The application controller concept: a first experience

P. Stecher* and Volker Allenstein†

The application controller concept addresses DPS/end user interface problems in applications of a stable and repetitive nature. These interface problems arise both during development and operation of an application. To tackle these problems the application controller concept proposes a methodology for the application development process which is based on a state networking technique. One main aim of this methodology is to involve the end user in the definition and design of his application to a high extent. This paper presents the experience gained with the methodology in the course of a major application development project at Lufthansa. The application controller concept is also a proposal for a software system which supports the actual operation of the application at run time. The last two sections of the paper will elaborate on some of the properties of this software system.

(Received April 1980)

The Application Controller (AC) concept (Stecher, 1977; 1978) consists of

(*a*) a methodology to structure and develop applications of a stable and repetitive nature

(*b*) a proposal for a software system which would support the development of the application *and* control the operation of an application at run time.

In the paper we describe how we applied the methodology of the AC concept in the project FAS (Frachtabfertigungs und informationssystem = cargo handling and information system) at Lufthansa. The basic principle of this methodology is to structure an application by state networks in a way which is close to the user's perception of the application. (As a matter of fact, one author of this paper is not a DP professional, but an end user.)

First we introduce in general terms the application of cargo handling, then we will present some facts of the project FAS. After we have defined the state networking technique in more detail we will describe its use during the phases 'basic design', 'detailed design' and 'programming and testing' of the application development process. Finally we deal with the AC as a software system.

1. The application FAS

In 1981 a new cargo base called 'Cargo Base West' is planned to be operating at Lufthansa in Frankfurt. The FAS is intended to provide part of the required data processing support. **Fig. 1** shows an overview of the cargo handling application: Cargo is received from customers or inbound flights at the transit station (cargo arrival). After the essential data of the cargo has been recorded the cargo is stored in warehouse. Documents pertaining to the cargo are recorded separately. Subsequently the document handling process starts:

(*a*) cargo which has arrived at its destination is assigned to customers

(*b*) cargo which has not arrived at its destination is assigned to outbound flights.

In the cargo departure section shipments are removed from warehouses on demand to be delivered to customers or to be transferred to outbound flights to which they were previously assigned.

Fig. 2 depicts some of the items of the cargo handling process. This figure also shows the relationship which may exist between these items: In the centre of the cargo handling we find the shipment (A). This shipment is defined by its airwaybill (AWB). Shipments can be handled loose or in load units (2). Example of load units (B) are containers, palets and igloos. Shipments can be loaded on aircraft loose (1) or on load units (3). They can be stored in the warehouse loose (4) or on load units (5). Transport frames (C) are used in automated warehouses. On these transport frames shipments are carried loose (6) or in load units (7) to their locations (8).

The FAS is going to support essential functions of the cargo handling system (**Fig. 3**): As part of the document handling of cargo the airwaybill data is recorded (1). Shipments are assigned to outbound flights (2). After the departure of the assigned outbound flights, or after the delivery to customers shipments are terminated (3; 4). As part of the physical handling of cargo, shipments are recorded (5) and related to their document data (6). The warehouse entry and exit messages (7; 8) are received and evaluated in the FAS.

In addition to these functions the FAS provides information to the user through CRT displays and printouts. In the following paragraphs we will explore the physical handling of cargo, in particular the recording of physical data of cargo. We focus on the function 'Recording load unit/breakdown'.

Recording load unit/breakdown

A load unit of the type 'breakdown' arrives by an inbound flight and may contain shipments with various destinations. Before this load unit is broken down and the shipments are separated according to their destinations, the load unit data is recorded and the load unit itself is stored temporarily in an automated warehouse. **Fig. 4** shows the screen layout for this function. The data elements HR-Nr., LE-Nr. and Flug-Nr. are mandatory: HR-Nr. is the number of the transport frame on which the load unit is to be carried; LE-Nr. is the number of the load unit and Flug-Nr. is the number of the inbound flight. The remaining data elements are optional and describe the load unit in more detail.

