

DIVISION OF COMPUTER SCIENCE

**Modelling Real Time Systems Functional
Requirements using Existing Data Flow Methods**

D. A. Fensome

Technical Report No.160

July 1993

Modelling Real Time Systems Functional Requirements using Existing Data Flow Methods

D.A.Fensome

Summary

This report summarises the data flow models from existing real time systems development methodologies, and describes some of the problems with using them to define real-time system functional requirements. The motivation for this report is to establish a suitable model for prototyping real-time systems.

A vending machine case study is used to model various system properties, and Z was used to describe the functional properties. The report shows that there is a case for a new data flow model to define system requirements.

Contents

1. Introduction.....	1
2. Overview of Structured Analysis and JSD requirement models	1
3. Model semantics.....	2
3.1 Processes and data flow.....	2
3.2 Issues of Determinism.....	3
3.3 Feedback and Deadlock	4
4. Formal Process description in Z.....	4
4.1 Structured Analysis.....	5
4.2 JSD.....	7
5. Formal event sequence definition.....	8
6. Time.....	9
7. Interface Refinement	9
8. Conclusion.....	10
Appendix A	13
A.1 Outline requirement.....	13
A.2 CPVM data flow diagrams	13

1. Introduction

To describe main features of real time application domain ie event processing, data processing, process oriented without large static data structures

To give an historical perspective on the development methods based on Structured Analysis and JSD

This report is concerned with requirement modelling of real time engineering systems and is motivated by research in prototyping methods.

Real time systems are event driven, with some data processing, and often concerned with the control and monitoring of external equipment. They are also know as 'embedded systems' and 'reactive' systems [Harel 92].

Data flow modelling has been used over many years to model computer systems requirements. It is one class of model that has considerable attraction from a visual viewpoint, and is in wide use in the commercial world.

Some current systems development methods, particularly those based on structured analysis, are a subset of the generic data flow model. These methods, especially those with an established reputation for large real-time systems, are an obvious starting point for requirement modelling.

The dominant methods, particularly in the USA [Wood 89], seem to be those based on Structured Analysis [Yourdon 89] with roots in the early 70's, but there are some using JSD. In Europe JSD [Cameron 89] seems to be more popular than in the USA. JSD evolved from JSP in the 70's culminating in the JSD method, first published in [Jackson 83]

The structured analysis variants developed for real-time systems are the Ward-Mellor method developed in the early 1980s [Ward & Mellor 1985] and [Ward 86], and the Hatley and Pirbhai method [Hatley 88]. Bowles [Bowles 90] describes the evolution of structured development methods.

2. Overview of Structured Analysis and JSD requirement models

2.1 To overview the development method and where the requirement model appears in the method

2.2 To describe the important features of both models ie process strategy, event sequence definition, process communication, data definition

The Ward and Mellor method defines an Essential model, which consists of a Behavioural Model and an Environmental model, and an Implementation model. The Behavioural Model consists of a 'transformation schema' based on data flow and an associated control model which activates and deactivates sets of processes.

The Hatley and Pirbhai method also produces a set of models. At the top level a requirement model is produced by progressive hierarchical refinement of a data flow model, coupled with the development of a control model similar to the Ward Mellor method.

In both these requirement models the data communication relies on synchronous unbuffered data flow (bi-directional or unidirectional) of a specific type, which can be merged, split and recombined. Global data stores are also allowed. Data types are defined using standard BNF type data dictionary notation.

JSD is also a data flow modelling technique but based on process composition, rather than hierarchical decomposition. It is centred around a set of 'model' processes, and does not have an associated control model. Processes are defined which model a set of real world entities, which then communicate with the real world in the computer system. In a sense these JSD model processes are a simulation of the real world.

However data communication in JSD consists of an asynchronous uni-directional buffered data stream connection (which can be merged but not split), or a 'state vector' connection which allows read only inspection of a process state. There are no global data stores, since all data is held in the state vectors of specific processes .

An additional feature of JSD, which is not apparent in the H & P method, is that the modelling phase also models the sequence of events which are applied to the system. One output of this phase is a structure chart (a regular expression substitute) which defines legal event sequences.

3. Model semantics

3.1 To describe the areas where semantics are unclear ie data merge/split, process execution, race conditions, feedback

3.2 To describe the areas of non determinism that exist in the models even after semantics have been defined (above) eg concurrent event processing

3.1 Processes and data flow

The generic data flow model relies on algorithmic specification of node equations, the evaluation order being determined by the data dependencies. However the operational semantics of the structured analysis models and JSD are different.

In the Ward Mellor method multiple data inputs to a process are merged first with all the composite elements being present, before being applied to a process. The process may have more than one discrete output, but only one of the outputs is produced for each process transformation.

Ward describes the operational semantics of the schema, based on a Petri-Net like execution method with the placement of tokens on data flow and processes as the processes execute. The processes execute in zero time for the behavioural model. The tokens describe the flow of data through the system, and the rules for process

execution are described.

The H & P method allows multiple data flow inputs to processes, which are triggered whenever data sufficient for process execution is available. No additional execution rules are given, although the token mechanism used by Ward could also be used for 'symbolic execution' of the H & P data flow model.

In fact there appears to be no difference between the two Structured Analysis requirement models as far as data flow semantics are concerned, although this is not at all obvious to start with. The confusion is because of the different sets of models, and because symbolic execution using tokens clouds the issue particularly when applied to implementation models (where the processes have a finite execution time). In fact the overriding consideration of the requirement models is that the processes execute in zero time ie every thing happens simultaneously within the model.

The semantics for JSD model processes are hard to establish. In Jackson's original book [Jackson 1983] the JSD model processes are described as truly concurrent, modelling the real world. This implies that the model processes have a finite execution time which reflects their real world processes. During the implementation phase the processes are converted into subroutines or concurrent tasks, depending on the target system and the timing requirements. Thus it seems that the JSD process network specification allows true concurrency to be modelled at the specification level.

3.2 Issues of Determinism

It is important to recognise where non-determinism appears in the requirement models. We may or may not require a deterministic result when an event is applied to a requirement model, as much as we may or may not wish to specify how a particular function should be implemented. Thus we may wish to constructively use non determinism. However if the model has undesirable or hidden non determinism then this should be identified.

Both the structured analysis methods use models where the processes execute in zero time. The implication of this is that effects of simultaneous external events are never interleaved within the model during model execution. If two events are applied simultaneously one of them will be non-deterministically chosen and processed first atomically. If any other ordering is required then this should be explicitly stated, or modelled somehow. This argument must also apply to simultaneous application of data and event (control) flow.

Another view of simultaneous external event application is that in practice it never occurs (or is outside the model bounds), because one event will always occur first by some very small time interval (at least by the model time scale).

However during model execution there are some possible areas of indeterminacy involving internal data flow, even though zero process execution time is assumed. This concerns simultaneous writes, and reads and writes, to a data store and

possible 'race' conditions. If all processes execute simultaneously then does a read from a data store access the same data written at that instant by another process? Or does the data store contain some 'magic' which separates written and read data at the execution instant, such that reads access previously written data? In fact the latter view must hold, since in the former case a data store would not behave as a store, but as a direct data flow connection. With this segregated read/write definition of a data store, the data store operations are deterministic.

In JSD the processes execute in real time, and have buffered data stream connections. Thus external events may not be processed atomically, and it is much more difficult to reason about the requirement model without considering time and consecutive external events. In addition if merged data streams are used, either component can trigger a process when a data record arrives; however when data records arrive simultaneously the process chooses a record non-deterministically. In the JSD method this non-determinism is resolved in the implementation phase, but the choice is not apparent in the network data flow model.

3.3 Feedback and Deadlock

Feedback is an integral part of the JSD method, since the real world 'model' processes may interact with the system under design with interactive processes. Therefore there is a possibility of system instability and also deadlock. With Structured Analysis methods there is no such overt use of feedback, the whole emphasis being to use a feedforward data flow network.

