Executable specification for information systems

Citation for published version (APA):

Document status and date:
Published: 01/01/1988

Publisher Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 02. Oct. 2020
Executable Specification for Information Systems
by
K.M. van Hee
G.J. Hoeben
L.J. Somers
M. Voorhoeve
88/02

January 1988
This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science of Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB Eindhoven
The Netherlands
All rights reserved
editor: F.A.J. van Neerven
Executable Specifications for Information Systems

K.M. van Hee, G.J. Houben, L.J. Somers, M. Voorhoeve

Eindhoven University of Technology

ABSTRACT
In this paper we present a survey of a framework for modeling and specifying information systems. Our method [4, 5] is supported by a software tool for checking and executing specifications. An executable specification may be considered as a prototype for a target system. The specification language resembles the language of mathematics; it is related to the Z and VDM methods [1, 3]. However, specifications in Z and VDM are descriptive (and therefore not executable), whereas ours are constructive. All these methods share almost the same power and versatility.

In section 1 we give our viewpoint on the engineering of information systems. In section 2 we give an informal treatment of our framework and design method. In section 3 we survey the specification language. In section 4 our solution to the inventory control case study [6] is presented and finally in section 5 we give a full specification of that case study.

1. INTRODUCTION
Software engineering is a branch of systems engineering focused on the automatic control of tasks in a system. A system is characterized formally by a state space and some transition mechanism that transfers the system from one state into another one. Here, we restrict ourselves to discrete dynamic systems, which means that we describe the behaviour of a system by a (possibly infinite) sequence of states.

In general the state spaces of a system can be considered as a cartesian product and therefore its states can be considered as vectors.

The transition mechanism often consists of one or more processors. The behaviour of a processor is described by a function that has two types of arguments: a trigger and a subvector of the system state. Its effect may consist of a change of the state subvector and a set of triggers for other processors or possibly itself. Processors may be implemented by persons, machines or computer systems. A state of a system may be determined by the presence of sets of physical or abstract objects such as products and agreements.

When considering a business system in more detail we distinguish a subsystem that executes the primary tasks of the business system, called the primary system, and a system that controls the primary system, called the control system.

Automated information systems play a role in control systems. One of the classical tasks of an (automated) information system is keeping track of the state of the primary system. A variable, called a
database, contains information about the actual states possibly combined with past states of the primary system. The case study belongs to this class of information systems.

A more advanced task of an information system is the support of decision makers in control systems. These subsystems are called expert systems or decision support systems.

In section 2 we sketch a formal framework to model such systems. The need for such descriptions is demonstrated by the wide use of dataflow diagram techniques in methods as SADT [9], ISAC [7] and Yourdon [10]. These techniques lack good semantics and hence are not suitable for the precise specification of an (information) system, although the diagrams often help to understand more formal specifications. Another approach to systems description is data modeling. It can be used for describing the state space of a primary system or its image in an information system. Data modeling techniques are much more formal, but they are only suitable for modeling state spaces.

Our approach seems to be interesting because it integrates formal modeling of processors and state spaces, combined with diagramming techniques. In earlier work [4] we used a similar framework, supported by a logic language.

Main objects to specify are variables, sometimes having a complex structure, and functions. In fact a database scheme defines a type for a variable called database. What we need is a type system that allows us to define rather complex types for variables, and a mechanism to define functions.

A typed lambda calculus or functional language seems to be a natural choice. This is the basis of our language, called EXSPECT. A nice feature of it is that we are able to stick to the relational model but that it is also possible to work in non-first-normal-form. In the application we have chosen for the last option.

Our software tool consists of an editor, a type checker and an interpreter. With the type checker one can test a description for type consistency. With the interpreter one can simulate the behaviour of the described system. This last facility is essential for validation purposes: for non-experts it is difficult to understand a formal system description. On the other hand it is relatively easy for future users of a system to validate a prototype, generated from an executable specification.

An important difference between an executable specification and a real implementation is that the designer of the specification is only concerned with the functionality of a system and not with matters like performance, system load, reliability, concurrency, etc. Therefore a specification language may use more powerful constructs than an implementation language; so it is much faster to design a specification than an implementation.

The first step in the lifecycle of an information system is the description of the environment, i.e. the primary system and possibly parts of the control system of a business system. It is possible to model this on several levels of detail within our framework.

One usually proceeds with requirements engineering as the next step. Here the tasks of an information system are defined. Usually the functional and non-functional requirements are written down informally.

We then have a preliminary specification. If it were used as a specification for implementation then with high probability the resulting implementation would be inadequate. We all know that system changes are very expensive. Therefore we advocate a third phase of the lifecycle that is devoted to a formal specification in the way sketched above. We call this conceptual modeling because this phase produces an abstract system that has the same functionality as the target system. With an executable specification we already have a primitive implementation of the target system.
2. FRAMEWORK AND DESIGN METHOD

Systems have three main aspects: state structure, data flow and control flow. Many methods are available for each individual aspect. As said before, state structure can be described by data models, data flow by data flow diagrams and control flow can be modeled for instance by Petri nets [8] or finite state machines.

Our framework integrates all three aspects. The difference between control flow and data flow in our framework is that the control flow directs the transport of parameters to processors, triggering the incorporated functions, while data flow means transport of parameters between a processor and a (stored) variable. For a formal treatment we refer to [4]. We call systems that fit into our framework Distributed Event Systems (DES). The term event is used to describe the triggering of a state transition and discrete means that each state on a process path has a successor. A DES is always a closed system, so a target system and its environment together form a DES. Of course we will not specify all components of the environment. We shall treat this subject later in more detail.

A DES is completely determined by a 8-tuple $\langle S, C, IC, OC, M, R, IS, OS \rangle$.

Before we explain the meaning of these components we give two diagrams of a DES. Of course it is possible to combine the two diagrams into one.

![Diagram of DES](image)

Fig. 1

The triangles $P_1 \ldots P_4$ represent processors. Each processor $i$ consists of two functions: $M_i$ and $R_i$. The circles $S_1 \ldots S_7$ represent (stored) variables. They may have simple structures like a calendar date, or complex like a database. The connections between processors and stores mean that a processor may access the variable. If there is an arrow in the direction of the variable then it is an output variable for the processor, if an arrow points to the processor it is an input variable. Note that there is no direct data flow between two processors. However, it is possible to transmit data from one processor to another as indicated in the second diagram. Each processor has one input channel and it may have several output channels.

For each channel, the type of the values that may pass through it are determined. Channels may split and join. Processors have a single input channel; they are triggered by the values arriving through that channel. Note that the type of a channel may allow very complex values. It is easy to deal with cases where it is intuitively felt natural to have more than one trigger channel. Trigger channels may be considered as mailboxes. The values passing through a channel are called triggers.
In fig. 1 we have already met four of the components of the 8-tuple:
- **IC** is a function that assigns to each processor one input channel-index, in the picture $x, x, y, z$ for respectively $P_1, P_2, P_3$ and $P_4$.
- **OC** is a function that assigns to each processor a set of output channel-indexes for $P_1 : x, y$, for $P_2 : y, z$ etc.

An output channel is connected to all input channels with the same index.
- **IS** is a function that assigns to each processor a set of indexes of (stored) variables that are used as input variables for that processor; for $P_1 : S_1$, for $P_2 : S_2$ and $S_3$ for $P_3 : S_3$ etc.
- **OS** is a function similar to **IS**, it assigns to each processor a set of indexes of output variables; for $P_1 : S_1, S_2$, for $P_2 : S_3, S_4$, for $P_3 : S_4, S_5$ etc.

Now we will explain $S$ and $V$.
- $S$ is a set-valued function, where $\text{dom}(S)$ is the set of indexes of stored variables and for such an index $i$, $S_i$ is a set that represents the type of the variable with index $i$.
- $C$ is a set-valued function, where $\text{dom}(C)$ is the set of channel indexes and for such an index $j$, $C_j$ represents the type of the triggers passing through the channel.

Finally we return to $M$ and $R$.
- $M$ is a function-valued function, where $\text{dom}(M)$ is the set of processor indexes. For a processor $k$, $M_k$ is a function with input variables with indexes in $IS_k$ and $IC_k$ and output variables with indexes in $OS_k$.
  $M_k$ is called the **manipulator** of processor $k$ because it may modify the stored variables.
- $R$ is also a function-valued function, where $\text{dom}(R)$ is the set of processor indexes. For a processor $k$, $R_k$ is a function with the same input variables as $M_k$ however its result is a partial function that assigns a value to zero or more trigger variables with indexes in the set $OC_k$. $R_k$ is called the **reactor** of processor $k$ because it produces triggers.

The functions $M_k$ and $K_k$ are specified by means of a typed lambda calculus or functional language. This is treated in section 3.

We will describe the behaviour of a DES in an informal way. For every input channel there is a multiset of triggers. At each moment a processor $k$ having a non-empty multiset of triggers may commit a transition which consists of the following actions:
- selection of a trigger from the available triggers,
- simultaneous computation of $M_k$ and $R_k$ with as input parameters the values of the input variables and the trigger value.

At the same moment, several processors may commit a transition, however no two processors sharing a stored variable that is an output variable may commit at the same moment. It is required that each produced trigger value is taken into execution at some moment, so a system must be starvation-free. Note that we not specify how processors select triggers from their multiset, nor how they control the exclusive updating of output variables. It is left to the implementers to choose a solution for these problems. It is easy to find a solution by committing transitions for processors sequentially, however it is often desired to exploit parallelism. Since for a DES the selection of triggers to be executed is not specified, it may be considered a non-deterministic system.
Many systems may be modeled as a DES, for instance many communication protocols between two systems can be modeled explicitly within the framework. Then we model in fact the communication at a higher level, while the implicit way of triggering is the lowest level of communication.

An important modeling issue, in connection with the case study, is the separation of a closed system into a target system and its environment. The target system is the information system we want to develop. Many information systems can be modeled at a high level as a reactive system, i.e. the environment offers a trigger and the system performs one transition and a trigger for its environment. In that case there is no internal triggering in the system. We may model the environment as one or more processors, possibly with stored variables. However, the specification of these processors is unknown, as are their stored variables.

The processors are considered as black boxes, their in- and output channels only are known (see fig. 2).

Blackbox $x$ may trigger processors 1, 2, 3 and 4, blackbox $y$ 4, 5 and 6, and every processor is producing a trigger for the invoker, however processor 4 may trigger both blackboxes.

It is also possible to model that a processor in the environment may access stored variables of the target system.

In the case study most system tasks are of the reactive type. We can model the several access control classes as different blackbox processors.

We conclude this section with some remarks on a design method based on our framework. We only consider the conceptual modeling phase. It is quite natural to proceed along the following steps:

1. Identify the processors and stored variables in the target system and identify the blackboxes in the environment (in fact this is a data flow analysis).
2. Identify the channel structure (control flow analysis).
3. Define types for the stored variables (data modeling).
4. Define constraints on the types of stored variables (database constraints).
5. Define types for the trigger variables.
6. Define the manipulator and reactor functions for each processor.
7. Verify that the processors keep the constraints invariant.