The function 'Recording load unit/breakdown' will be used in this paper to describe how we applied the state networking technique in the FAS. Before doing this let us consider some facts of the FAS project.

2. The FAS project

The FAS project was started in October 1977 and will be completed in the middle of 1980. Currently the project team consists of seven end users and 12 DP professionals.

*IBM Deutschland GmbH, Abraham-Lincoln-Str. 26, D-6200 Wiesbaden, West Germany, now at IBM Europe, Dept. No. 3433, Tour Franklin, F-92081, Paris La Défense, France.

†Deutsche Lufthansa AG, Lufthansa Basis, Abteilung FRA OG 2, Flughafen, D-6000 Frankfurt, West Germany.

CCC-0010-4620/81/0024-0097 $05.00

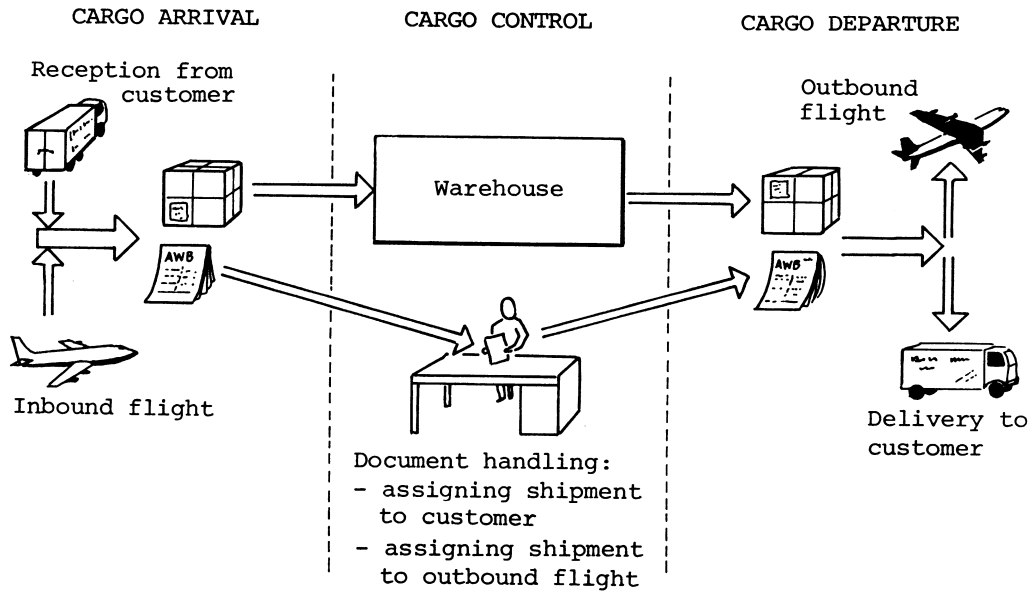

Fig. 1 Overview of a cargo handling system

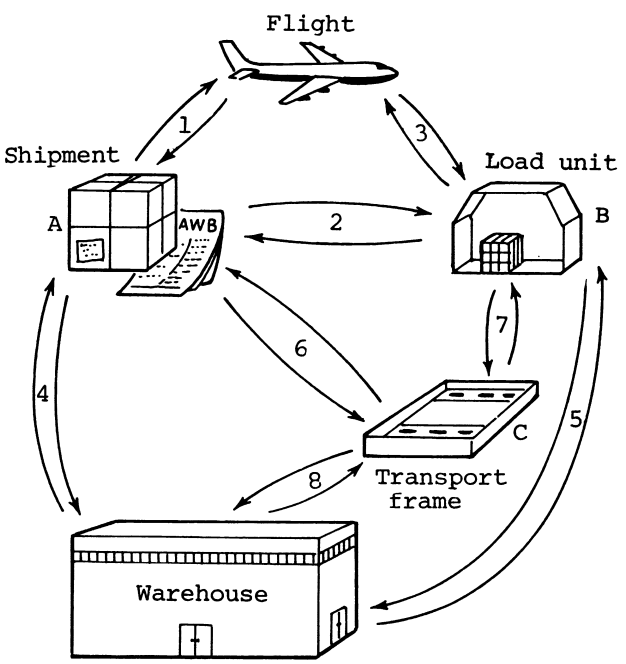

Fig. 2 Cargo items and their relationships

The data base/datacommunication system is IMS/VS (IBM, 1978). The programs are developed with COBOL and TSO-SPF (IBM, 1977). The automated warehouse control in the 'Cargo Base West' is done by a process control system and is separated from the FAS. However the FAS is in continuous dialogue with this system.

A principle of the FAS project, which was formulated at project conception, is the close and continuous co-operation between the end user and the DP department. This principle has been incorporated by the following procedures:

1. Common work locations.
2. Mixed task forces.
3. The total project was split into distinct phases. Each phase uses the results of the preceding phase and develops them further. In *phase* 1 'Requirement definition' the objectives and the scope of the FAS are specified. In *phase* 2 'Basic design' we define each of the FAS functions at a crude level, and their interactions, and we describe input and ouput formats and data elements. Data bases are designed and the hardware configuration is finalised.