Again the JSD solution seems to be to resolve feedback and deadlock problems in the requirement model during the implementation phase. This does not seem a good solution, particularly if the requirement model is being used to build a prototype, and also if the problems are not apparent.

4. Formal Process description in Z

To show it is possible to model processing using Z, for both H & P and JSD and to outline one strategy for doing this.

Since there may be a need to represent processes in a non-deterministic way (either for internal or external non-determinism), and also to avoid specifying processes in a procedural manner, it is desirable to use mathematical notation to represent aspects of the models. For example the vending machine case study in Appendix A has state and process specifications which could be formalised.

Whilst the system state and types can easily be represented using conventional function and set notation, the process specifications could require multiple outputs and the possibility of read and writes to the system state. A more formalised language is therefore required, and Z has been chosen solely on the grounds of familiarity (VDM would have been another appropriate choice).

However, since Structured Analysis and JSD use different principles, the application of formal techniques is considered separately for each method.

4.1 Structured Analysis

Considering first the physical CPVM machine state from DFD0 and CSPEC 0 in Appendix A

```
CPVM
total : PRODUCT --> GOODSHOPPER
price : PRODUCT --> COST          /*these model 'product_data'

coins : COINHOPPER              /*models the machine coin register

control_machine : STATE          /* models CSPEC 0
```

The types (apart from STATE) are the structured analysis types taken from the types dictionary. STATE is a convenient type representing the CPVM control state and declared as

STATE = {s0, s1, s2, s3}

corresponding to the finite state machine states in CSPEC 0.

Some operations defined by the data flow processes are -

```
GetProductPrice /* P1
ΔCPVM
entered_price?: CATALOGUE

price' = price ⊕ {cost&product entered_price?}
total' = total ⊕ {total&product entered_price?}
current_payment' = current_payment
coins' = coins
```

where

cost&product : CATALOGUE --> PRODUCT X COST

cost&product ((a,b,c) : CATALOGUE) (x,y) : PRODUCT X COST

pre_cost&product = true

post_cost&product = (x = a) ^ (y = b)

and similarly for total&product

and

calculate_change_____/* P4.1

```
ΔCPVM
the_product? : PRODUCT
overpayment! : MONEY
```

```
overpayment! = current_payment - price (the_product?)
```

For both these operations the input and output data flow has a clear correspondence to the input and output data in the operation schema. Similar approaches can be found in [Randell 91].

To show some non-determinism, the system could be allowed some choice as to the exact change given to a customer. For example

Convert_to_coins_____/* P 4.2

```
ΔCPVM
over_payment? : MONEY
change! : COINHOPPER
```

```
∃ change! • ( (moneyvalue change! = overpayment?)
              OR change! = min (overpayment?, coins))
              AND coinsavailable (change!, coins)
```

where

```
moneyvalue : COINHOPPER --> MONEY
```

which returns the monetary value of the coinhopper combination

and

```
min : MONEY X COINHOPPER --> COINHOPPER
min : ( m : MONEY, c : COINHOPPER) ch : COINHOPPER
```

which minimises (m - moneyvalue c) with combination of change ch.

and

```
coinsavailable : COINHOPPER X COINHOPPER --> BOOLEAN
coinsavailable ( c, r : COINHOPPER) b : BOOLEAN
```

which returns true if all the coins in c are available from r.

The control machine CSPEC 0 is quite easily defined assuming one input of type EVENT defined as

```
EVENT = {ci, mp, rc, op, pg}
```

where ci is coin_inserted, mp is minimum_payment, rc is return_coins, op is overpayment and pg is product_given. Also there are two process activators pat3! and pat4! which need to be set true for those processes, and any child processes, to run.

```

_____CSPEC 0_____/* the control machine
|
|  ΔCPVM
|  events?: EVENT
|  pat3!, pat4! : BOOLEAN
|
|-----|
|
|  events? = ci <=> control_machine' = s1
|  ((events? = rc) ^ (control_machine = s1)) => control_machine' = s0
|  ((events? = mp) ^ (control_machine = s1))
|  <=> (pat3! = true)^(control_machine' = s2)
|  etc
|
|-----|

```

corresponding to the finite state machine diagram for CSPEC 0.

4.2 JSD

In JSD the overall state is held within the system processes. The state of any one process is required externally to that process if a state vector connection is present. For example the Stocker process in the JSD CPVM network in Appendix A has a state vector connection to the Customer process. The stocker state can be defined as

```

_____Stocker_State_____
|
|  Stock : PRODUCT --> COST X GOODSHOPPER
|
|-----|

```

and one of the operations on that state occurs when the DS data stream sends data to the Stocker process. For example the decrement stock operation could be defined as

Decrement_Stock

Δ Stocker_State
ds? : PRODUCT

Stocklevel Stock ds? > 0
Stocklevel Stock ds? = Stocklevel Stock ds? - 1

where Stocklevel returns the hopper component of the Stock tuple, as follows

Stocklevel : COST X GOODSHOPPER \rightarrow GOODSHOPPER
Stocklevel (a,b) : COST X GOODSHOPPER h : GOODSHOPPER
pre Stocklevel = true
post Stocklevel = h = b

For the customer operations that require the stocker state, schema inclusion can be used in the normal way. For example

Validate_Selection

Ξ Stocker_State

Pa! : BOOLEAN /* product available light */
S? : PRODUCT /* customer selection */

Pa! = Stocklevel Stock S? > 0

Thus it is possible to use Z to represent the state and operations that Structured Analysis and JSD build on, and the identification of the system state and operations can be easily be found from the data flow diagrams.

5. Formal event sequence definition

To describe how the methods approach this

The safety properties of real time systems define what the system is allowed to do, see [Lamport 88], and this can be represented as an event sequence. How to specify this sequence, and to formalise this, is an important part of modelling real time systems requirements.

Neither of the structured analysis methods model the external event sequences. Both methods specify an internal control machine, and Ward-Mellor lists the

external events, but that is all. This must be a serious weakness of Structured Analysis methods.

In JSD the method starts by modelling event actions from external entities, and a representation of these with structure charts (which are acknowledged to be a diagrammatic representation of regular expressions in [Cameron JR 86]). These structure charts and their data are used to define and build the system 'model' processes. Appendix A 2 shows the structure chart and the corresponding data flow network for the CPVM case study. Work by [Sridhar & Hoare 85] has also shown that CSP can be used to formally specify the JSD processes and event sequences, although there is no easy way to model the JSD internal state vector communication method in CSP.

6. Time

To show that timing considerations can be represented by incoming events

Motus describes time concepts in real time systems in [Motus 92], and he describes 3 categories as follows :-

- a) performance bound properties
- b) timewise correctness of data
- c) time correctness of interprocess communication

together with appropriate formal techniques.

From the functional modelling viewpoint a) is not relevant, and c) is only relevant in JSD. The timewise correctness of data is the main concern, and includes any liveness properties of the system. For example that after a 'coin inserted' event the machine state *must* eventually return to s0.

Structured Analysis data flow models have considerable appeal over JSD when considering b) since the processes and communication take place in zero time. In order to reason about timewise correctness all that is required is to add events to the model representing absolute or relative time.

7. Interface Refinement

To show that this is an important feature and how the methods deal with this

Interface refinement may be considered as moving from a logical to a physical data flow model. This is an important step in the development of a real time control system, and the development method should be able to deal with this.

In Structured Analysis methods the data flows from the external terminators will change from logical entities (eg the customer in the CPVM), to physical interface devices eg the coin hopper, product selector etc. Appendix A2 shows such a refinement. The effect of this change is that the data flows may change (particularly the types) on each level of the process hierarchy. For example money inserted in the logical context diagram is of type OBJECT = (slug | 10p | 20),

whereas in the physical system the coin hopper produces data of type PHYSICAL = (DIAMETER X WEIGHT).