Of course, it is sometimes useful to change the order or to take on two steps simultaneously. However, if all 7 steps are accomplished the specification is complete.
3. LANGUAGE

In this section we describe a language for specifying systems according to the model defined in section 2. The description is given in an informal way, for a more concise treatment we refer to [5].

A DES is built from processors and stores. A processor repeatedly selects a trigger from the available triggers, updates the values of the stores it is connected to and sends new triggers to other processors. The language EXSPECT, of which a subset is treated here, is suited for specifying and executing such systems.

From now on we call stored variables stores and they may be considered as global variables. They are declared by giving their name and type. For example,

```
store s: str
```

declares a store of name s and type string.

Processors are defined by giving their name, the type of the input trigger, and the actions they perform. For example,

```
proc pl[i:str] ::= s ← i,
    q <= 'store updated'
```

This processor is named pl and is triggered by a string. When pl reacts upon a certain trigger it stores the value of this trigger, which is denoted by i, in the store s we have declared above. Furthermore it sends a trigger with value 'store updated' to processor q. In the present version of EXSPECT, input channels cannot be shared by processors and therefore we identify a processor and its input channel.

For each store updated there is a line containing a `<--` and for each trigger sent there is a line with a `<=`.

In general, processors also transform input values of stores and triggers into other values. This is where the functional aspect of EXSPECT comes in. At the right hand side of a `<--` or `<=` sign we may use any function of the input trigger of the processor and the stores in the system, which are connected to this processor as input stores. A function is defined in terms of other functions and so on till we reach the basic functions of the language. For example, we may define a function to calculate the length of a string

```
strlen[x:str] ::= if x = '' then 0
    else strlen(tail(x)) + 1
    fi
```

with the help of already existing functions to add numbers (+) and to take all but the first character of a string (tail).

This new function in turn can be used for defining other functions.

Assignments to stores and triggers in the definition of a processor can also be done conditionally. A processor p2 which only updates store s when the length of the trigger is more than 10 is given by

```
proc p2[i:str] ::= if strlen(i) > 10 then
    s ← i,
    q <= 'store updated'
else
    q <= 'store not updated'
    fi
```
Untouched stores, like \( s \) in the second alternative of the \( \text{if} \), are left invariant.

Up to this point we have dealt only with simple types like "bool", "num" and "str"; respectively for boolean values (true/false), rational numbers and strings. For modeling more real-life situations we need more complex types like sets.

With the help of the type constructors \( \times \), \( \$ \) and \( \rightarrow \) we can construct compound types of arbitrary complexity.

Cartesian products are constructed with the help of \( \times \), for example pairs of numbers, or triples of boolean, number, string:

\[
\text{num} \times \text{num} \\
\text{bool} \times \text{number} \times \text{string}
\]

Sets are constructed with the help of \( \$ \) and mappings (functions with finite domains, to be interpreted as sets of pairs) with the help of \( \rightarrow \). Examples are

\[
\$\text{num} \\
\text{num} \rightarrow \text{bool} \\
\text{num} \times \$\text{bool}
\]

for a finite set of numbers, a mapping from num to bool and pairs of sets of numbers and bools.

We are now able to define a processor \( p_3 \) that updates a store \( t \) that holds a set of strings,

\[
\begin{align*}
\text{store } & t: \$\text{str}; \\
\text{proc } & p_3[i: \text{str}] := \text{if } \text{strlen}(i) > 10 \text{ then } t \leftarrow \text{ins}(i, t), \\
& \quad q \leftarrow \text{'store updated'} \\
& \quad \text{else } q \leftarrow \text{'store not updated'}
\end{align*}
\]

Here "ins" is a basic function that inserts an element in a set.

New types can also be introduced by giving them a name. We can introduce a type \( \text{addr} \) (ess), which holds street, house, town and postal code (all considered to be strings) by

\[
\begin{align*}
\text{type } & \text{addr from } \text{str} \times \text{str} \times \text{str} \times \text{str} \\
\text{store } & \text{index: name } \rightarrow \text{addr}
\end{align*}
\]

We have used a mapping, since for each name there is never more than one address.

A processor \( p_4 \) that adds a name, which is not yet present, and address to "index" may be written as

\[
\begin{align*}
\text{proc } & p_4[i: \text{name} \times \text{addr}] := \text{if } \pi_1(i) \notin \text{dom}(\text{index}) \text{ then } \text{index} \leftarrow \text{ins}(i, \text{index}) \\
& \quad \text{fi}
\end{align*}
\]

The function \( \pi_1 \) projects upon the first element of a pair. This process can also be written as

\[
\begin{align*}
\text{proc } & p_5[i: \text{name}, j: \text{addr}] := \\
& \quad \text{if } i \notin \text{dom}(\text{index}) \text{ then } \text{index} \leftarrow (x: \text{ins}(i, \text{dom}(\text{index}))) \; \mid \; \text{if } x = i \text{ then } j \text{ else } \text{index} \cdot x \\
& \quad \text{fi}
\end{align*}
\]

The constructor \( [x: \text{Di E}(x)] \) defines a mapping with domain \( D \) and range \( E(D) \). So the example assigns to (the store) index a new mapping that has the same domain as the old index, but with the new name added. The values of the mapping are the old ones (\( \text{index} \cdot x \) means apply index to \( x \)) and the new address.
Yet a third way to represent the above processor is

```
proc p6([i:name, j:addr]::
  if i ∈ dom(index) then index ← fupd(index, [x: [i] l j])
fi
```

Here \( \{i\} \) is the set with \( i \) as only element, \( [x: [i] l j] \) is therefore the mapping consisting of only one pair \( \langle i, j \rangle \) and "fupd" (defined formally in section 4.4) is a general function that accepts two mappings as parameters and returns the "overwriting" of the first mapping by the second one, it is defined formally in section 4.4.

In the above we have given enough information about the language to understand the case in the next section.

It is possible to construct libraries of functions. These libraries will assist in developing a description in a modular way.

Apart from libraries of functions one can also make toolboxes of parametrized processors or networks of processors. These can be used to assemble a system from existing parts. The part of the language that deals with these modular networks is not treated in this paper, since the case of the next chapter is essentially a flat one.

### 4. THE INVENTORY CONTROL SYSTEM IN EXSPECT

#### 4.1 Control Flow and Data Flow

The first step in designing an EXSPECT prototype for an information system consists of designing the control and data flow of the various processors of the system. First we must draw a boundary between the system and its environment.

In the inventory control system of the case study [6], the environment consists of a number of users who can perform a selection out of several tasks. A dialogue guides the user to the task he wishes to perform and prompts him for the right parameters for this task. Before even this dialogue starts, the user must login to the system, whereby his access control class becomes known.

We have chosen to exclude the dialogue and access control part from our system because it is not typical for this case. Therefore our environment consists of a number of user agents, who can trigger any of the task processors. A few background task processors are not triggered by any user agent but by some of the foreground processors. Our control flow scheme thus becomes as follows.

![Diagram](image-url)

Fig. 3
For the data flow, we model our database as a single stored variable. The user agents have no access to it; some of the processors ("queries") only consult the database, others ("updates") also modify it. The data flow scheme thus becomes as follows.

![Diagram](Fig. 4)

For maximal clarity (since our goal is a prototype) we have reduced the number of user agents to one. This single-user system can be converted into a multi-user one by extending the trigger of each foreground processor with the user agent index of the caller and adding code to send the response to the caller.

### 4.2 Datatypes and Stores

The second step in designing the prototype is to design a structure for the stored database variable. This step is (for a strongly data-oriented case like this) more important than the preceding one. As mentioned in the introduction, we can choose to do so in various ways, ranging from many "flat" parts to few "structured" ones. To demonstrate the data structuring capabilities of EXSPECT we have chosen for this last option. To understand the following discussion one must study the case description [6].

We divide the database into five parts, called respectively the stocked item type file (sitf), stock item file (sif), supplier file (supf), purchase order file (pof) and the calendar (cal). Since we have no need for the database as a whole, we model these parts as separate stores.

The more or less compound "attributes" of the above stores are often described by defining a special "derived" datatype for them; these datatypes also serve as a vehicle for triggering processors. Inside the user agent formatting and checking information could be attached to them.

The stock item type file (sitf) is as specified in the case; it consists of attributes stock item type code (sitc) and description (sitd). Since the sitc attribute must be unique, we model sitf as a store of type "mapping of sitc to sitd", where sitd and sitc are both types derived from "string". We write this formally as follows.
type sitc from str;

type sitd from str;

store sif : sitc → sitd

The value for this variable as given in the case description would be represented as the following set of pairs.

```
{ << 'E', 'Office Equipment (capital expense)' >>,
  << 'S', 'Stationary supplies' >>,
  ...
  << 'K', 'Kitchen supplies' >> }
```

The stock item file (sif) consists of the "flat" attribute structure as specified in the case: stock item code (sic), stock item type code (sitc), stock item description (sid), replenishment level (type qty: quantity). To this is added a "history" component, containing the recorded stock levels (date and qty) together with the withdrawals (qty and issue) and replenishments (qty and purchase order responsible for it) at that moment. The date forms a key to a recorded stock level. Our "sif" store thus combines the stock item, stock on hand, replenishment and withdrawal files in the case description.

We could have added all stock items of a certain type as an attribute to the same stock item type in the "sif" store. This would however make the retrieval of a stock item on its code quite cumbersome. Remodeling the stock item code as a pair << sitc, n >> removes this disadvantage. At the same time it would be nice to deduce the item type directly from its code without accessing the database. We have however stuck to the description as given and therefore chosen to model "sif" as a separate store as follows.

```
type date from num;

type sic, sid, qty from num;

type sidat from sic × sid × qty;

type ponr from num; -- purchase order nr

type wdr from qty × str;

type repl from qty × ponr;

type history from date → (qty × wdr × repl);

store sif : sic → (sidat × history)
```

A possible value for the "sif" variable would be as follows.

```
{ << 5632, << 'E', 'Compaq Plus Computer', 0 >>, {} >> >>,
  << 2389, << 'F', 'Paracetamol', 144 >>,
    { << 870901, { << 1,'headache' >>, << 2,'lost' >> }, {} >> ,
     << 871111, { << 24,'lost' >>, { << 288, 74324 >> } } } } >> }
```
The supplier file (supf) with key supplier number (supnr) has as attributes the name, address and phone number of the supplier plus the set of item types he sells. This store thus combines the supplier and supplier of stock item type files. We model it as follows.

\[
\text{type supnr from num;}
\]
\[
\text{type phone, addr, name from str;}
\]
\[
\text{type supdat from name x addr x phone;}
\]
\[
\text{store supnr : supnr → (supdat x \$ site)}
\]

The purchase order file (pof) with key purchase order number (ponr) has as attributes the order date and the supplier plus the set of ordered items. This set is modeled as a mapping (pol) from "sic" to price (per unit) and quantity. This store thus combines the purchase order and purchase order line files. It is described as follows.

\[
\text{type podat from date x supm;}
\]
\[
\text{type pol from sic \rightarrow (price x qty);}\]
\[
\text{store pof : ponr → (podat x pol)}
\]

The calendar (cal) consists of a single date variable.

\[
\text{store cal : date}
\]

After defining the stores and auxiliary types, it is helpful to define auxiliary functions based upon these stores. For instance, the order date of an order \(x\) is represented much more nicely by the expression "date \((x)\)" than by \(\pi1 (\pi1 (\text{pof} \cdot x))\). It does not matter that there exists already a type "date", because the parser knows when to expect a type or an expression. There could even be more functions named "date", provided their parameter types do not conflict.

