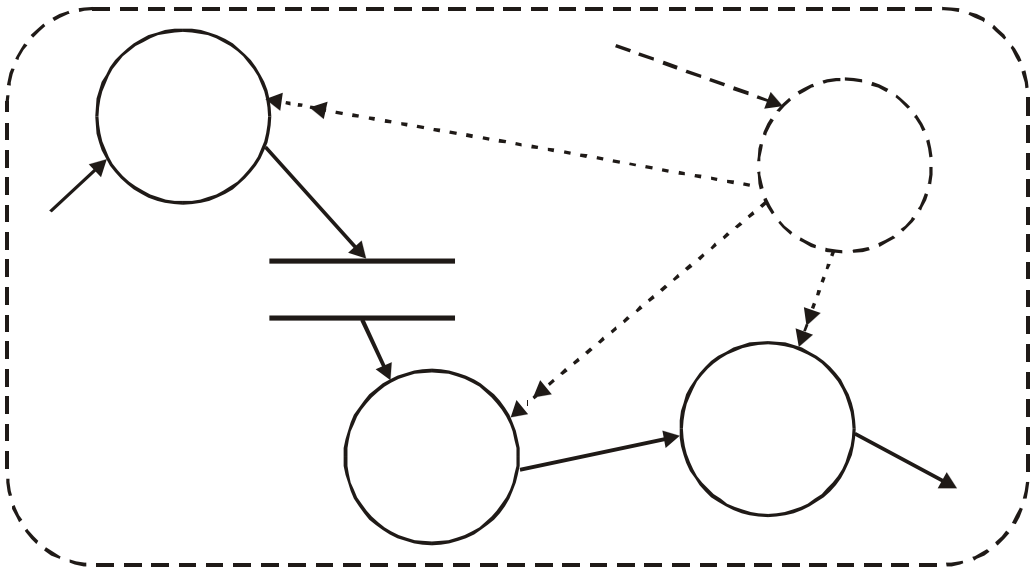
Cooker Control

Method of use

The method consists of building a model of the system in a hierarchical manner; this is known as levelling. The highest level is known as the context diagram. Typically, it consists of a single data transformation, some data/control flows and terminators.