In JSD input processes act as a filter between the system inputs and the model processes. The input processes collect data from the real world and filter out erroneous data or actions, and hence can be thought of as physical to logical data converters. For example in the CPVM case study the input processes could reject 'slugs' from the customer and supply only coin data to the model processes. See Appendix A2 for the JSD network. Thus JSD is much more suited to interface refinement.

Similar experience was found by Cameron and Butcher in a paper on the use of JSD on the Spearfish Torpedo System see [Cameron 89]. This paper also states that context filtering was very useful in testing the system, because the model processes only had to be tested with legal sequences.

8. Conclusion

To summarise the semantic and deterministic problems
To summarise how formal notation can be used

This report has shown the development and main features of Structured Analysis and JSD. Both of these methods rely on data flow models, but the communication model and process execution assumptions are significantly different. These differences are important when reasoning about the semantics of requirement models based on these methods.

With Structured Analysis models processes execute in zero time. When simultaneous external events occur then it may be assumed that one of them is chosen non-deterministically and executed atomically. Within the model a segregated read/write data store must also be assumed to remove possible 'race' conditions.

JSD models are much more difficult to reason about because of finite process execution time, buffered communication, and because there is some internal non-determinism in merged data streams. JSD may also have more problems than Structured Analysis methods because of possible feedback and deadlock in the process network.

Another concern is whether formal techniques can be used to model the processes and state of the methods, and also the external event sequences. Z was shown to be adequate for the CPVM case study, and CSP could be used to represent the external event sequence. Timewise correctness of data and liveness properties could be reasoned about by using external timing events.

Finally the JSD method of separating model processes from the real world by context filtering, would seem to have some attractions in developing a physical model from a logical model.

Overall there would appear to be a case for developing a new data flow model based

on -

- zero execution time processes that are composed rather than hierarchically decomposed
- unbuffered process communication
- global or process state storage
- an appropriate mathematical formalism
- context filtering
- easy translation into an executable prototype

Future work to determine the influence of the animation language and paradigm, will resolve some of these options for prototyping functional requirements.

References

- Bowles A J *A note on the Yourdon Structured method* ACM Sigsoft April 1990 pp 27
- Cameron JR 1986 *An Overview of JSD* IEEE Trans SE Vol SE-12 No 2 Feb 86
- Cameron C 1989 *JSP & JSD: The Jackson Approach to Software Development* 2nd Ed, IEEE Computer Society Press 1989
- Harel D 1992 *Biting the Silver Bullet* IEEE Computer Jan 1992
- Hatley, D.J. and Pirbhai, I. A. (1988) *Strategies for Real-Time System Specification* Wiley
- Jackson M I 1983, *Systems Development* Prentice Hall 1983
- Lampert L 1988, *A Simple Approach to Specifying Concurrent Systems* DEC System Research Centre Monogram
- Motus L 1992 *Time Concepts in Real Time Software* IFAC/IFIP Real Time Programming conference Brugge 1992
- Randell G 1991 *Data Flow Diagrams and Z* Z User Workshop Oxford 1990 Springer Verlag/ BCS
- Sridhar K T & Hoare CAR 1985 *JSD expressed in CSP* Oxford University Technical Monograph PRG -51 (also in Cameron 89)
- Ward & Mellor 1985, *Structured Development for Real-Time Systems* Prentice Hall 1985
- Ward P T 1986 *The Transformation Schema: An Extension of the Data Flow Diagram to represent control and timing.* IEEE Trans on Software Engineering Vol SE-12 No 2 Feb 1986
- Wood D P and Wood W G 1989 *A comparative evaluation of four specification methods for real-time systems* SEI technical report CMU/SEI-89-TR-36
- Yourdon, E. (1989). *Modern Structured Analysis* Prentice Hall

Appendix A

Crook Proof Vending Machine Case Study

A.1 Outline requirement

The vending machine is to stock Cola or Beer priced 20p and 30p respectively. Legal coins accepted by the machine are 10p, 20p and 50p; other objects will be rejected and delivered back to the customer. The customer selects the required product, and if this is available, it is delivered; otherwise a 'product not available' light is lit and the machine waits for further selection, or a request for all coins inserted to be returned.

Legal coins inserted into the machine will be routed to the appropriate change hopper if it is not full, or to a coin bin when full. Change will be given automatically, but only if the change register hopper has the appropriate coins and a product has been delivered to the customer. The customer may ask for the inserted coins to be returned at any time after one legal coin has been input, and before receiving a product.

A 'Stocker' may refill the product hopper and reprice any product via a small maintenance panel inside the machine. Access to this is by unlocking the vending machine front panel containing all the coin and product dispensing hardware.

A computer controller is required which will interface into -

inputs

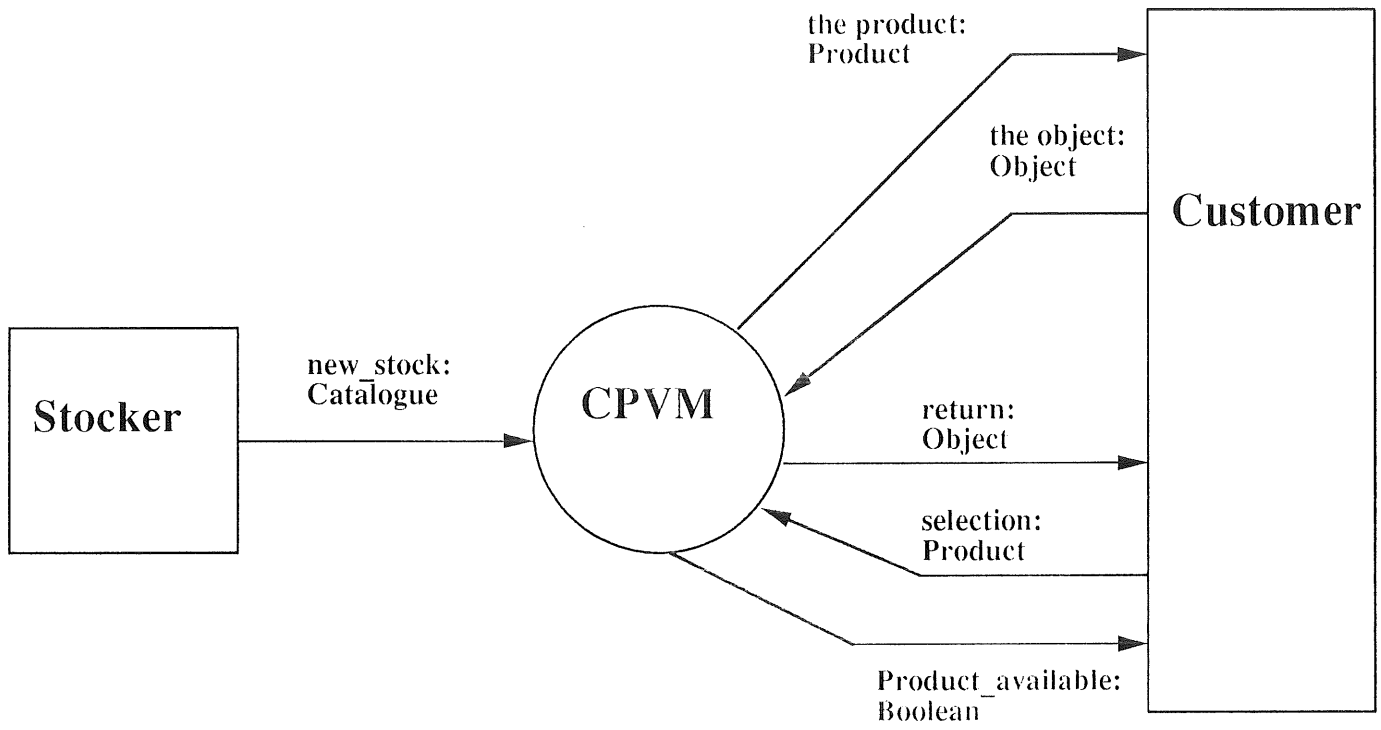
- coin measurement system
- change coin register
- selection panel
- return coins button
- maintenance panel