One of the more complicated auxiliary functions computes the stock level of item \(x\) at date \(y\). Since we only record stock level changes, this involves searching the history of \(x\) to find the last recorded change before or at date \(y\). If item \(x\) has no history at or before date \(y\) (e.g. at date \(y\) it had just been decided to keep the item in stock and orders had been placed but no supply had arrived yet), the function must return 0. In full the definition reads

\[
\text{qty [x:sic, y:date] :=}
\]
\[
\text{if S[t: \text{dates}(x) \mid t \leq y] = \{\} then 0}
\]
\[
\text{else } \pi1 (\text{hist}(x) \cdot \max (S [t: \text{dates}(x) \mid t \leq y])))
\]

The expression "\(S[t: \text{dates}(x) \mid t \leq y]\)" denotes the set of stock level change dates for the item \(x\) that lie before or at the date \(y\). Taking the maximum of this set gives the last recorded change date before or at \(y\). Applying the history to this date and taking the first part yields the stock level recorded at that date.
4.3 Constraints

The next step in the design process is the formulation of constraints. These are computable boolean expressions depending on the store contents. The designer of the prototype must show that every processor changing the store contents leaves the constraints invariant, i.e., assuming that they are true in the old state, they must be true in the new state too.

In our design for the inventory control case, a lot of constraints as formulated are immediately guaranteed by the store definition. For instance, uniqueness constraints are met by defining stores as mappings. A lot of referential constraints are met by combining files into a single non-first-normal-form store. There are some referential constraints left, for instance

"each stocked item has an existing type"

which is represented as

∀ [x: dom(sif)] type(x) ∈ dom(sif).

In studying the above formula, one sees how closely EXSPECT text resembles conventional mathematical notations.

There are some more interesting constraints, not mentioned in the case description; for instance,

"for each replenishment of a certain item, there must exist an order line for the same item; the replenishment date must not exceed the order date; the sum of all quantities replenished for the same order may not exceed the quantity ordered."

The EXSPECT formulation of the above constraint (using some earlier defined notions) becomes

∀ [x: dom(sif)] ∀ [y: dates(x)] ∀ [z: repls(x,y)]

po(z) ∈ dom(pof) and x ∈ items(po(z)) and date(po(z)) ≤ y

and replenqty(x, po(z)) ≤ orqty(po(z), x)]

The latter expression may be harder to understand (and to formulate) than the former, its meaning is uniquely determined. If legal texts were written in EXSPECT, a lot of lawyers would lose their jobs.
4.4 Processors

The following step in designing a prototype is to specify the diverse processors in the system. Having specified the structure of the stores, we must adapt the functionality of the system. The automatic generation of purchase orders is impossible, since data is lacking (suppliers per item, price per supplier-of-item). Instead, this processor produces a list of items that have to be ordered.

Now is the moment to go back to step 1; we identify the 25 processors (23 foreground and 2 background) as given in the case description, and specify their control flow in greater detail; also the data flow can be specified in more detail, having distinguished 5 stores.

Our task then becomes to determine the trigger types of each processor and determine its definition. For the user agent, we define the trigger "report" of type string.

As an example we treat action 4 of [6] (the addition of new items). It requires as input a list of items, each item consisting of item code, type code, item description and reorder level. Since the item codes are all different, we model the input type as

\[ \text{sic} \rightarrow \text{sidat} \]

To keep our constraints invariant, the item codes must be new and the type codes (included in sidat) must exist already. If these input requirements are met, the items are added to the "sif" store together with an empty history. To achieve this we call the input \( x \) and define the mapping

\[ f := [y: \text{dom}(x)] \prec x \cdot y, \{\} \succ ] \]

So \( f \) is a mapping derived from \( x \); each \( y \) in its domain is mapped to the pair formed by the value of \( x \) in \( y \) and the empty set. This \( f \) is thus the transformation of the input to "sif"-compatible format; the "sif" store is updated with this \( f \). If the input requirements are not met, a message is sent to the user. In full the specification of action 4 is as follows.

\[
\text{AddItems} \ [x: \text{sic} \rightarrow \text{sidat}] := \\
\text{if } \text{dom}(x) \cap \text{dom}(\text{sif}) = \{\} \\
\text{then if } \pi \{ \text{rg}(x) \} \subset \text{dom}(\text{sif}) \\
\text{then } \text{sif} \leftarrow \text{fupd} (\text{sif}, [t: \text{dom}(x)] \prec x \cdot t, \{\} \succ ) \\
\text{report} \leftarrow 'ok' \\
\text{else } \text{report} \leftarrow '\text{undefined key in sif}' \\
\text{else } \text{report} \leftarrow '\text{key conflict in sif}' \] 

The generic function \text{fupd} used here accepts two mappings \( f \) and \( g \) of type \( A \rightarrow B \) and returns the mapping \( h \) with as domain the union of the domains of \( f \) and \( g \); an element \( x \) in the domain of \( h \) is mapped to \( f \cdot x \) if \( x \) was in the domain of \( f \), otherwise to \( g \cdot x \). Formally

\[
\text{fupd} [f: A \rightarrow B, g: A \rightarrow B] := \\
[x: \text{dom}(f) \cup \text{dom}(g)] \text{if } x \in \text{dom}(f) \text{ then } f \cdot x \text{ else } g \cdot x \] 

In this way, we have modeled each of the 25 actions of [6]. A few concluding remarks have to be made. Action 13 (adding dates) is altered to setting a new system date. Also we have "sinned" by letting background processors perform checks and report to the user agent.
5. SPECIFICATION TEXT

-- types

type date from num;
type qty from num;
type phone from str;
type addr from str;
type price from num;
type sitc from str;
type sitd from str;
type sic from num;
type sid from str;
type sidat from sitc >< sid >< qty;
type supnr from num;
type supdat from str >< addr >< phone;
type pol from sic -> (price><qty);
type ponnr from num;
type podat from date >< supnr;
type wdr from qty >< str,
type repl from qty >< ponnr;
type history from date -> (qty><$wdr><$repl);

-- stores

store sitf : sitc -> sitd;
store sitf : sic -> (sidat><history);
store supf : supnr -> (supdat><$sitc);
store pof : ponnr -> (podat><pol);
store cal : date;

-- auxiliary functions (data)
data [x:sic] := m1(sif·x);
type [x:sic] := m1(data(x)));
relev [x:sic] := m3(data(x));
hist [x:sic] := m2(sif·x);
dates [x:sic] := dom(hist(x));
qty [x:sic, y:date] := if $[t: dates(x) l t ≤ y] = {} then 0 else
	 m1 ( hist(x) · max ($[t: dates(x) l t ≤ y] ) ) fi;
curqty [x:sic] := qty (x, cal);
wdr[x:sic, y:date] := if y ∈ dates(x) then m2(hist(x)·y) else () fi;
reps [x:sic] := U(m3(rg(hist(x))));
repl[x:sic, y:date] := if y ∈ dates(x) then m3(hist(x)·y) else () fi;
po [x:rep] := m2(x);
reporset [x:sic, y:ponr] := $[t: reps(x) l po(t)=y];
reporqty [x:sic, y:ponr] := Σ[t: reporset(x,y)] m1(t);
orlines [x:ponr] := m2(pof·x);
items [x:ponr] := dom(orlines(x));
orders [x:sic] := $[t: dom(pof) l x ∈ items(t)];

-- yymmddd
-- natural numbers only
-- leading zero's are significant
-- street + house + town + code
-- multiples of .01
-- stock item type code
-- sit descr
-- stock item code
-- si descr
-- type, descr, replenishlevel
-- supplier number
-- name, address, phone
-- purchase order lines:
-- si code, unitprice, qty
-- purchase order nr
-- po data
-- qty and issue of withdrawal
-- qty and purchase order
-- stock history of item:
-- moment of change, new qty,
-- set of withdraw. and repl.
sup [x: ponr] := \pi2_m1(pof\cdot x));           -- supplier of order x
date [x: ponr] := \pi1_m1(pof\cdot x));          -- date of order x
orqty [x: ponr, y: sic] := m2(orlines(x)\cdot y);  -- qty ordered in x of item y
data [x: supnr] := \pi1(supf\cdot x));           -- supplier data of x
types [x: supnr] := \pi2(supf\cdot x));        -- types supplied by x

-- constraints
-- \forall x: dom(sif) \exists type(x) \in dom(sif);
-- \forall x: dom(pof) \exists items(x) \subseteq dom(sif);
-- \forall x: dom(supf) \exists types(x) \subseteq dom(sif);
-- \forall x: dom(pof) \exists sup(x) \subseteq dom(supf);
-- \forall y: \exists x: dom(sif) \forall z: dates(x) \exists repls(x, y);
-- po(z) \subseteq dom(pof) \land x \in items(po(z)) \land
-- date(po(z)) \leq y \land report qty (x, po(z)) \leq orqty (po(z), x))];
-- each replenishment of a stock item has a purchase order responsible
-- for it; in this purchase order, a line must point to the item in
-- question; the replenishment date cannot exceed the order date and
-- the replenished quantities due to this order cannot
-- exceed the number of items ordered.
-- \forall x: dom(sif) \forall y: dates(x) \exists y \equiv call;
-- \forall x: dom(pof) \exists date(x) \equiv call;

-- system environment
proc report [x: str]; -- trigger to user

-- auxiliary functions (general)
convstr [x: T] :: str;
fupd [x: T->S, y: T->S] :=
  \{ u: dom(x) \cap dom(y) \mid if u \in dom(y) then y\cdot u else x\cdot u fi};
disjdom [x: T->S, y:T->U] := dom(x) \cap dom(y) = ();
contdom [x:T->S, y:T->U] := dom(x) \cap dom(y);

-- report messages
c1 := 'key conflict in sif';
c2 := 'key conflict in sif';
c3 := 'key conflict in supf';
c4 := 'key conflict in pof';
u1 := 'undefined key in sif';
u2 := 'undefined key in supf';
u3 := 'undefined key in supf';
u4 := 'undefined key in pof';
sh := 'the stock level of the specified item is too low';
id := 'illegal date'
ok := 'ok';
s1 := 'the following items have to be ordered: ';