   Phase 3 'Detailed design' consists of detailing the functions down to the programming level. *Phase* 4 comprises programming and testing. Before cutover the functions are integrated into one system and tested as a whole.

   When each phase has been completed the results are inspected by committees. These committees are made up of end users and DP professionals. At the moment the project is in phase 4.
4. Test cases are developed and documented by the end user department which also analyses the test results.
5. The methods and documention tools have been selected for the project with their user acceptability in mind:
   (*a*) the documentation method for the whole project is HIPO (Hierarchical Input-Process-Output; IBM, 1974), its supporting program is HIPO-draw (IBM, 1).
   (*b*) the state networking technique is the development method in part of the FAS.
   This method is the subject of the following sections.

3. The state networking technique

Because of the number of item relationships the FAS can be considered a complex system. The main items are

(*a*) transport frame (FM)
(*b*) load unit (LU)
(*c*) airwaybill (AWB)
(*d*) flight (FL)
(*e*) location (LO)

During the operation of the FAS these items relate to one another. Within the system the items are represented by data bases, the item relationships by data base relationships. While designing the application we soon realised that a particular function can be specified only if it is known what the other functions do with respect to item connection and disconnection. Since this statement is valid for nearly all functions we decided to analyse first those activities of each function which connect or disconnect items. Only after this has been done