outputs

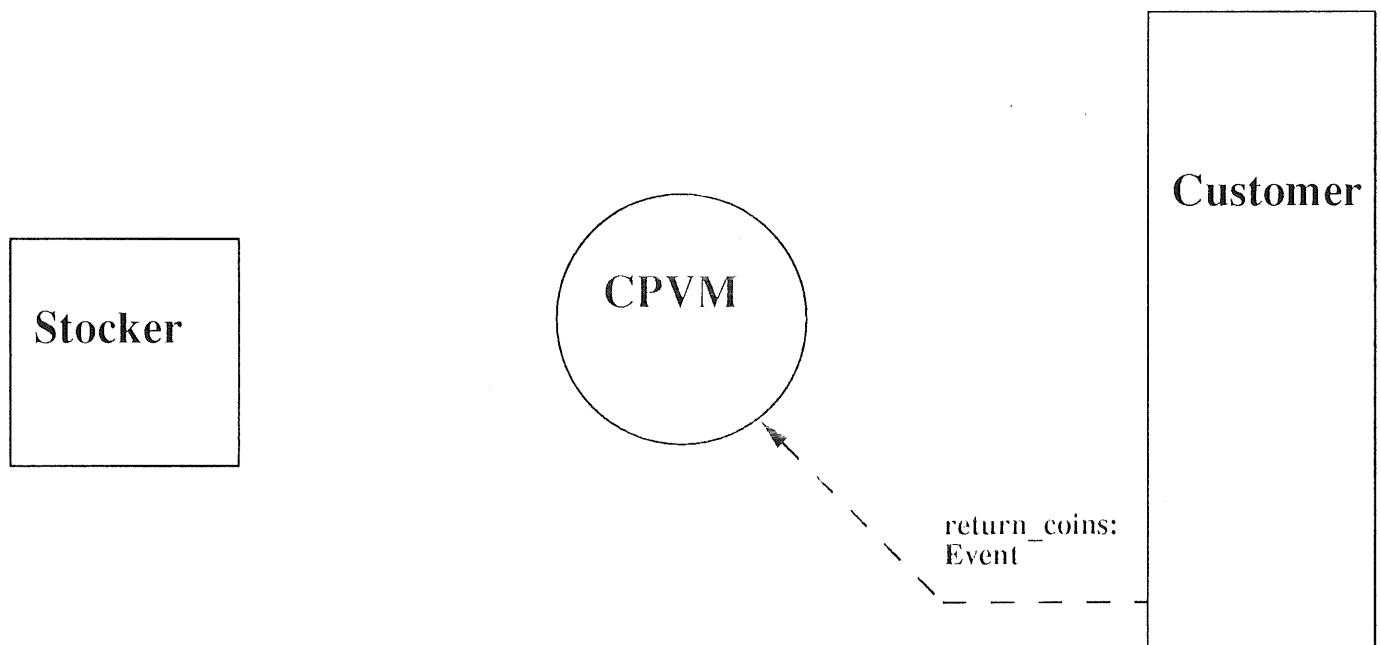
- product delivery mechanism
- change delivery mechanism
- product not available light

The controller must be 'crook proof' since vending machines are run on small profit margins, and losses on current vending machines are mounting.