-- processors
proc AddItemTypes [x: sitc -> sitd] ::= -- action 1
  if disjdom (x, sitf)
    then sitf <- fuzzp (sitf, x), report <= ok
    else report <= c1 fi;
proc UpdItemTypes [x: sitc -> sitd] ::= -- action 2
  if contdom (x, sitf)
    then sitf <- fuzzp (sitf, x), report <= ok
    else report <= u1 fi;
proc SeeItemTypes ::= -- action 3
  report <= convstr (sitf);
proc AddItems [x: sic -> sidat] ::= -- action 4
    if disjdom (x, sif)
    then if \( \pi(rg(x)) \in \text{dom}(sif) \)
        then sif <- fupd (sif, [\( t:\text{dom}(x) \mid (x \cdot t, \{\}) \)])
        report <= ok
    else report <= u1 fi
    else report <= c2 fi;
proc UpdItem [x: sic, y: sidat] ::= -- action 5
    if \( x \in \text{dom}(sif) \)
    then if \( \pi(y) \in \text{dom}(sif) \)
        then sif <- fupd (sif, [\( t:\text{dom}(x) \mid (y, \text{hist}(t)) \)])
        report <= ok
    else report <= u1 fi
    else report <= u2 fi;
proc SeeItems ::= -- action 6
    report <= convstr ([\( t:\text{dom}(sif) \mid \text{data}(t) \)])
proc AddSupp [x: supnr -> supdat] ::= -- action 7
    if disjdom (x, supf)
    then supf <- fupd (supf, [\( t:\text{dom}(x) \mid (x \cdot t, \{\}) \)])
    report <= ok
    else report <= c3 fi;
proc UpdSupp [x: supnr, y: supdat] ::= -- action 8
    if \( x \in \text{dom}(supf) \)
    then supf <- fupd (supf, [\( t:\text{dom}(x) \mid (y, \text{types}(t)) \)])
    report <= ok
    else report <= u3 fi;
proc SeeSupp [x: supnr] ::= -- action 9
    report <= convstr ([\( t:\text{dom}(supf) \mid \text{data}(t) \)])
proc AddSuppTypes [x: supnr, y: $sitc] ::= -- action 10
    if \( x \in \text{dom}(supf) \)
    then if \( y \in \text{dom}(sif) \)
        then supf <- fupd (supf, [\( t:\text{dom}(x) \mid (\text{data}(t), y \ U \text{types}(t)) \)])
        report <= ok
    else report <= u1 fi
    else report <= u3 fi;
proc SeeTypeSupp [x: sitc] ::= -- action 11
    report <= convstr ([\( t:\text{dom}(supf) \mid \text{data}(t) \)])
proc SetDate [x: date] ::= -- action 12
    report <= convstr ([\( t:\text{dom}(supf) \mid \text{data}(t) \)])
    if \( x > \text{cal} \)
    then \cal <= x, report <= ok
    else report <= id fi;
proc UPdStock [x: sic, y: $dr, z: $rep] ::= -- action 14
    if \( x \in \text{dom}(sif) \)
    then if \( \Sigma(\pi(y)) \leq \text{curqty}(x) + \Sigma(\pi(z)) \)
        then sif <- fupd (sif, [\( t:\text{dom}(x) \mid (\text{data}(t), \text{fupd}(\text{hist}(t), [s:(\text{cal}) \mid (\text{curqty(x)} - \Sigma(\pi(y)) + \Sigma(\pi(z))), \text{wdr}(t,s) U \ y, \text{repl}(t,s) U \ z)])\)])
        ShowShortage <= \{x\}
    else report <= shortage
    else report <= u2 fi;
proc RecordMan [x: sic -> qty] ::= -- action 15
    if contdom (x, sif)
    then sif <- fupd (sif, [\( t:\text{dom}(x) \mid (\text{data}(t), \text{fupd}(\text{hist}(t), [s:(\text{cal}) \mid (\text{curqty(x)} - \Sigma(\pi(y)) + \Sigma(\pi(z)), \text{wdr}(t,s) U \ y, \text{repl}(t,s) U \ z)])\)])
        ShowShortage <= \text{dom}(x)
    else report <= u2 fi;
proc SeeSohProfile [x: sic, y: date, z: date] ::= -- action 16
    report <=
    if \( x \in \text{dom}(sif) \)
    then convstr ([\( s: \{t: \text{dates}(x) \mid y \leq t \leq z \} \mid \text{qty}(x,s) \)])
    else u2 fi;
proc SeeSohType[x: date, y: sitc] ::=  -- action 17
  report <= convstr ([t: selcode(y)\ t qty(t,x)])
where selcode [x: sitc] ::= \{t: dom(sif) I type(t) = x\}
proc AddPurchOrder [x: ponr, y: supnr, z: pol] ::=  -- action 18
  if x \in dom(pof) then report <= convstr (\{t: \}
else if y \in dom(supf) then if contdom(z, sif) then pof -= fupd(pof, [t: (x)\ (cal, y), z])
else report <= ok
else report <= u2
else report <= u3 fi fi fi;
proc ShowShortage [x: $sic] ::=  -- action 19
  report <= if shortitems(x) = () then ok
else s1 & convstr (shortitems(x)) fi
where shortitems [x: $sic] ::= \{t: x I vs(t) \le rlev(t)\}
andwh vs [x:sic] ::=  -- virtual stock
  curqty(x) + \Sigma[t: orders(x) I orqty(t,x) - reporqty(x,t)]
proc DatePurch [x: date] ::=  -- action 20
  report <= convstr ([s: dom(pof) I date(s) = x]\ pof.s)
proc NumPurch [x: ponr] ::=  -- action 21
  report <= if x \in dom(pof) then convstr (pof.x)
else u4 fi;
proc WdrStock [x: sic, y: $wdr] ::=  -- action 22
  UpdStock <= \{x, y, ()\};
proc SeeWdrProfile [x: sic, y: date, z: date] ::=  -- action 23
  report <= if x \in dom(sif) then convstr ([s: \{t: dates(x) I y \le t \le z\} I wdr(s(x,s))]
else u2 fi;
proc ReplStock [x: sic, y: $repl] ::=  -- action 24
  UpdStock <= \{x, \}, y\};
proc SeeReplProfile [x: sic, y: date, z: date] ::=  -- action 25
  report <= if x \in dom(sif) then convstr ([s: \{t: dates(x) I y \le t \le z\} I repl(s(x,s))]
else u2 fi;
References

Formal specification and software development
Prentice-Hall, 1982

A framework for modeling of discrete dynamic systems
Proceedings TAIS conference Sophia Antipolis 1987
North-Holland, 1988

Specification case studies
Prentice-Hall, 1987

A formal model for system specification
to appear

The language EXSPECT
to appear

[6] IFIP Working Group 8.1 Conference (Computerized Assistance during the system life cycle)
Example A: Business analysis and system design specifications for an inventory control and purchasing system

A systematic approach to information systems development
Information systems 4 (1979)

[8] Peterson, J.L.
Petri net theory and the modeling of systems
Prentice-Hall, 1981

Software design using SADT
Auerbach Publishers Portfolio 35-05-03 (1977)

Structured development for real-time systems
Yourdon Press, 1985
1. Introduction

This example gives a step by step illustration of the specifications for a typical Inventory Control and Purchasing application. The documentation from the Business Analysis phase is aimed at users and is hence intended to be user friendly. The documentation from the System Design phase is aimed at those building the system. It is intended to be informative, but above all definitive.

In order to distinguish comment on rationale from parts of the analysis and design documentation, all comment is printed in italics.

2. Inventory Control and Purchasing

A small company maintains a stock of various items for its own use. For the purpose of this illustration, all stock is kept at one location. As and when necessary, the company orders new supplies from one of its suppliers. It therefore keeps track of the kinds of supplies available from each supplier and it also keeps track of its current stock on hand for each kind of item it needs.

3. Business Analysis steps

The steps in the Business Analysis phase are as follows:

1. Business activity analysis
2. Entity type identification
3. Entity type relationship analysis
4. Attribute analysis
5. Population analysis
6. Relationship cardinality analysis
7. Business activity to entity type cross reference
8. Feasability of design alternatives
3.1 Business Activity analysis

The top level activity is "Inventory Control and Purchasing". This can be categorized as "Acquisition of external resources". It can initially be broken down as follows:

1. Handling new types of stock
2. Ordering stock when required
3. Monitoring stock levels.

The first two of these can be broken down further as follows:

1.1 Categorization of new stock items
1.2 Investigation of possible suppliers
2.1 Select supplier
2.2 Send purchase order

This two level breakdown defines the area of interest for the Business Analysis. The hierarchical relationship between activities on one level and their subordinate activities on the next level down is given by a Business Activity numbering scheme as follows:

1. Handling new types of stock
   1.1 Categorization of new stock items
   1.2 Investigation of possible suppliers

2. Ordering stock when required
   2.1 Select supplier
   2.2 Send purchase order

3. Monitoring stock levels.

In practice, there would normally be several top level activities and in a large study they would be in more than one category. Furthermore, each top level activity covered might decompose into three or four levels instead of just two as illustrated here. It is also possible to indicate the hierarchical breakdown using a two column table in which the first column contains the name or number of a higher level activity and the other contains the name or number of one of its parts.

3.2 Entity type identification

The following entity types are relevant to the activity of Inventory Control. The sequence of these entity types has been chosen to aid comprehension when they are elaborated further. Each entity type is categorized in two ways, firstly according to its Entity Type Category and secondly according to its Dynamic Class.

<table>
<thead>
<tr>
<th>Entity type number</th>
<th>Entity type name</th>
<th>Entity type category</th>
<th>Dynamic class</th>
</tr>
</thead>
<tbody>
<tr>
<td>ET1</td>
<td>STOCK ITEM TYPE</td>
<td>Categorizing</td>
<td>&lt; 1 per year</td>
</tr>
<tr>
<td>ET2</td>
<td>STOCK ITEM</td>
<td>Fundamental</td>
<td>&lt; 1 per week</td>
</tr>
<tr>
<td>ET3</td>
<td>SUPPLIER</td>
<td>Fundamental</td>
<td>&lt; 1 per week</td>
</tr>
<tr>
<td>ET4</td>
<td>SUPPLIER OF STOCK ITEM</td>
<td>Cross ref</td>
<td>&lt; 1 per week</td>
</tr>
<tr>
<td>ET5</td>
<td>DATE</td>
<td>Fundamental</td>
<td>&lt; 1 per month</td>
</tr>
<tr>
<td>ET6</td>
<td>STOCK ON HAND</td>
<td>Event</td>
<td>&lt; 1 per hour</td>
</tr>
<tr>
<td>ET7</td>
<td>PURCHASE ORDER</td>
<td>Event</td>
<td>&lt; 1 per hour</td>
</tr>
<tr>
<td>ET8</td>
<td>PURCHASE ORDER LINE</td>
<td>Cross ref</td>
<td>&lt; 1 per hour</td>
</tr>
</tbody>
</table>

The Dynamic Class column places each entity type in one of eight frequency bands to give a rough indication of how often entities of that type are added or modified.