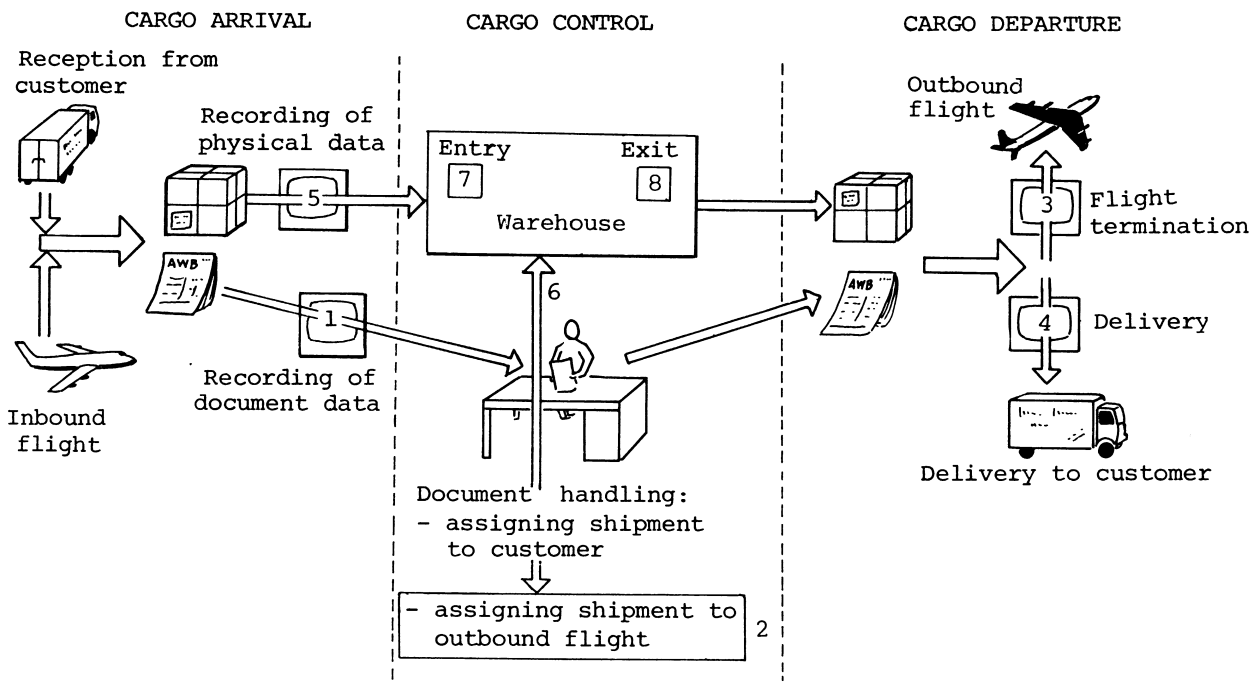

Fig. 3 FAS functions

would we consider the remaining activities of each function. The method with which we define the connection and disconnection of items is the state networking technique.

The basic elements of this technique are activity and state. A state expresses the result of an activity, in Fig. 3 for instance, the statement 'the physical data of the shipment has been recorded' describes a state of the activity 'recording of physical data'. The statement 'the shipment has been stored' describes a state of the activity 'warehouse entry'. The logic of the application is such that a shipment assumes the state 'stored' only after the state 'recorded' has been reached. The statement 'the shipment has been assigned to an outbound flight' describes a state of the activity 'assigning shipment to outbound flight'.

On the one hand a state expresses the result of an activity, on the other hand a certain state must have been reached before a certain activity can be executed. This dual relationship between state and activity is the characteristics of the state networking technique. We have been applying this technique in the phases 'basic design', 'detailed design', 'programming and testing'. From the experience gained so far we may say that the state networking technique is beneficial at all levels of detail of an application.

In the 'basic design' phase the state networking technique was the vehicle for defining the item relationships mentioned above. Each one of the functions was analysed separately. How we proceeded in the 'basic design' phase we will demonstrate by means of the function 'Recording load unit/breakdown':

When this function is being executed, the identifying numbers of the items transport frame, load unit and flight are recorded using the screen layout as shown in **Fig. 5.** The aim is to set up the relationships FM ∪ LU, LU ∪ FM, LU ∪ FL, FL ∪ LU of the items thus recorded **(Figs. 6 and 7).** It should be noted that pointers in data bases are not sufficient in our application to establish a data base relationship, but that additional status values are required.

In order to establish relationships between items in the data base these items must not have any relationship to any other item. This condition has to be checked for each item. If any relationship between items happen to exist—this may be due to end user errors or programming bugs—they have to be detached. **Fig. 8** is an example of relationships which a single item, the transport frame no. 12, may have.

Table 1 defines how relationships between data bases are connected and disconnected for 'Recording load unit/breakdown'. To simplify the matter we consider only the relationships of the item transport frame. Under the heading 'activity' the name of the function is specified. Under the heading 'intention' the subfunction is pinpointed on which we focus in our analysis; for instance 'connect FM ∪ LU' means that we want to establish the relationship from FM to LU. In the example of Fig. 7, the relationship would be established between FM-No. 12 and LU-No. P1G4711LH. While trying to connect these items we may encounter the following states (see Table 1):

(*a*) there is a relationship between FM and LO

(*b*) there is a relationship between FM and LU

(*c*) there is a relationship between FM and AWB

(*d*) there is *no* relationship between FM and any other item.