A.2 CPVM data flow diagrams



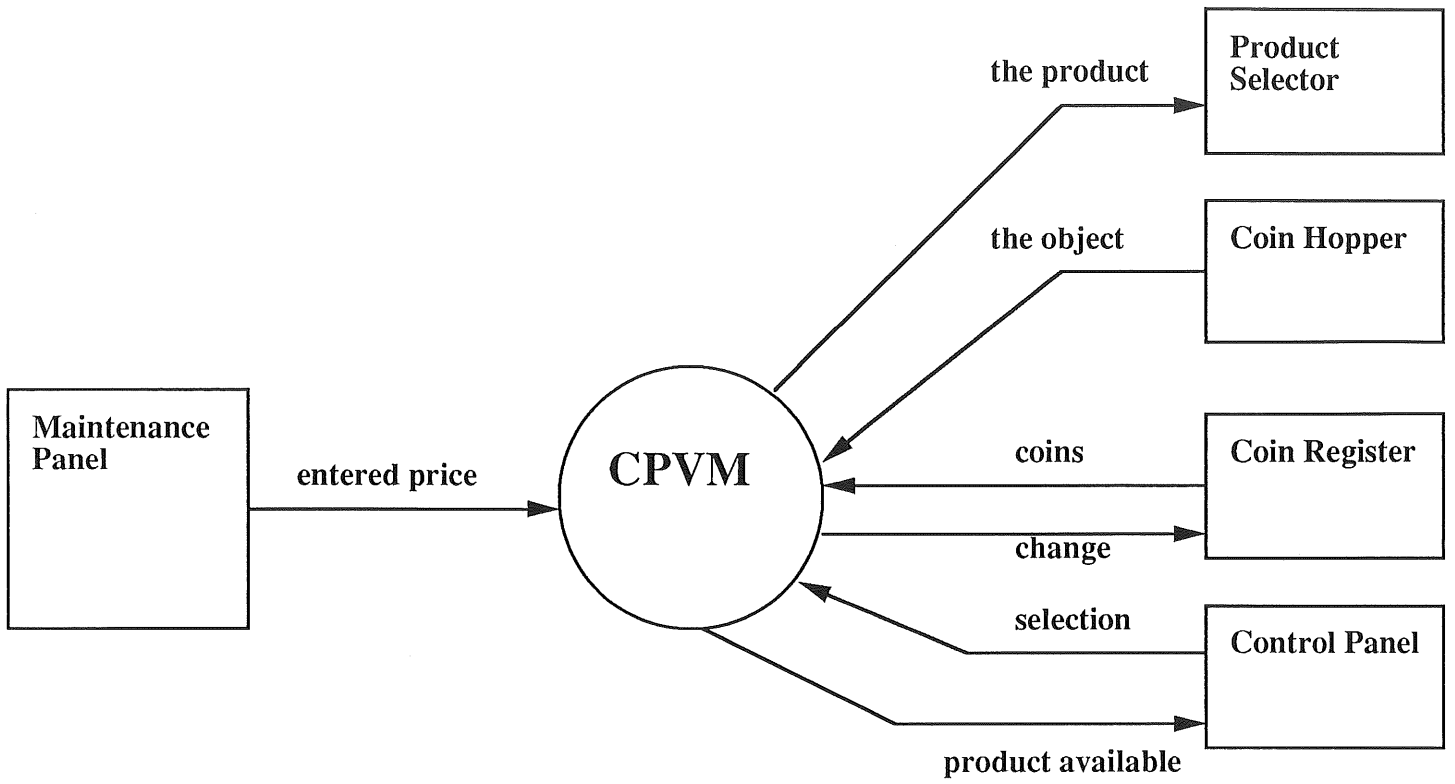
DATA CONTEXT



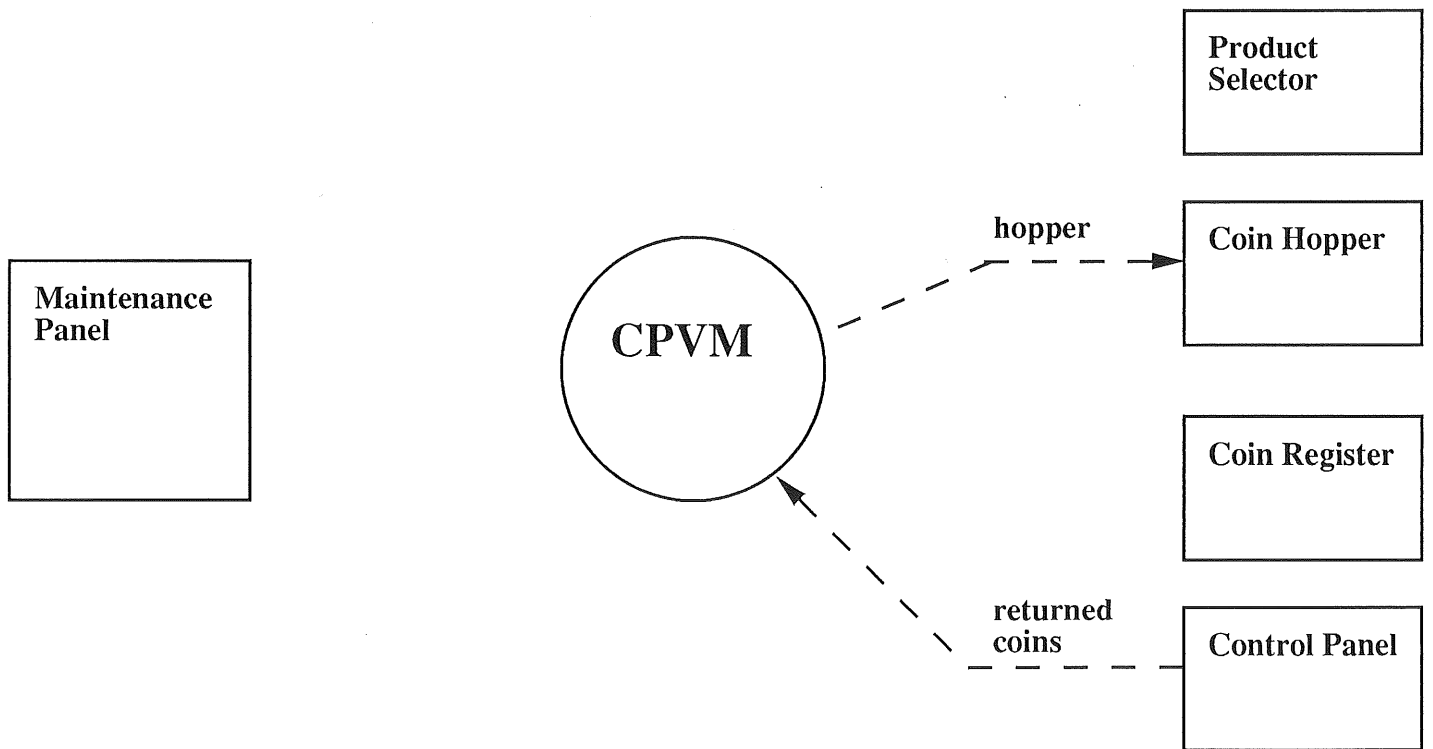
CONTROL CONTEXT

Crook Proof Vending Machine

(after Hatley & Pirbhai)