Normally, the list of Entity Types goes through several iterations before analysis is cut off.

3.3 Entity type relationship analysis

The Relationships between the eight Entity Types are shown in the data structure diagram on the following page. In this diagram, each rectangle represents an Entity Type and each arrow between two Entity Types represents a one to many relationship between those Entity Types.

The meaning of the one to many relationship is explained in terms of an example. The one chosen is the Relationship between SUPPLIER and PURCHASE ORDER. The Relationship means that each SUPPLIER has been sent zero one or more PURCHASE ORDERS and that each PURCHASE ORDER has been sent to one and only one SUPPLIER.

Other kinds of Relationship are possible in this and more complex examples, but it is chosen for simplicity to use only one kind of Relationship and one kind of arrow in this example.
Each Relationship involves two Entity Types. However, each Relationship is asymmetric and it is therefore necessary to distinguish between the two Entity Types. This is done by identifying one as independent (I) and the other as dependent (D). Each Relationship is given an identifying number, which corresponds to one of those in the data structure diagram.

<table>
<thead>
<tr>
<th>Relationship No</th>
<th>Relationship Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M1</td>
<td>Stock item in type</td>
</tr>
<tr>
<td>2</td>
<td>M1</td>
<td>Line in purchase order</td>
</tr>
<tr>
<td>3</td>
<td>M1</td>
<td>Date of purchase order</td>
</tr>
<tr>
<td>4</td>
<td>M1</td>
<td>Date of stock on hand</td>
</tr>
<tr>
<td>5</td>
<td>M1</td>
<td>Stock item reference on line</td>
</tr>
<tr>
<td>6</td>
<td>M1</td>
<td>Purchase order to supplier</td>
</tr>
<tr>
<td>7</td>
<td>M1</td>
<td>Stock on hand for stock</td>
</tr>
<tr>
<td>8</td>
<td>M1</td>
<td>Stock item available</td>
</tr>
<tr>
<td>9</td>
<td>M1</td>
<td>Supplier of stock item</td>
</tr>
</tbody>
</table>

3.4 Attribute analysis

The best way to present the results of the attribute analysis to users is by preparing one table for each entity type and representing each attribute as a column in a table. To clarify the meaning of the entity types and attributes, it is advisable to include some rows in the table which as far as possible should consist of "live" data. The user is used to thinking in these terms rather than in terms of abstractions such as entity types.

The following pages contain a description of each entity type and a table of examples for each entity type. In addition there is certain standard information about each entity type which needs to be documented as follows: The format for each entity type is as follows:

1. Description
2. Table of examples, indicating which attributes serve as unique identifiers of each entity type.
3. Estimated volume
4. Whether dynamic
5. Initial source
6. On-going responsibility for updating
7. Questions
The estimated volume indicates the number of entities of each type (i.e., the number of rows in the table). If the data is dynamic, then it indicates how many are created in a one year period.

The "whether dynamic" section indicates roughly how often the data is updated. (This corresponds to the "dynamic class" in section 3.2).

The "initial source" section indicates where in the company the first version of the data will come from. (This is intended to start the user thinking about this problem).

The "on-going authority for updating" indicates who or which organisation in the company will have the authority and hence the responsibility for making updates to this data. This information will be needed later for designing the security system (or access control system) needed to protect the data in the system.

The "questions" section allows the analyst to note issues relating to the entity type on which he has outstanding questions.

3.4.1 STOCK ITEM TYPE

1. Description.

This entity type is used to provide a simple categorization of STOCK ITEMS for budgeting and management control purposes.

2. Table of examples

<table>
<thead>
<tr>
<th>Stock item type code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Office Equipment (capital expense)</td>
</tr>
<tr>
<td>S</td>
<td>Stationery supplies</td>
</tr>
<tr>
<td>V</td>
<td>Vehicles (capital expense)</td>
</tr>
<tr>
<td>F</td>
<td>First aid supplies</td>
</tr>
<tr>
<td>K</td>
<td>Kitchen supplies</td>
</tr>
</tbody>
</table>

The stock item code must be unique.

3. Estimated volume

Less than 10.

4. Whether dynamic

Modified less than once per year

5. Initial source

Code adopted from existing manual system, but modified to ensure each STOCK ITEM TYPE is mutually exclusive.

6. On going updating authority

Financial director.
3.4.2 STOCK ITEM

1. Description

A STOCK ITEM is a class of article which is held in the stores and issued to members of the staff for consumption or use. A STOCK ITEM is not a physical object, but there may be many physical objects associated with one STOCK ITEM in the stores at any one time.

2. Table of examples

<table>
<thead>
<tr>
<th>Stock item type code</th>
<th>Stock item code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>5632</td>
<td>Compaq Plus computer</td>
</tr>
<tr>
<td>E</td>
<td>6234</td>
<td>Panasonic Printer</td>
</tr>
<tr>
<td>L</td>
<td>2357</td>
<td>Canon PC24 photocopier</td>
</tr>
<tr>
<td>S</td>
<td>3214</td>
<td>Xerox copying paper</td>
</tr>
<tr>
<td>F</td>
<td>2389</td>
<td>Parker fibre tip refill</td>
</tr>
</tbody>
</table>

Each STOCK ITEM has a unique Stock Item Code.

3. Estimated volume

Between 300 and 400

4. Dynamic class

About 50 new STOCK ITEMS are added every year.

5. Initial source

Existing manual system

6. On-going updating authority

Stores manager

3.4.3 SUPPLIER

1. Description

A SUPPLIER is a company who is able to sell the kind of STOCK ITEMS the company keeps in the stores. A SUPPLIER can be recorded as a SUPPLIER without having sold anything to the company.

2. Tables of examples

<table>
<thead>
<tr>
<th>Supplier number</th>
<th>Supplier name</th>
<th>Supplier address</th>
<th>Phone number</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>Rymans</td>
<td>23 South St, Walton</td>
<td>234567</td>
</tr>
<tr>
<td>32</td>
<td>Boots</td>
<td>12 High St, Weybridge</td>
<td>678912</td>
</tr>
<tr>
<td>12</td>
<td>Inmac</td>
<td>17 First St, London</td>
<td>1234566</td>
</tr>
<tr>
<td>13</td>
<td>Mayfair Micros</td>
<td>23 West St, Putney</td>
<td>2452456</td>
</tr>
</tbody>
</table>

Each Supplier Number must be unique.

3. Estimated volume

About 50 in the existing system

4. Whether dynamic

There are about 10 new Suppliers added to the list every year.

5. Initial source

Existing system

6. On-going updating authority

Head of purchasing

7. Questions

What happens when a supplier goes out of business?
3.4.4 SUPPLIER OF STOCK ITEM

1. Description

This cross reference indicates which STOCK ITEMS a SUPPLIER is able to supply and at the same time it indicates for each STOCK ITEM the SUPPLIERS from which it can be obtained.

2. Table of examples

<table>
<thead>
<tr>
<th>Supplier number</th>
<th>Stock item code</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>3214</td>
</tr>
<tr>
<td>32</td>
<td>2389</td>
</tr>
<tr>
<td>12</td>
<td>2214</td>
</tr>
<tr>
<td>13</td>
<td>5632</td>
</tr>
<tr>
<td>13</td>
<td>6234</td>
</tr>
<tr>
<td>43</td>
<td>1234</td>
</tr>
</tbody>
</table>

The combination of Supplier number and Stock item code must be unique.

3. Estimated volume

Each of 50 SUPPLIERS sells on average about 20 STOCK ITEMS of interest to the company, and this means the estimated volume is about 1000 and that each stock item can be obtained from between 2 and 3 SUPPLIERS.

4. Whether dynamic

Since the number of STOCK ITEMS increases by about 50 per year and each can be obtained from about 2 SUPPLIERS, the average growth is about 100 per year.

5. Initial source

The 'existing system gives the primary SUPPLIER of each STOCK ITEM and data about other SUPPLIERS will need to be built up.

6. On-going updating authority

Head of purchasing

7. Questions

Should information about each SUPPLIER'S price for the named STOCK ITEM be maintained? If so, should the validity date of the price also be recorded? Who will keep this information up to date?

3.4.5 DATE

1. Description

The entity type DATE is included to emphasize the importance of analysing carefully all time dependent data in any information system.

2. Table of examples

Any valid calendar date is a valid example of the entity type DATE. Most computerised systems record dates in the format year, month, date (e.g. 860305 instead of 5 March 1986) in order to make it easier to sort output on dates.

3. Estimated volume

It is assumed that the system will contain 3 years data at any time and the estimated volume is 3 x 365 = 1095 or 1000 as a working figure.

4. Whether dynamic

Since no dates need to be recorded in advance, there will be one date added every work day.

5. Initial source

The dates of purchase orders which are still open when the system goes live will need to be recorded.

6. On-going updating authority

The system will have its own internal calendar and dates will be generated as and when necessary.

7. Questions

Will it be necessary to ask questions from a terminal such as "Which purchase orders were issued on 3 Jan 85?"
3.4.6 STOCK ON HAND

1. Description

This entity type represents the quantity of stock of a given STOCK ITEM which is available in the stores on a given date and at a given time. The quantity is updated each time there is a withdrawal from the stores, each time goods are returned, and each time the stock level is replenished.

2. Table of examples

<table>
<thead>
<tr>
<th>Stock item code</th>
<th>Date</th>
<th>Time</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>5632 (Compaq Plus computer)</td>
<td>860305</td>
<td>14.45</td>
<td>3</td>
</tr>
<tr>
<td>6234 (Panasonic Printer)</td>
<td>860203</td>
<td>10.45</td>
<td>2</td>
</tr>
<tr>
<td>6235 (Panasonic Printer)</td>
<td>860401</td>
<td>8.45</td>
<td>1</td>
</tr>
<tr>
<td>2357 (Canon PC24 photocopier)</td>
<td>860704</td>
<td>9.30</td>
<td>1</td>
</tr>
<tr>
<td>3214 (Xerox copying paper)</td>
<td>861003</td>
<td>17.30</td>
<td>25</td>
</tr>
<tr>
<td>1234 (Parker fibre tip refill)</td>
<td>870304</td>
<td>9.40</td>
<td>237</td>
</tr>
<tr>
<td>2384 (Paracetemol)</td>
<td>870402</td>
<td>14.30</td>
<td>531</td>
</tr>
</tbody>
</table>

The combination of stock item code, date and time must be unique. There may be more than one movement for a given STOCK ITEM on any DATE.

3. Estimated volume

Each of 400 STOCK ITEMS has on average 1 movement per week and hence the number of occurrences of this entity type per year is about 20,000.

4. Whether dynamic

There are 400 movements per week, which is about 10 per working hour.

5. Initial source

Initial stock taking before new system goes live. Stock movements prior to that date will not be recorded in the new system.

6. On-going updating authority

Store room clerk

7. Questions

What about re-ordering levels?