These states are defined in column 3 of Table 1. In the example in Fig. 8, the transport frame no. 12 which has just been input from the screen is still connected to LU-No. UQ60003LH, to AWB-No. C11-1234 and LO-No. 112. Depending on the state which applies, a specific subactivity has to be executed which is defined in the right hand column of Table 1. Usually there are no relationships between items and the new relationship can be established. If, however, old relationships do exist they have to be disconnected.

For each of the FAS functions which incorporate item relationships, an analysis was made similar to 'Recording load unit/breakdown'. The number of these functions is 35. The result of this analysis was documented by means of the state networking technique, as well.

Connecting and disconnecting items is one of the more difficult tasks of the FAS. The state networking technique has enabled us to tackle this problem in an early phase of the project and to understand fully its implications. The results of the analysis are the basis for the next project phase: the 'detailed design' phase.

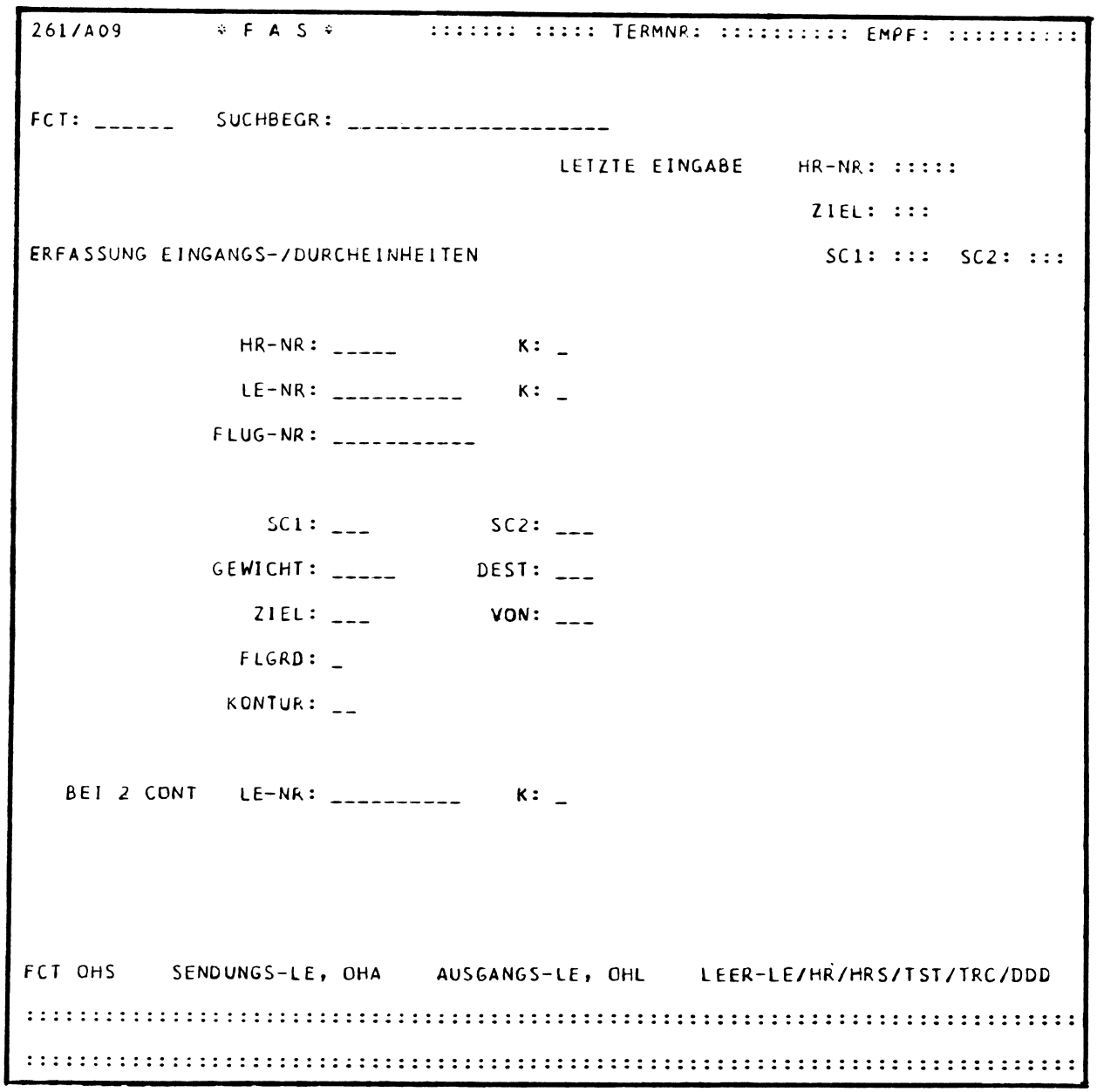

Fig. 4 The screen layout for the recording of physical data of a load unit/breakdown

4. The use of the state networking technique during the detailed design phase

The function 'Recording load unit/breakdown' does not of course only consist of the activity 'connecting and disconnecting relationships of items' which we presented in Section 3. Other subactivities are

1. The validation of input data, i.e. the attributes of each input element such as numeric or alphabetic are checked.
2. The consistency test, i.e. the logical dependency among the input elements is checked and the compatibility of the input data with already recorded data is tested.
3. The storing of the input data.

During the 'detailed design' phase functions are specified down to a level where programming can start right away. In this phase functions are also grouped into programs. **Figs. 9, 10** and **11** present an overview of the program 'Recording load unit/breakdown' by means of the graphical state networking technique. In these figures boxes represent activities and bars represent states. The state which directly precedes an activity is meant to trigger this activity. The states which directly follow an activity describe the results of this activity.