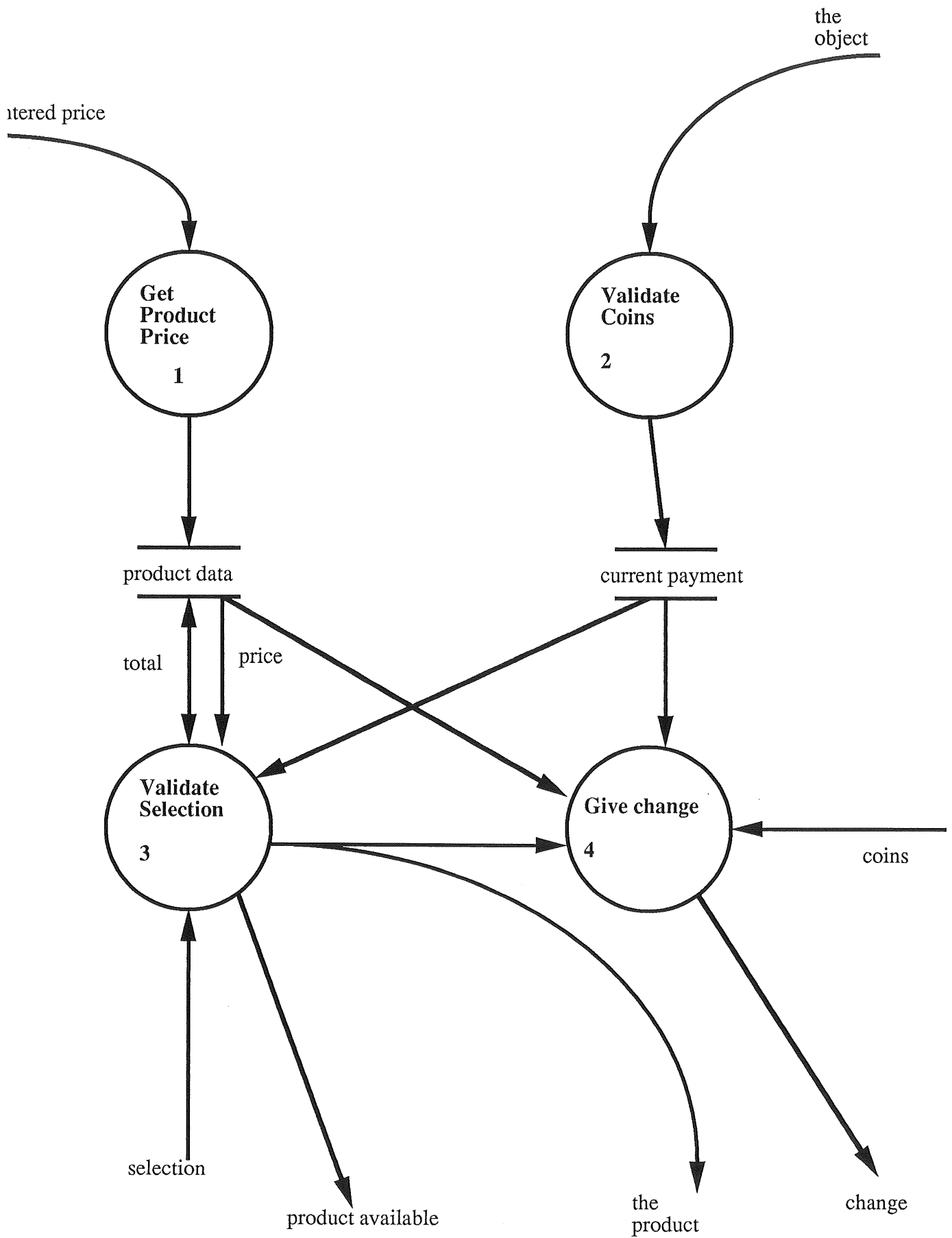


DATA CONTEXT

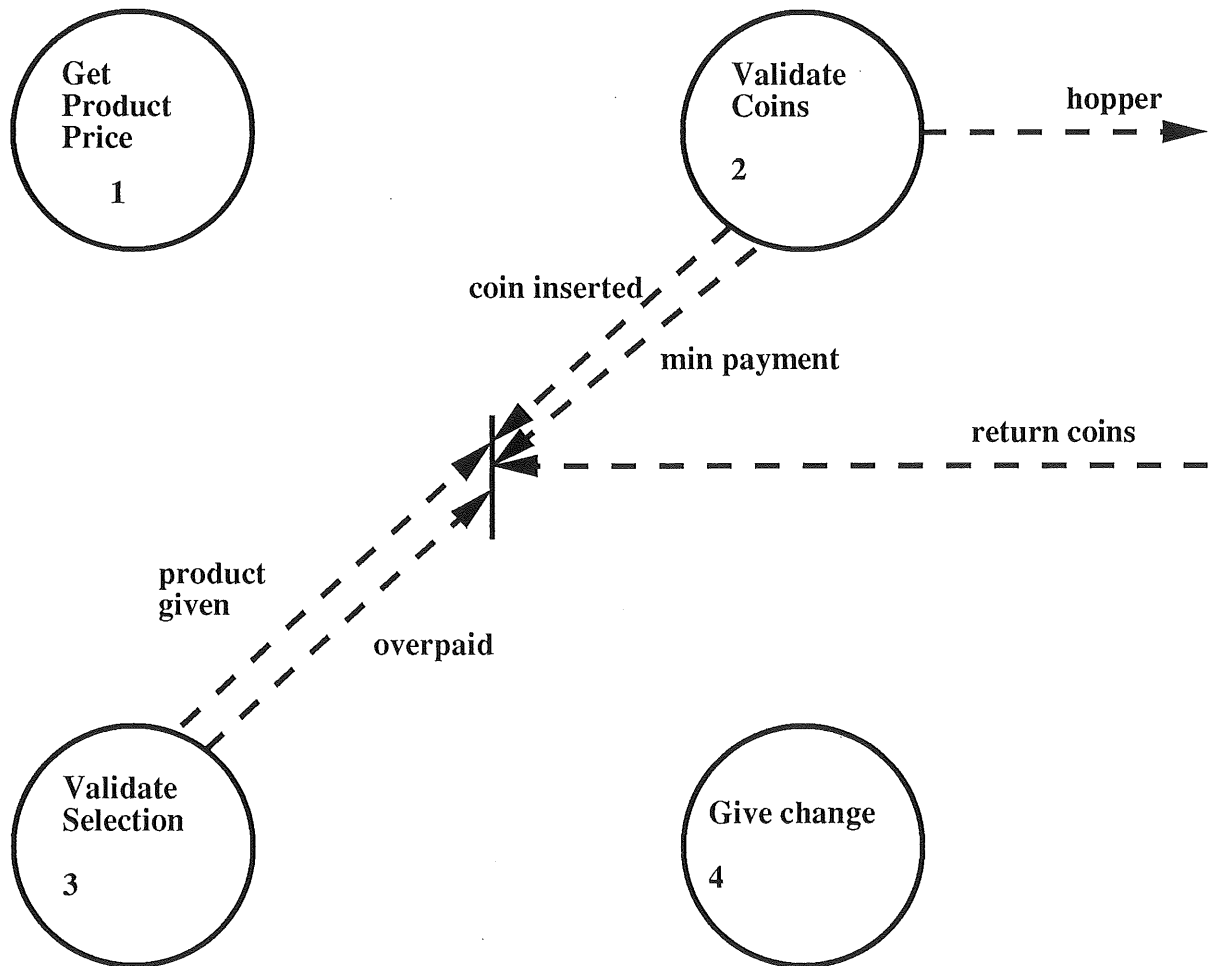


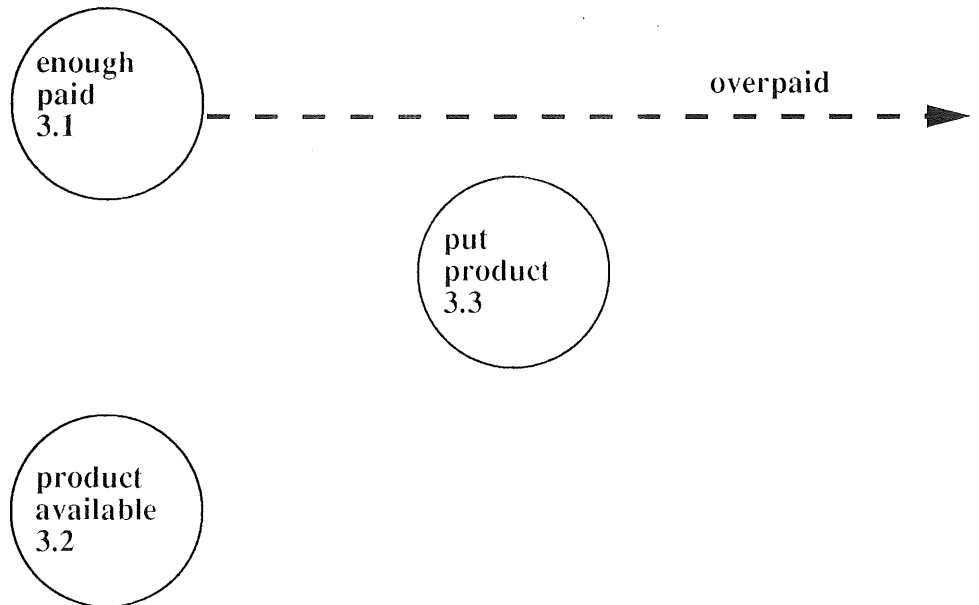
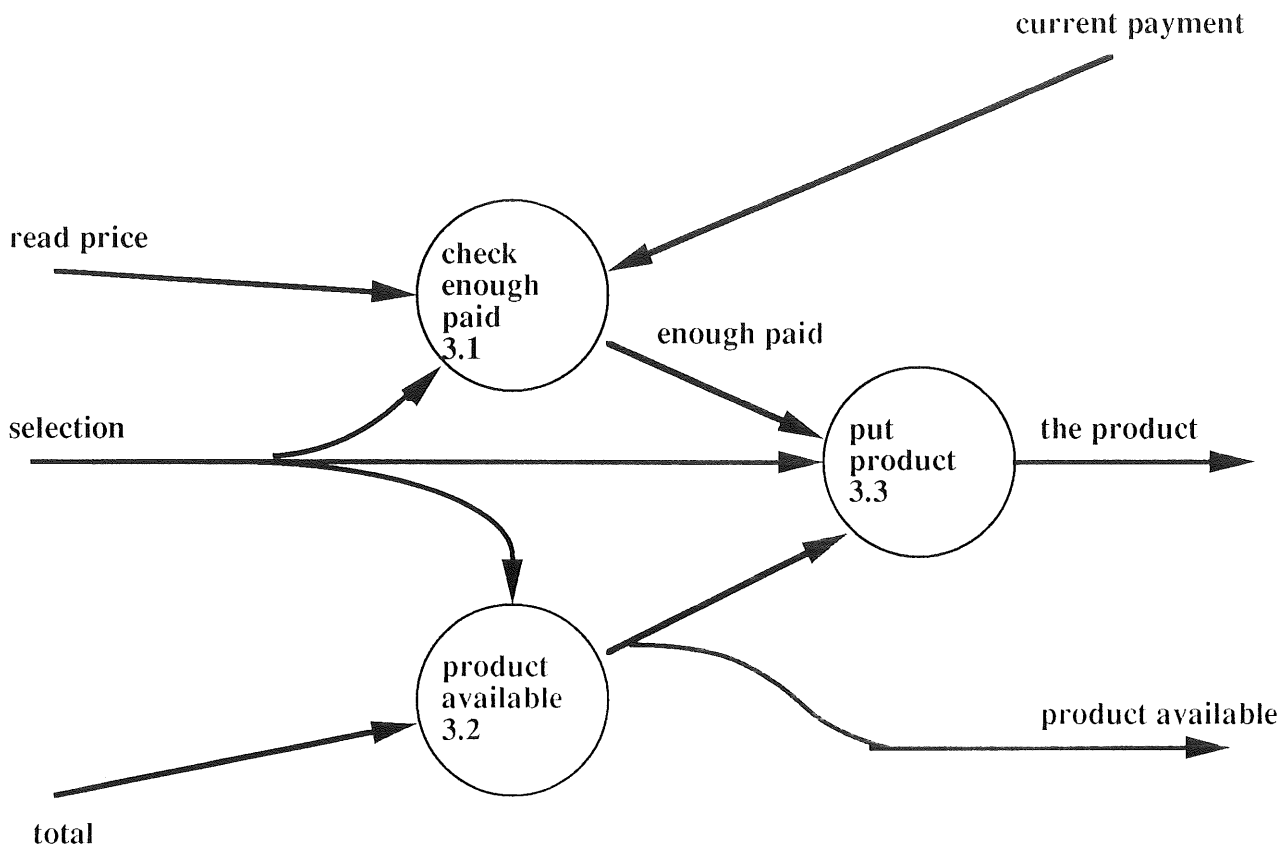
CONTROL CONTEXT