3.4.7 PURCHASE ORDER

1. Description

A PURCHASE ORDER is issued to a SUPPLIER on a given DATE to order a new supply of one or more STOCK ITEMS.

2. Table of examples

<table>
<thead>
<tr>
<th>Purchase order no</th>
<th>Date</th>
<th>Supplier number</th>
</tr>
</thead>
<tbody>
<tr>
<td>74321</td>
<td>860406</td>
<td>12</td>
</tr>
<tr>
<td>74322</td>
<td>860406</td>
<td>32</td>
</tr>
<tr>
<td>74323</td>
<td>860408</td>
<td>13</td>
</tr>
<tr>
<td>74324</td>
<td>860408</td>
<td>12</td>
</tr>
</tbody>
</table>

Each Purchase Order number is unique.

3. Estimated volume

There are about 50 PURCHASE ORDERS issued every year.

4. Whether dynamic

Less than 1 per week.

5. Initial source

When the system goes live at the beginning of the Financial Year, all open purchase orders will be recorded in the new system.

6. On-going updating authority

Purchasing department.

7. Questions

What about re-ordering levels?
3.4.8 PURCHASE ORDER LINE

1. Description

This cross reference entity type indicates the STOCK ITEMS ordered on each PURCHASE ORDER and the quantity of each ordered.

2. Table of examples

<table>
<thead>
<tr>
<th>Purchase order line no.</th>
<th>Stock item number</th>
<th>Unit</th>
<th>Quantity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>74321</td>
<td>5632</td>
<td>3000</td>
<td>2</td>
<td>6000</td>
</tr>
<tr>
<td>74322</td>
<td>2234</td>
<td>250</td>
<td>4</td>
<td>1060</td>
</tr>
<tr>
<td>74323</td>
<td>2384</td>
<td>2</td>
<td>20</td>
<td>2800</td>
</tr>
</tbody>
</table>

3. Estimated volume

Each of the 50 PURCHASE ORDERS per year contains on average 4 PURCHASE ORDER LINES and hence the estimated volume is 200 per year.

4. Whether dynamic

Less than one per day.

5. Initial source

As for PURCHASE ORDERS.

6. On-going updating authority

As for PURCHASE ORDERS.

7. Questions

3.5 Population analysis

This example does not contain entity types which would make the kind of analysis required in this step meaningful.

3.6 Relationship cardinality analysis

The cardinality of a relationship is the average number of entities of one type which correspond to one entity of the other type. The cardinality of each relationship from step 3 (entity type relationship analysis) is shown in the following table.

<table>
<thead>
<tr>
<th>Relationship No</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>Stock item in type</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Line in purchase order</td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>Date of purchase order</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>Date of stock on hand</td>
</tr>
<tr>
<td>5</td>
<td>1.5</td>
<td>Stock item reference on line</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>Purchase order to supplier</td>
</tr>
<tr>
<td>7</td>
<td>150</td>
<td>Stock on hand for stock</td>
</tr>
<tr>
<td>8</td>
<td>2.5</td>
<td>Stock item available</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>Supplier of stock item</td>
</tr>
</tbody>
</table>

This table is one of the most important pieces of analysis with respect to the performance of the system to be designed. Although it shows only average cardinalities, the figures are accurate enough to pin-point potential problem areas and to take evasive action if necessary.

The above figures are predicated on the assumption that three years data will be held in the system at any one time. The cardinality of some of the relationships is proportional to the amount of time dependent data that needs to be held.

For example, the number of instances of STOCK ON HAND generated per year has been estimated at 20,000 (see section 3.4.6). This means that for each STOCK ITEM, there will be on average 150 movements every 3 years. This may or may not be a problem. If it is clear that any querying of the entity type STOCK ON HAND will always be time related (for example "how many withdrawals between 860401 and 860630?"), then there is no problem.

In general, any relationship with an average cardinality of more than 40 will have to be examined carefully when the data base is designed.
3.7 Step 7: Business activity to entity type cross reference.

The aim of this step is to check that the business activities specified in step 1 are consistent with the entity types from step 2. The reason that this step is not included earlier is that, in practice, steps 3 to 6 can cause a re-iteration of step 2 and even of step 1. Some analysts might, however, find it helpful to prepare this cross reference after completion of step 2.

The entity types are as follows:

<table>
<thead>
<tr>
<th>Entity type number</th>
<th>Entity type name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ET1</td>
<td>STOCK ITEM TYPE</td>
</tr>
<tr>
<td>ET2</td>
<td>STOCK ITEM</td>
</tr>
<tr>
<td>ET3</td>
<td>SUPPLIER</td>
</tr>
<tr>
<td>ET4</td>
<td>SUPPLIER OF STOCK ITEM</td>
</tr>
<tr>
<td>ET5</td>
<td>DATE</td>
</tr>
<tr>
<td>ET6</td>
<td>STOCK ON HAND</td>
</tr>
<tr>
<td>ET7</td>
<td>PURCHASE ORDER</td>
</tr>
<tr>
<td>ET8</td>
<td>PURCHASE ORDER LINE</td>
</tr>
</tbody>
</table>

The cross reference table indicates which Business Activities update each Entity Type and which retrieve information about it.

<table>
<thead>
<tr>
<th>Entity type</th>
<th>ET1</th>
<th>ET2</th>
<th>ET3</th>
<th>ET4</th>
<th>ET5</th>
<th>ET6</th>
<th>ET7</th>
<th>ET8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business activity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1 Categorisation of new stock items</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2 Investigation of possible suppliers</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1 Select supplier</td>
<td>R</td>
<td>U</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>2.2 Send purchase order</td>
<td>R</td>
<td>U</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Monitor stock levels</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This cross reference table is intended as an illustration of inconsistency in order to provide a basis for the following questions which would need to be asked of the user.

The following questions arise from this step:

1. Which activity creates STOCK ITEM TYPE?
2. Which activity creates STOCK ON HAND?
3. Which activity makes use of PURCHASE ORDER?

There are three ways of handling these questions. One is to revise the results of steps 1 to 7 in order to have a consistent analysis. The second way is to leave the problems to be solved in the System Design phase. The third is to redo steps 1 and 7 before going on to the analysis.

To answer these questions, a revised set of Business Activities is needed. These are as follows:

1. System start up
2. Handling new types of stock
   2.1 Categorization of new stock items
   2.2 Investigation of possible suppliers
3. Monitoring stock levels
   3.1 Keep stock on hand up to date
   3.2 Establish re-order levels
4. Ordering stock when required
   4.1 Select supplier
   4.2 Send purchase order
5. Purchase order follow up

The cross reference table between Business Activity and Entity Types is then as follows:

<table>
<thead>
<tr>
<th>Entity type</th>
<th>ET1</th>
<th>ET2</th>
<th>ET3</th>
<th>ET4</th>
<th>ET5</th>
<th>ET6</th>
<th>ET7</th>
<th>ET8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business activity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. System start-up</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1 Categorization of new stock items</td>
<td>R</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2 Investigation of possible suppliers</td>
<td>R</td>
<td></td>
<td>U</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1 Keep stock on hand up to date</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2 Establish re-order levels</td>
<td>R</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1 Select supplier</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2 Send purchase order</td>
<td>R</td>
<td>R</td>
<td>U</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Purchase order follow up</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This cross reference table now indicates that each Entity Type is updated by a Business Activity and furthermore each Entity Type is used by at least one Business Activity.
4. System Design steps

The steps in the System Design phase are as follows:

1. Entity type to table assignment
2. Attribute to column assignment
3. Table specification
4. Definition of referential constraints
5. Task definition
6. Specification of stepping stones
7. Task to menu assignment
8. Task to table cross reference
9. Assignment of tasks to access control classes

It is assumed that the questions raised in section 3.4.3, 3.4.4, 3.4.5, and 3.4.6 have been answered by the user and the designer is required to take these answers into account.

4.1 Step 1: Entity Type to Table assignment

It is felt to be advisable to make a number of changes to the results of the Business Analysis in order to provide a better System Design. The design consists of the following ten Tables.

<table>
<thead>
<tr>
<th>Table number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>STOCK ITEM TYPE</td>
</tr>
<tr>
<td>T2</td>
<td>STOCK ITEM</td>
</tr>
<tr>
<td>T3</td>
<td>SUPPLIER</td>
</tr>
<tr>
<td>T4</td>
<td>SUPPLIER OF STOCK ITEM TYPE</td>
</tr>
<tr>
<td>T5</td>
<td>CALENDAR DATE</td>
</tr>
<tr>
<td>T6</td>
<td>STOCK ON HAND</td>
</tr>
<tr>
<td>T7</td>
<td>PURCHASE ORDER</td>
</tr>
<tr>
<td>T8</td>
<td>PURCHASE ORDER ITEM</td>
</tr>
<tr>
<td>T9</td>
<td>STOCK WITHDRAWAL</td>
</tr>
<tr>
<td>T10</td>
<td>STOCK REPLENISHED</td>
</tr>
</tbody>
</table>

The changes will be clarified shortly. The data structure diagram in Figure A.3 shows the above ten Tables and how they are inter-dependent. (The rectangles in this diagram should now be interpreted as relational tables. Each arrow is to be interpreted as a referential constraint on the table pointed to. The constraint refers in each case to matchable columns in the table from which the arrow points.)

Figure A.3 Revised data structure diagram for System Design

The cross reference showing how the Entity Types from the Business Analysis phase have become Tables in the System Design phase is as follows:

<table>
<thead>
<tr>
<th>TABLE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTITY TYPE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The use of the figure "1" in the above cross reference, indicates in each case that one Entity Type has been converted into one Table. In more complex examples it is sometimes necessary to convert two or more Entity Types into one Table, or one Entity Type into two or more Tables.

It should be noted that this example indicates that one of the Entity Types, namely number 8, SUPPLIER OF STOCK ITEM has been omitted on the basis that it is impossible to keep such information up to date. Three Tables not resulting from the Business Analysis are included as follows:

- **4. SUPPLIER OF STOCK ITEM TYPE**
- **9. STOCK WITHDRAWAL**
- **10. STOCK REPLENISHED**

### 4.2 Step 2: Attribute to Column assignment

The following list of Column Names consists of the Attribute Names from step 4 of the Business Analysis phase plus others necessitated by the introduction of the three Tables which are not from the results of the Business Analysis stage.

- STOCK CODE
- STOCK ITEM NAME
- CONSIGNMENT
- DELIVERY
- ISSUE
- PRICE
- SUPPLIER NUMBER
- SUPPLIER NAME
- STOCK ITEM TYPE CODE
- STOCK ITEM TYPE DESCRIPTION
- DATE OF PURCHASE
- DATE OF DELIVERY
- PURCHASE ORDER NUMBER
- PURCHASE ORDER ITEM NUMBER

### 4.3 Step 3: Table specification

Each Column must belong to a Table. This means that each of the above Column Names should be assignable to one of the ten Tables already identified. It may transpire that some of these Column Names cannot be assigned to a Table. The following cross reference table indicates the initial assignments.