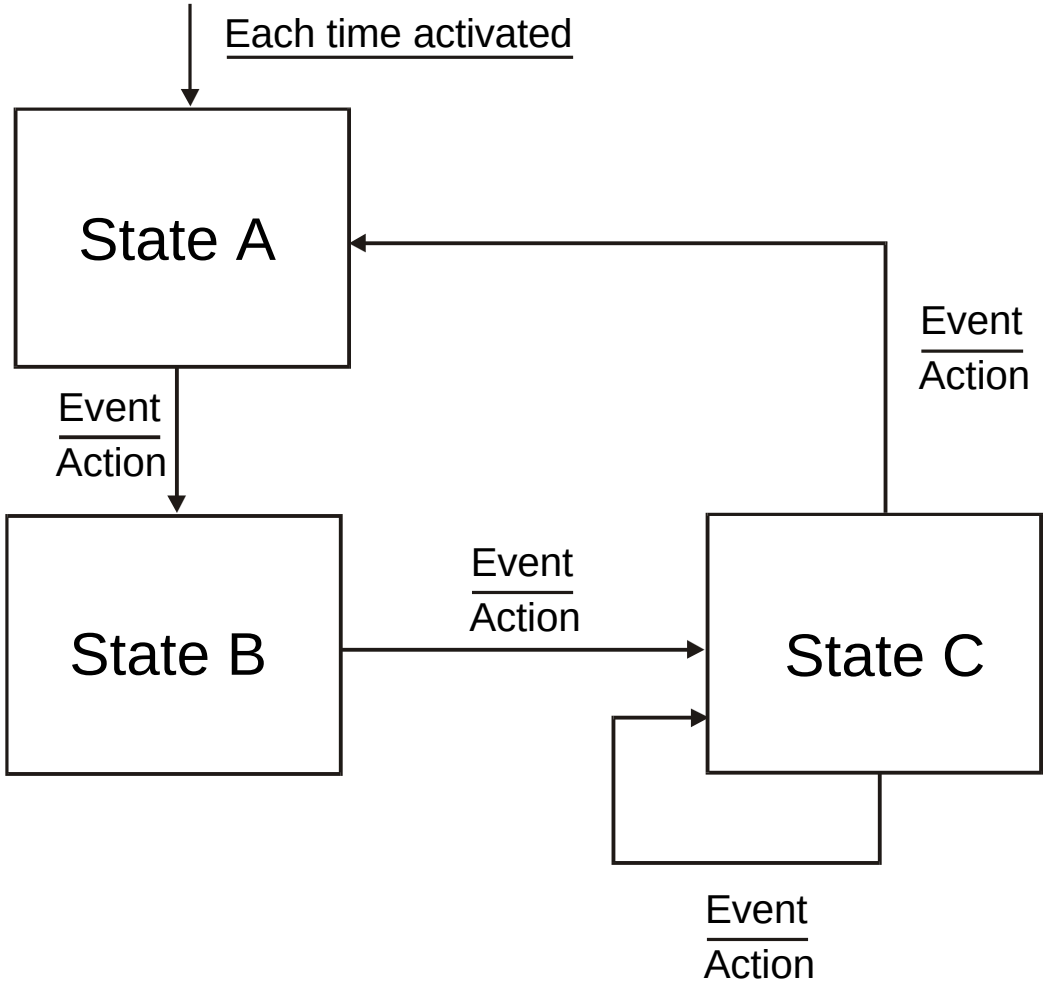


The data transformation node conceals a ‘child’ data flow diagram (DFD).



The control process

Control specifications are given in terms of the familiar state transition diagram



Transitions between states occur as a result of input events

Action routines produce output event flows and prompts (Mealy FSM)

Process Specifications

The procedure of decomposing the system continues until a level is found when the operations to be performed within a transformation need not be subdivided further.

At this point, description of the individual processes is done by means of a PSPEC.

Process specifications usually consist of structured English, tables, mathematical formulae, graphs etc.

```
@IN = list of input data items
@OUT = list of output data items

@PSPEC
FOR (all input data items)
  calculate (output data items)
  write (output data items)

@COMMENT
```

Structured English has 4 basic constructs :

Concurrency

```
Do function A
Do function B
Do function C
```

Sequence

```
Do function A
then
Do function B
then....
```

Decision

```
IF condition
  DO function A
ELSE
  DO function B
```

Iteration

```
WHILE
  condition
DO
  function A
```

The Data Dictionary

Flows leaving and arriving at the system are usually non-primitives, representing groups of data.

These groups decompose into smaller and smaller groups as they proceed down the hierarchical levels until they reduce to primitive components.

The dictionary specifies the components and structure of each group using a special notation - the Backus-Naur form (BNF).

Example : The pilot's display of a flight management system can be one of several types, e.g. display initialising, route display, progress display and may, at any time, carry superimposed alert/advisory messages.

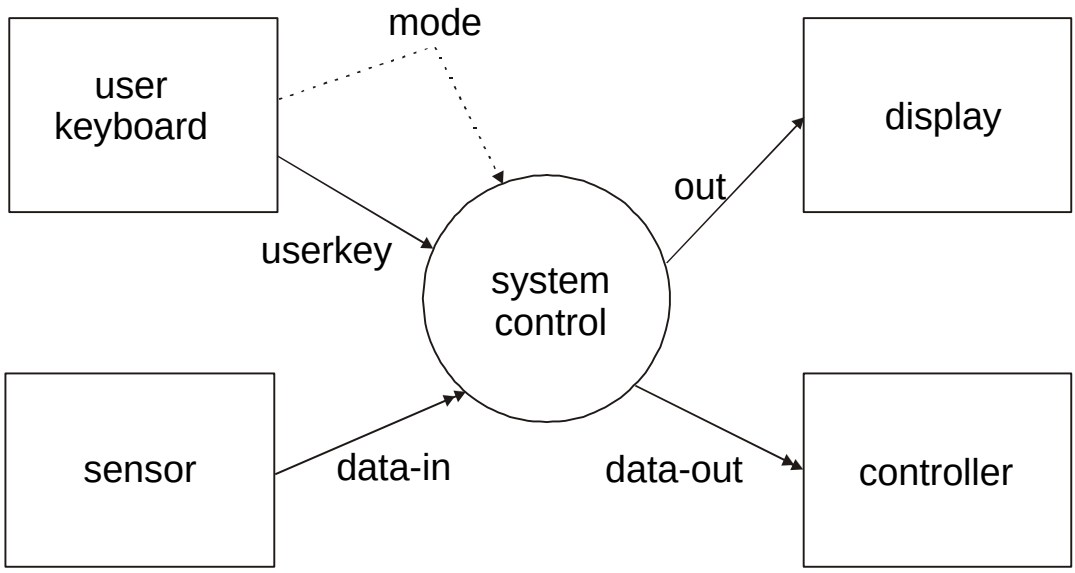
VisualDisplay = [InitialDisplay | RouteDisplay| ProgressDisplay] + (Alert/AdvisoryMessage)