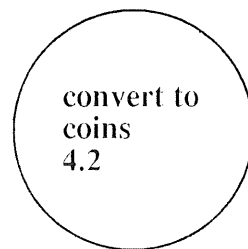
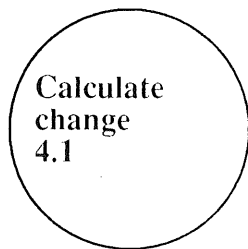
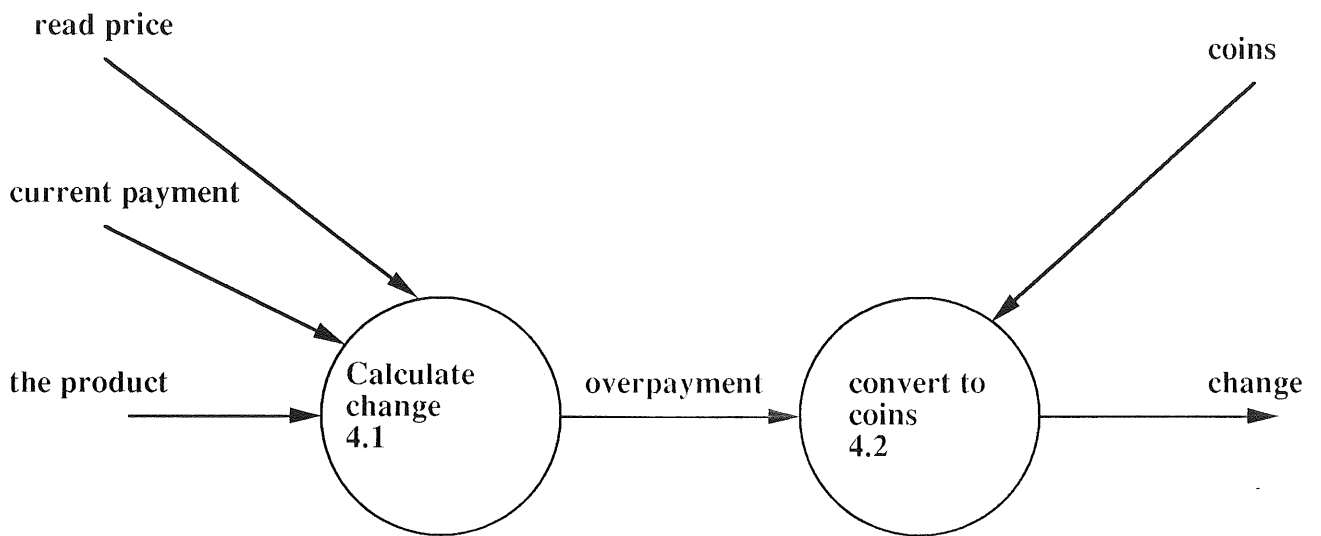
Crook Proof Vending Machine
(Physical data flow)