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock Code</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stock Item Name</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delivery</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Supplier Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supplier Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stock Item Type Code</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stock Item Type Description</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date of Purchase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date of Delivery</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date of Withdrawal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Purchase Order Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Purchase Order Item Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Quantity of Stock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Date</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

It is normally not useful to prepare a detailed cross reference between Attributes and Columns.
4.4 Step 4: Constraint Definition

There should be at least one uniqueness Constraint defined for each of the ten Tables. Each such Constraint is expressed either for a single Column or for a Column Group of that Table. The uniqueness Constraints for the ten Tables are as follows:

<table>
<thead>
<tr>
<th>Table</th>
<th>Constraint Type</th>
<th>Column or Column Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOCK ITEM TYPE</td>
<td>R</td>
<td>Stock Item Type Code</td>
</tr>
<tr>
<td>STOCK ITEM</td>
<td>R</td>
<td>Stock Item Code</td>
</tr>
<tr>
<td>SUPPLIER</td>
<td>R</td>
<td>Supplier Number</td>
</tr>
<tr>
<td>SUPPLIER OF STOCK ITEM</td>
<td>R</td>
<td>Stock Item Type Code</td>
</tr>
<tr>
<td>TYPE</td>
<td>R</td>
<td>Supplier Number</td>
</tr>
<tr>
<td>CALENDAR DATE</td>
<td>R</td>
<td>Stock Item Type Code</td>
</tr>
<tr>
<td>STOCK ON HAND</td>
<td>R</td>
<td>Stock Item Code</td>
</tr>
<tr>
<td>Stocktaking Date</td>
<td>R</td>
<td>Date</td>
</tr>
<tr>
<td>PURCHASE ORDER</td>
<td>R</td>
<td>Purchase Order Number</td>
</tr>
<tr>
<td>PURCHASE ORDER ITEM</td>
<td>R</td>
<td>Purchase Order Number</td>
</tr>
<tr>
<td>Stock Item Code</td>
<td>R</td>
<td>Purchase Order Item Number</td>
</tr>
<tr>
<td>STOCK WITHDRAWAL</td>
<td>R</td>
<td>Date of Withdrawal</td>
</tr>
<tr>
<td>Withdrawal Number</td>
<td>R</td>
<td>Date of Delivery</td>
</tr>
<tr>
<td>STOCK REPLENISHED</td>
<td>R</td>
<td>Stock Item Code</td>
</tr>
<tr>
<td>Date of Delivery</td>
<td>R</td>
<td>Replenishment Number</td>
</tr>
</tbody>
</table>

Since there is only one Uniqueness Constraint on each of the 10 Tables, the constraint can also be referred to as a "primary key".

As indicated, each of the fourteen arrows depicted in the data structure diagram represents a referential Constraint. Each of these is a Constraint on the values of one or more Columns in one of the Tables.

The Columns to which the Constraints apply are always "matchable" with the values of Columns in the other Table at the tail of the arrow. The Columns in the Table referred to are the Primary Key for that Table.

The following table indicates the Table and the constrained Columns for each of the fourteen Constraints:

It should be noted that the first 13 of the above list of referential Constraints are based on individual Columns. The last one is based on a Column Group. All 14 referential Constraints are based on value equality.

As an example, the Constraint on Table 8 (PURCHASE ORDER ITEM) implies that the value in the Column called Purchase Order Number in the Table PURCHASE ORDER ITEM must correspond to an extant value in the Column called Purchase Order Number, but in Table 7 (PURCHASE ORDER).

It should also be noted that there are no referential Constraints on Tables 1, 4 and 5. It will be seen in the structure diagram these three Tables, namely:

1. STOCK ITEM TYPE
2. SUPPLIER
3. CALENDAR DATE

are not dependent on any other Tables.
4.5 Step 5: Task Definition

The following design decisions are taken at this point. It is assumed that the Tasks will be defined by first examining each of the ten Tables and deciding which of the following four Task Types are needed:

- Add
- Modify
- Delete
- Retrieve

The results of this examination is shown in the following cross reference which identifies 25 different Tasks.

It should be noted that the two Tables PURCHASE ORDER and PURCHASE ORDER ITEM are processed by the same TASKS on the basis that every PURCHASE ORDER has at least one PURCHASE ORDER ITEM. It is not meaningful to add a PURCHASE ORDER without adding at least one PURCHASE ORDER ITEM. Furthermore, any retrieval Task which accesses one of these two Tables will also need to access the other.

<table>
<thead>
<tr>
<th>Table no</th>
<th>Description</th>
<th>Add</th>
<th>Delete</th>
<th>Modify</th>
<th>Retrieve</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STOCK ITEM TYPE</td>
<td>1</td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>STOCK ITEM</td>
<td>4</td>
<td></td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>SUPPLIER</td>
<td>7</td>
<td></td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>SUPPLIER OF STOCK</td>
<td>10</td>
<td></td>
<td>11, 12</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>DATE</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>STOCK ON HAND</td>
<td>14</td>
<td></td>
<td>15</td>
<td>16, 17, 18</td>
</tr>
<tr>
<td>7</td>
<td>PURCHASE ORDER</td>
<td>19</td>
<td></td>
<td>20, 21</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>PURCHASE ORDER ITEM</td>
<td>19</td>
<td></td>
<td>20, 21</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>STOCK WITHDRAWAL</td>
<td>22</td>
<td></td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>STOCK REPLENISHED</td>
<td>24</td>
<td></td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>

4.6 Step 6: Specification of stepping stones between tables

In order to assess the need for stepping stones, it is first necessary to update the results of the Cardinality Analysis step from the Business Analysis phase. The revised analysis is presented in the annotated data structure diagram in Figure A.4.

As in Figure A.2, each rectangle contains an estimate in the bottom right hand corner of the number of rows expected in that Table. The number in the top left hand corner is the Table number. The cardinality of the relationship (or in design terms more correctly the constraint) is included in the diagram for each of the 14 referential Constraints.
Figure A.4 Cardinalities for System Design tables and constraints

The highest cardinality shown appears to be 50 and it occurs twice. The two instances are the following:

1. From STOCK ITEM to SUPPLIER OF STOCK ITEM TYPE
2. From STOCK ITEM to STOCK ON HAND

A cardinality of 50 is not likely to cause tremendous problems and hence no stepping stones need to be introduced.

4.7 Step 7: Task to menu assignments

Each of the 25 Tasks identified in Step 5 now needs to be assigned to a menu. It is proposed to have a two level menu structure, with one top level and four second level menus. The starting point for designing the menu structure is the revised breakdown of Business Activities from the step 8 in the Business Activity phase.

1. System start up
2. Handling new types of stock
   2.1 Categorization of new stock items
   2.2 Investigation of possible suppliers
3. Monitoring stock levels
   3.1 Keeping stock on hand up to date
   3.2 Establishing re-order levels
4. Ordering stock when required
   4.1 Select supplier
   4.2 Send purchase order
5. Purchase order follow up

On the basis of this breakdown, it is proposed to design the top level menu as follows:

XYZ COMPANY'S INVENTORY CONTROL SYSTEM

TOP LEVEL MENU

1. System start up
2. Handling new types of stock
3. Monitoring stock levels
4. Purchase order processing and follow-up

Indicate choice by typing appropriate integer

Of the 25 Tasks from Step 5, 23 are initiatable from a terminal. Each of these can now be assigned to one of the four second level menus. The assignment is shown in the annotated version of the cross reference table from Step 5 as follows. An S indicates that the Task is system initiated.
Table 1

<table>
<thead>
<tr>
<th>Description</th>
<th>Add</th>
<th>Delete</th>
<th>Modify</th>
<th>Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOCK ITEM TYPE</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>STOCK ITEM</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>SUPPLIER</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>SUPPLIER OF STOCK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITEM TYPE</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>DATE</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>STOCK ON HAND</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>PURCHASE ORDER</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>PURCHASE ORDER ITEM</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>STOCK WITHDRAWAL</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>STOCK REPLENISHED</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
</tr>
</tbody>
</table>

The integer to the right of each of the four rightmost columns indicates the menu to which the Task has been assigned.

As a result of this the four second order menus now appear as follows. The numbers against each Task correspond to the Task numbers assigned in step 5. This is for documentation purposes only and the system should number its own entries when it generates a menu for a user in a given Access Class.

---

### XYZ COMPANY'S INVENTORY CONTROL SYSTEM

#### MENU FOR SYSTEM START-UP

1. Add one or more Stock Item Types
2. Modify one or more existing Stock Item Types
3. Display available Stock Item Types
4. Add initial list of Stock Items
5. Add initial list of suppliers
6. Add list of Stock Item Types carried by a supplier
7. Enter range of dates for which records will be kept

Indicate choice by typing appropriate integer__

---

### XYZ COMPANY'S INVENTORY CONTROL SYSTEM

#### MENU FOR NEW TYPES OF STOCK

1. Modify an existing stock item
2. Display list of stock items
3. Modify an existing supplier
4. Display list of suppliers

Indicate choice by typing appropriate integer__

---

### XYZ COMPANY'S INVENTORY CONTROL SYSTEM

#### MENU FOR MONITORING STOCK LEVELS

15. Manual recording of stock on hand
16. Display stock on hand profile for a given stock item between given dates
17. Display stock on hand on a given date for all stock items of a given type
18. Record a stock withdrawal
19. Display list of stock withdrawals for a given stock item between two dates
20. Record stock replenished
21. Display list of replenishments for a given stock item between two dates

Indicate choice by typing appropriate integer__

---

A-28
XYZ COMPANY'S INVENTORY CONTROL SYSTEM

MENU FOR PURCHASE ORDER PROCESSING AND FOLLOW UP

11. Display all suppliers of a given type of stock
12. Display all types of stock available from a given supplier
18. Generate a purchase order manually
20. Display all purchase orders issued on a given date
21. Display a purchase order with a given number

It should be noted that Tasks 14 and 19 do not appear on any of these menus as they are initiated by the system automatically. Task 14 is an automatic updating of the Stock on Hand when there is a withdrawal or a replenishment. Task 19 is an automatic creation of a Purchase Order when the Stock on Hand drops below the Re-order level for that Stock Item.

4.8 Step 8: Assignment of Tasks to Access Control Classes

For the purposes of Access Control, the potential users are grouped into the following mutually exclusive classes.

A. Data administrator
B. Purchasing manager
C. Purchase department staff
D. Warehouse personnel

Each of the 23 Tasks initiatable from a terminal is assigned to one or more of these four classes. The assignments are shown in the following cross reference table.