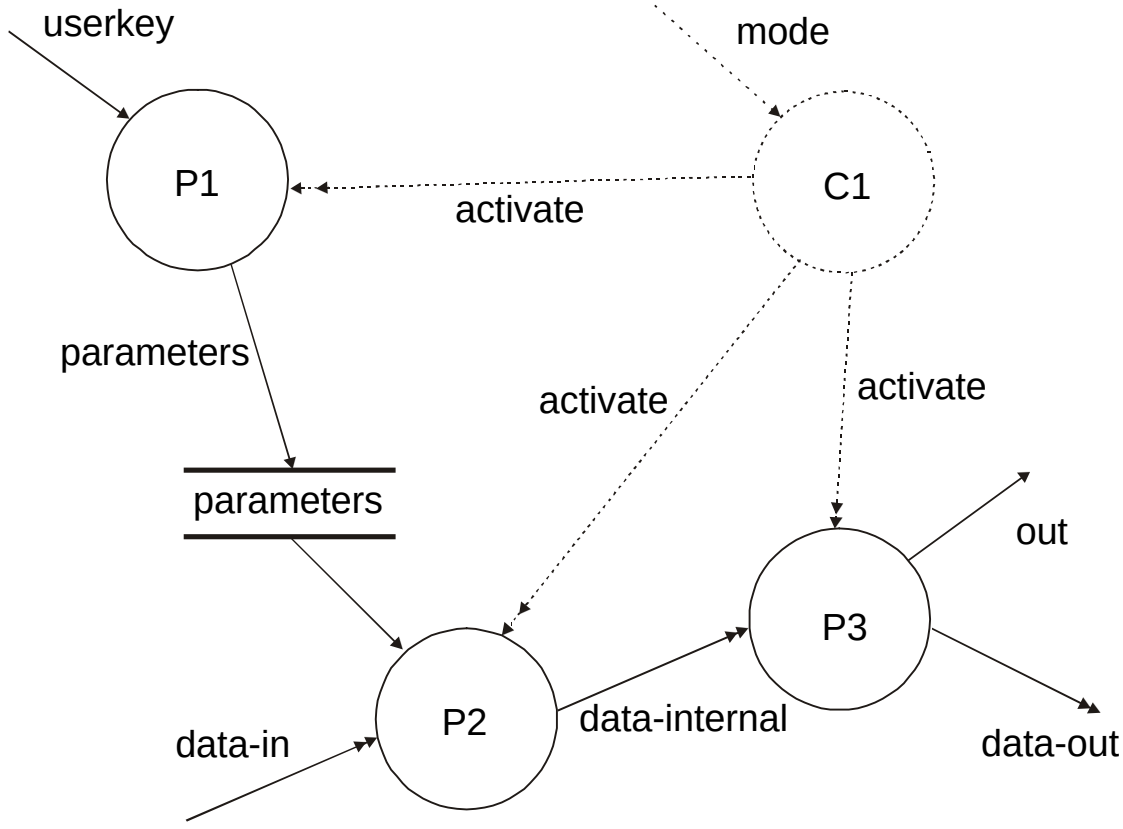
where : Alert/AdvisoryMessage = [WingFallenOff | EngineOnFire | YouAreLost]

Data Dictionary Symbols (BNF)