CPVM DFD 0 Level 1

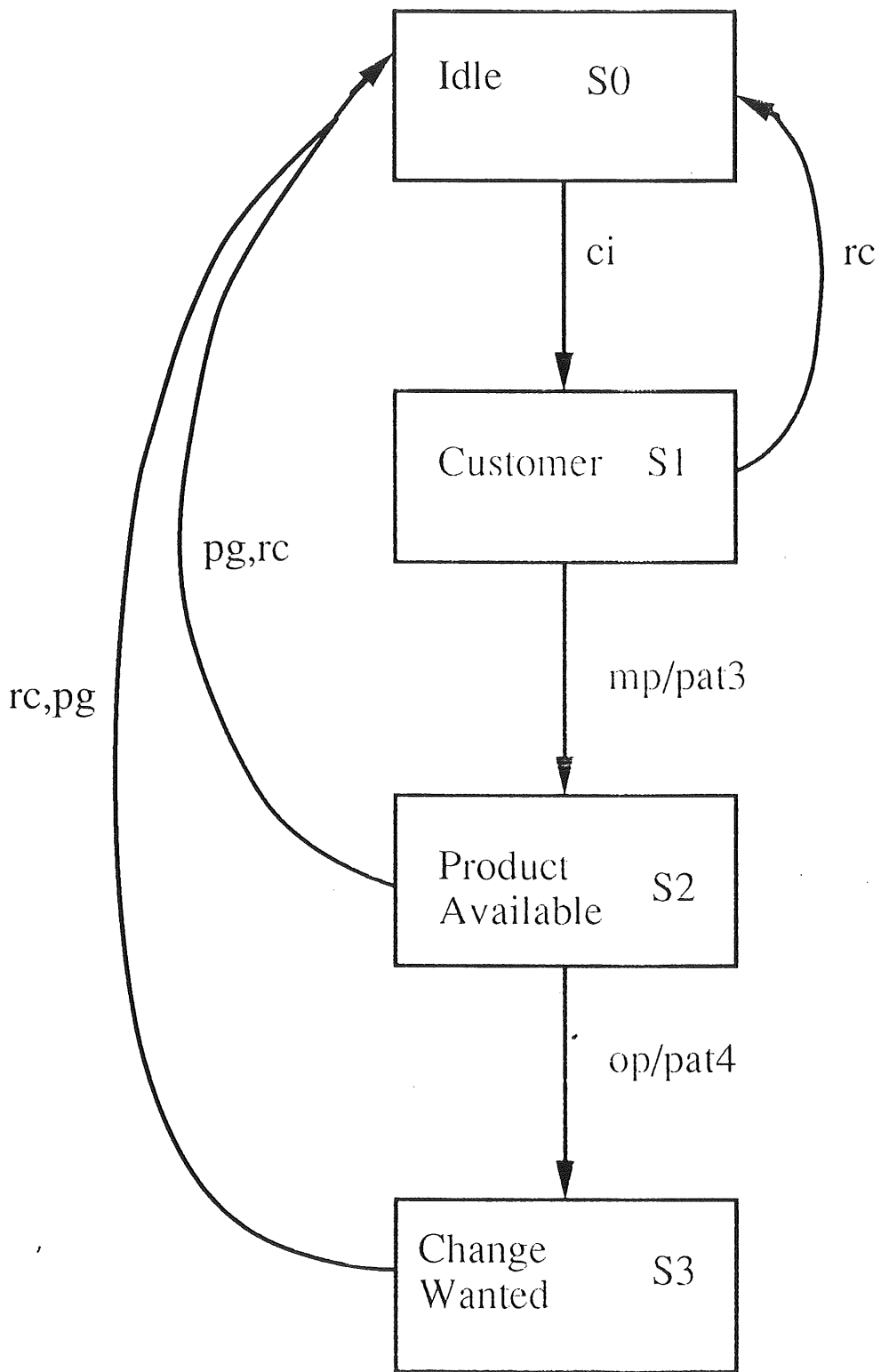






dfd/cfd 4

level 2

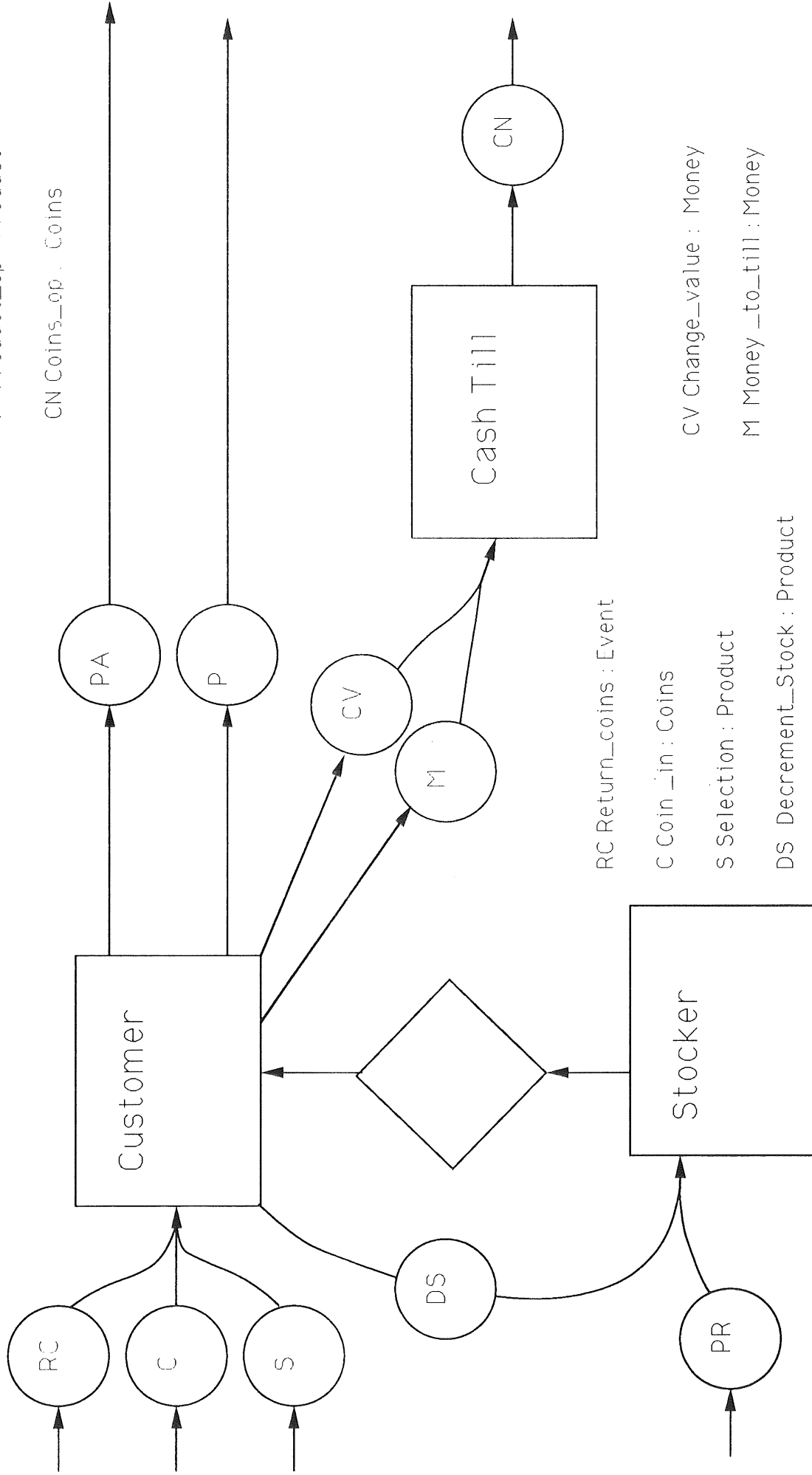


Type	definition	comments
CATALOGUE	PRODUCT X COST X GOODSHOPPER	
COINCOUNT	0..Maxcount	
COINHOPPER	COINCOUNT X COINCOUNT X COINCOUNT	50ps X 20ps X 10ps
COST	Mincost..Maxcost	
GOODSHOPPER	0 .. Maxhopper	
MONEY	Minmoney..Maxmoney	
OBJECT	10p 20p 50p slug	
PRODUCT	Beer Cola	assume just 2 products
PHYSICAL	DIAMETER X WEIGHT	
DIAMETER	Mindia .. Maxdia	
WEIGHT	Minweight .. Maxweight	
EVENT	null	no values

PA product_available : Boolean

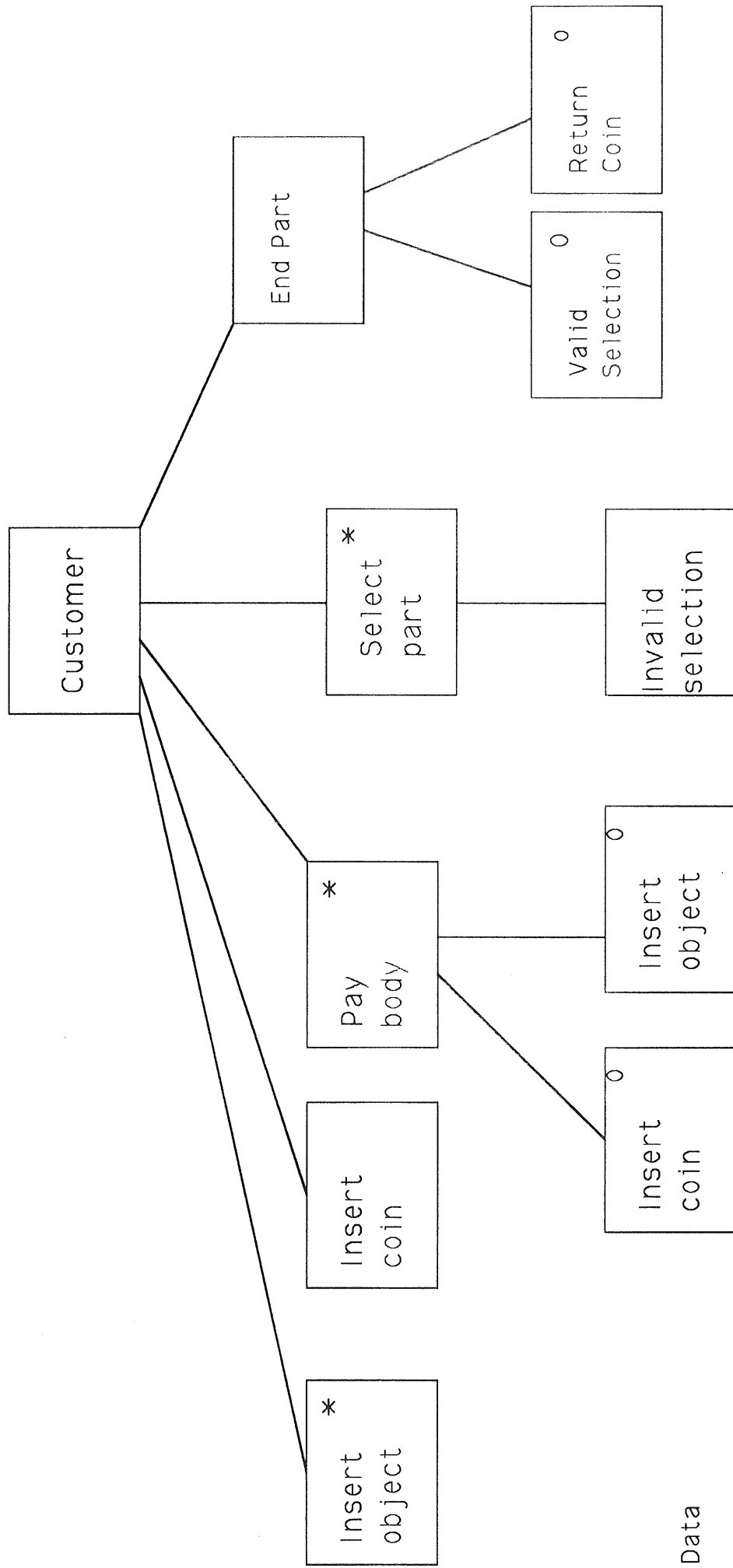
P Product_op Product

CN Coins_op : Coins



CPVM JSD

(logical model)



CPVM JSD entity structure diagram