<table>
<thead>
<tr>
<th>Task</th>
<th>Access Control Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A Y</td>
</tr>
<tr>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>Y Y</td>
</tr>
<tr>
<td>4</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>Y Y</td>
</tr>
<tr>
<td>7</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>Y</td>
</tr>
<tr>
<td>9</td>
<td>Y Y</td>
</tr>
<tr>
<td>10</td>
<td>Y Y</td>
</tr>
<tr>
<td>11</td>
<td>Y</td>
</tr>
<tr>
<td>12</td>
<td>Y</td>
</tr>
<tr>
<td>13</td>
<td>Y</td>
</tr>
<tr>
<td>14</td>
<td>Y</td>
</tr>
<tr>
<td>15</td>
<td>Y</td>
</tr>
<tr>
<td>16</td>
<td>Y Y</td>
</tr>
<tr>
<td>17</td>
<td>Y Y</td>
</tr>
<tr>
<td>18</td>
<td>Y</td>
</tr>
<tr>
<td>19</td>
<td>Y</td>
</tr>
<tr>
<td>20</td>
<td>Y Y</td>
</tr>
<tr>
<td>21</td>
<td>Y Y</td>
</tr>
<tr>
<td>22</td>
<td>Y</td>
</tr>
<tr>
<td>23</td>
<td>Y</td>
</tr>
<tr>
<td>24</td>
<td>Y</td>
</tr>
<tr>
<td>25</td>
<td>Y Y</td>
</tr>
</tbody>
</table>

It should be noted that all of the retrieval tasks are regarded as being available to all users, because none of the information in the system is felt to be confidential. The two system initiated tasks are included in the above table for completeness.
4.9 Step 9: Task to table cross references

The following cross reference table illustrates Task Uses Table with the categorization of each Table used by a Task in terms of its Role in the information path.

<table>
<thead>
<tr>
<th>Table</th>
<th>Task</th>
<th>STARTING POINT</th>
<th>INTERMEDIATE POINT</th>
<th>TARGET POINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>iStock</td>
<td>iStock</td>
<td>iStock Item Type</td>
<td>STARTING POINT</td>
<td>Intermediate Point</td>
</tr>
<tr>
<td>Supplier</td>
<td>Supplier</td>
<td>Supplier</td>
<td>STARTING POINT</td>
<td>Intermediate Point</td>
</tr>
<tr>
<td>Item Type</td>
<td>Item Type</td>
<td>Item Type</td>
<td>STARTING POINT</td>
<td>Intermediate Point</td>
</tr>
<tr>
<td>Item</td>
<td>Item</td>
<td>Item</td>
<td>STARTING POINT</td>
<td>Intermediate Point</td>
</tr>
<tr>
<td>Item Type</td>
<td>Item Type</td>
<td>Item Type</td>
<td>STARTING POINT</td>
<td>Intermediate Point</td>
</tr>
<tr>
<td>iDate</td>
<td>iDate</td>
<td>iDate</td>
<td>STARTING POINT</td>
<td>Intermediate Point</td>
</tr>
<tr>
<td>iStock On Hand</td>
<td>iStock On Hand</td>
<td>iStock On Hand</td>
<td>STARTING POINT</td>
<td>Intermediate Point</td>
</tr>
<tr>
<td>iPurchase</td>
<td>iPurchase</td>
<td>iPurchase</td>
<td>STARTING POINT</td>
<td>Intermediate Point</td>
</tr>
<tr>
<td>iStock Withdrawal</td>
<td>iStock Withdrawal</td>
<td>iStock Withdrawal</td>
<td>STARTING POINT</td>
<td>Intermediate Point</td>
</tr>
<tr>
<td>iStock Replenishes</td>
<td>iStock Replenishes</td>
<td>iStock Replenishes</td>
<td>STARTING POINT</td>
<td>Intermediate Point</td>
</tr>
</tbody>
</table>

The codes used in this cross reference table are as follows:

**TASK TYPE**

- S Select
- A Add
- M Modify
- D Delete

**ROLE**

- S Starting point
- P Retrieval reference
- T Target
- V Validation reference
- M Intermediate point
- F Inflection point

**HOMING IN OR OUT OF THE BLUE**

- B Out of the blue
- A Homing in

This table provides a compact and highly codified definition of the 25 Tasks and of the path to be used by each Task from its starting point or points to its target or targets. It shows which Tables each Task needs to access. The sequence is defined in terms of the starting points, possible intermediate points and tables referenced, through to an end point (target) for each path.

As an example, Task 15 can be described in detail. It accesses the following four Tables:

**Starting point:** iDate, iStock Item Type
**Intermediate point:** iStock Item Type
**Target point:** iStock On Hand.

This Task homes in on a Stock Item from Stock Item Type. The terminal user specifies a Date (the system takes the date from the calendar clock). A row in the Stock On Hand table is then modified.

This example does not cover all the aspects of the design of Tasks. Some Tasks, such as 22 and 24, will need to perform numeric computation in order to modify the content of the Stock On Hand Table.

It should be noted that the content of this cross reference table could also be represented as a series of data flow diagrams in which some or all of the Tables are represented as "data stores".

A-33
In this series appeared:

<table>
<thead>
<tr>
<th>No.</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>85/01</td>
<td>R.H. Mak</td>
<td>The formal specification and derivation of CMOS-circuits</td>
</tr>
<tr>
<td>85/02</td>
<td>W.M.C.J. van Overveld</td>
<td>On arithmetic operations with M-out-of-N-codes</td>
</tr>
<tr>
<td>85/03</td>
<td>W.J.M. Lemmens</td>
<td>Use of a computer for evaluation of flow films</td>
</tr>
<tr>
<td>85/04</td>
<td>T. Verhoeff, H.M.J.L. Schols</td>
<td>Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate</td>
</tr>
<tr>
<td>86/01</td>
<td>R. Koymans</td>
<td>Specifying message passing and real-time systems</td>
</tr>
<tr>
<td>86/02</td>
<td>G.A. Bussing, K.M. van Hee, M. Voorhoeve</td>
<td>ELISA, A language for formal specifications of information systems</td>
</tr>
<tr>
<td>86/03</td>
<td>Rob Hoogerwoord</td>
<td>Some reflections on the implementation of trace structures</td>
</tr>
<tr>
<td>86/04</td>
<td>G.J. Houben, J. Paredaens, K.M. van Hee</td>
<td>The partition of an information system in several parallel systems</td>
</tr>
<tr>
<td>86/05</td>
<td>Jan L.G. Dietz, Kees M. van Hee</td>
<td>A framework for the conceptual modeling of discrete dynamic systems</td>
</tr>
<tr>
<td>86/06</td>
<td>Tom Verhoeff</td>
<td>Nondeterminism and divergence created by concealment in CSP</td>
</tr>
<tr>
<td>86/07</td>
<td>R. Gerth, L. Shira</td>
<td>On proving communication closedness of distributed layers</td>
</tr>
<tr>
<td>Year</td>
<td>Author(s)</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>86/08</td>
<td>R. Koymans</td>
<td>Compositional semantics for real-time distributed computing (Inf. &amp; Control 1987)</td>
</tr>
<tr>
<td></td>
<td>R.K. Shyamasundar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W.P. de Roever</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R. Gerth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S. Arum Kumar</td>
<td></td>
</tr>
<tr>
<td>86/09</td>
<td>C. Huizing</td>
<td>Full abstraction of a real-time denotational semantics for an OCCAM-like language</td>
</tr>
<tr>
<td></td>
<td>R. Gerth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W.P. de Roever</td>
<td></td>
</tr>
<tr>
<td>86/10</td>
<td>J. Hooman</td>
<td>A compositional proof theory for real-time distributed message passing</td>
</tr>
<tr>
<td>86/11</td>
<td>W.P. de Roever</td>
<td>Questions to Robin Milner - A responders commentary (IFIP86)</td>
</tr>
<tr>
<td>86/12</td>
<td>A. Boucher</td>
<td>A timed failures model for extended communicating processes</td>
</tr>
<tr>
<td></td>
<td>R. Gerth</td>
<td></td>
</tr>
<tr>
<td>86/13</td>
<td>R. Gerth</td>
<td>Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)</td>
</tr>
<tr>
<td></td>
<td>W.P. de Roever</td>
<td></td>
</tr>
<tr>
<td>86/14</td>
<td>R. Koymans</td>
<td>Specifying passing systems requires extending temporal logic</td>
</tr>
<tr>
<td>87/01</td>
<td>R. Gerth</td>
<td>On the existence of a sound and complete axiomatizations of the monitor concept</td>
</tr>
<tr>
<td>87/02</td>
<td>Simon J. Klaver</td>
<td>Federatieve Databases</td>
</tr>
<tr>
<td></td>
<td>Chris F.M. Verbeme</td>
<td></td>
</tr>
<tr>
<td>87/03</td>
<td>G.J. Houben</td>
<td>A formal approach to distributed information systems</td>
</tr>
<tr>
<td></td>
<td>J. Paredaens</td>
<td></td>
</tr>
<tr>
<td>87/04</td>
<td>T. Verhoeff</td>
<td>Delay-insensitive codes - An overview</td>
</tr>
<tr>
<td>87/05</td>
<td>R. Kuiper</td>
<td>Enforcing non-determinism via linear time temporal logic specification</td>
</tr>
<tr>
<td>Year</td>
<td>Author(s)</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>87/06</td>
<td>R. Koymans</td>
<td>Temporele logica specificatie van message passing en real-time systemen (in Dutch)</td>
</tr>
<tr>
<td>87/07</td>
<td>R. Koymans</td>
<td>Specifying message passing and real-time systems with real-time temporal logic</td>
</tr>
<tr>
<td>87/08</td>
<td>H.M.J.L. Schols</td>
<td>The maximum number of states after projection</td>
</tr>
<tr>
<td>87/10</td>
<td>T. Verhoeff</td>
<td>Three families of maximally nondeterministic automata</td>
</tr>
<tr>
<td>87/11</td>
<td>P. Lemmens</td>
<td>Eldorado ins and outs. Specifications of a database management toolkit according to the functional model</td>
</tr>
<tr>
<td>87/12</td>
<td>K.M. van Hee, A. Lapinski</td>
<td>OR and AI approaches to decision support systems</td>
</tr>
<tr>
<td>87/13</td>
<td>J. van der Woude</td>
<td>Playing with patterns, searching for strings</td>
</tr>
<tr>
<td>87/14</td>
<td>J. Hooman</td>
<td>A compositional proof system for an occam-like real-time language</td>
</tr>
<tr>
<td>87/15</td>
<td>G. Huizing, R. Gerth, W.P. de Roever</td>
<td>A compositional semantics for statecharts</td>
</tr>
<tr>
<td>87/16</td>
<td>H.M.M. ten Eikelder, J.C.F. Wilmont</td>
<td>Normal forms for a class of formulas</td>
</tr>
<tr>
<td>87/17</td>
<td>K.M. van Hee, G.J. Houben, J.L.G. Dietz</td>
<td>Modelling of discrete dynamic systems framework and examples</td>
</tr>
</tbody>
</table>
An integer algorithm for rendering curved surfaces

Optimalisering van file allocatie in gedistribueerde database systemen

The $R^2$-Algebra: An extension of an algebra for nested relations

Fully abstract denotational semantics for concurrent PROLOG

A Parallel Program That Generates the Möbius Sequence

Executable Specification for Information Systems

Settling a Question about Pythagorean Triples