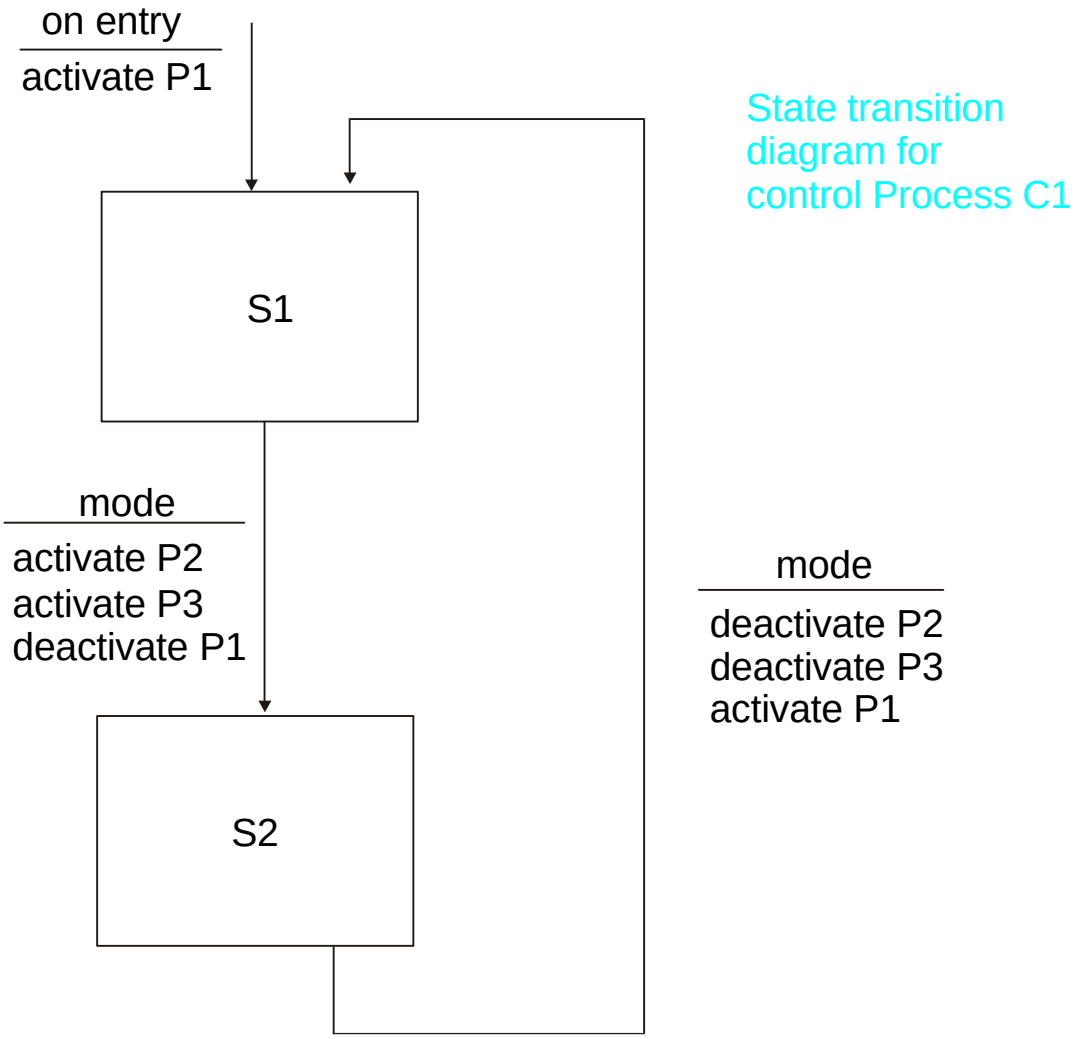
Symbol	Meaning	Description
=	composed of	The flow named on the left is composed of the flows named on the right.
+	together with	Collects members into a group but does not imply ordering.
{ }	iterations of	The expression within the brackets may occur any number of times in a given instance of the flow. The brackets may be indexed. M{ }N indicates any number of iterations from M to N, e.g. { }2 is 0, 1 or 2 iterations 2{ } is 2 or more iterations 2{ }2 is exactly 2 iterations
[|]	select one of	A given instance of the flow will contain exactly one of the options within the brackets.
( )	optional	The expression within parantheses may optionally appear in a given instance of the flow.
“ “	literal	The symbols enclosed within quotes literally constitute the data flow.
\ \	comment	Textual information.

System Decomposition





N.B. This example is not a complete specification



State transition diagram for control Process C1

Process specification for data transformation P1 :

1.2 - interpret keyboard buffer, PSPEC02.DAT

@IN = userkey
@OUT = parameters

@PSPEC interpret keyboard buffer

SEQ
  decode userkey to decoded userkey
  convert decoded userkey to 16 bit REAL type parameters

@

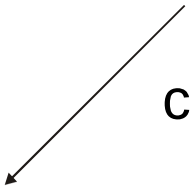
The Code Organisation Model

This uses the Constantine program structure chart with the following notation:

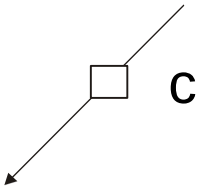
program module

sub-system : represents some pre-defined function, perhaps within the operating system or language (sub-systems cannot have children)

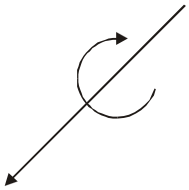
library function : pre-defined function that may be used across many modules



call



conditional call



iterated call

data couple

control couple

A Constantine diagram shows the architecture or calling structure of modules within a project.