The names of the states and activities are such that they are meaningful to the end user. The state 'message start' triggers the activity 'data validation'. The result of this activity is either the state 'input error', or 'input ok'. In the case of 'input error', an error message is sent to the user, and the processing of the message is terminated. In the case of 'input ok', the consistency of the input data is checked. If the state 'consistency ok' is attained, the activity 'identifying existing DB-relationships' is triggered. This activity determines the kind of relationships the items FM and LU, which have just been recorded, currently

```
261/A09        * F A S *        ::::::: ::::: TERMNR: ::::::::::: EMPF: ::::::::::

FCT: QHE___   SUCHBEGR: ____________________

                                        LETZTE EINGABE    HR-NR: :::::

                                                          ZIEL: :::

ERFASSUNG EINGANGS-/DURCHEINHEITEN                        SC1: :::  SC2: :::

                HR-NR: ___12          K: _
                LE-NR: P1G4711LH_     K: _
              FLUG-NR: LH9999/28__

                  SC1: ___          SC2: ___
              GEWICHT: _____        DEST: ___
                 ZIEL: ___          VON: ___
                FLGRD: _
               KONTUR: __

   BEI 2 CONT   LE-NR: __________    K: _

FCT CHS    SENDUNGS-LE, OHA    AUSGANGS-LE, CHL    LEER-LE/HR/HRS/TST/TRC/DDD
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Fig. 5 Example for the recording of physical data of a load unit/breakdown

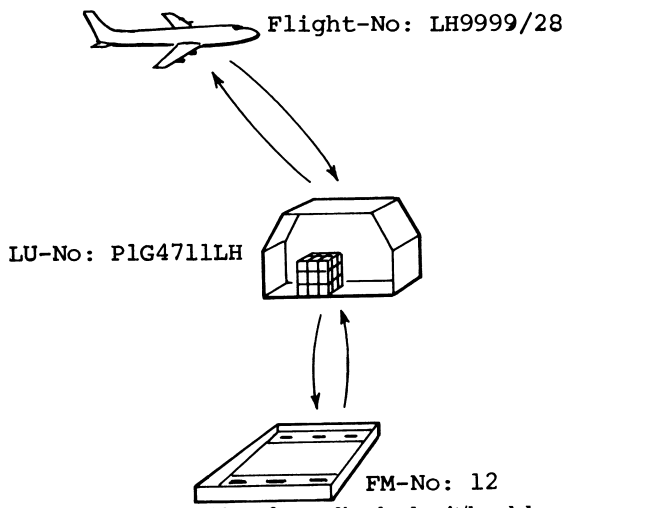

Fig. 6 Item relationships of recording load unit/breakdown

have. The result of this activity is reflected by the states 'EX FM ∪ LO', 'EX FM ∪ LU', etc. Contrary to the previous activities it is possible at this point that a number of states may hold simultaneously. This fact is expressed by the symbol 'D'. If there is a branch directly after an activity, i.e. if an activity is not followed by the symbol 'D', then this represents an exclusive OR of states: one and only one state may be reached by this activity. The symbol 'D' represents an inclusive OR, i.e. any number of states can be reached by this activity. The branch which in Fig. 10 leads to the states 'EX FM ∪ LO' and 'FM ∪ LO disconnected' respectively, expresses an exclusive OR. In the case of 'Recording load unit/breakdown' each branch after the symbol 'D' is entered. If an existing DB-relationship has been identified, a state of the kind 'connected' such as 'EX FM ∪ LO' is set. This state triggers a disconnecting activity such as 'disconnect FM ∪ LO', which in turn sets a state of the kind 'disconnected' such as 'FM ∪ LO disconnected'. If no existing DB-relationship has been identified the activity 'identifying existing DB-relationship' directly sets

the state of the kind 'disconnected' such as 'FM ∪ LO disconnected'.

Before the function 'Recording load unit/breakdown' proceeds, all states of the kind 'disconnected' must have been reached. This fact is reflected in the compound state 'FM ∪ LU disconnected and FM ∪ LO disconnected and . . . , etc.'. This state now triggers the activities 'storing of input data' and 'storing of data affected by disconnections'. Fig. 11 shows that these two activities may be executed in parallel, as they are independent from one another and are triggered by the same state.

After both activities have been terminated, i.e. the state 'data stored' has been reached, an 'ok' message to the sender of 'Recording load unit/breakdown' is transmitted and the final state of the function has been reached. The state networks in Figs. 9, 10 and 11 enable us to depict the logical dependency among activities, i.e. the logical flow of a function. This state network is developed stepwise:

We start with an overview of a function which is successively detailed using the state networking technique. The refinement of the function is stopped when the scope of each activity is sufficiently narrow. It is difficult to establish a general rule when this is the case. Our experience in the FAS had led us to believe that an activity with a narrow scope has in general no more than 30 single steps. (A single step in programming corresponds roughly to an imperative statement which may be constrained by a condition clause.) We call such an activity 'action'. There are, however, actions in the FAS with more than 100 single steps. Each action was specified using pseudo-code as the definition method and HIPO-draw as the documentation tool (see **Fig. 12**). Hence the complete definition of a function consists of three parts: the final state network, the specification of the actions and the input/output formats.

The complete definition of the function 'Recording load unit/breakdown' comprises 42 actions and 80 states. The state network of Figs. 9, 10 and 11 represents 'Recording load unit/breakdown' at a rather crude level: There we have only 10 actions and 21 states. The state networking technique can also be employed within the actions to control the single steps. In the complete definition of the function 'Recording load unit/breakdown' there are 30 more states of this kind.

A comparison between Fig. 10 and Tables 1 and 2 will show

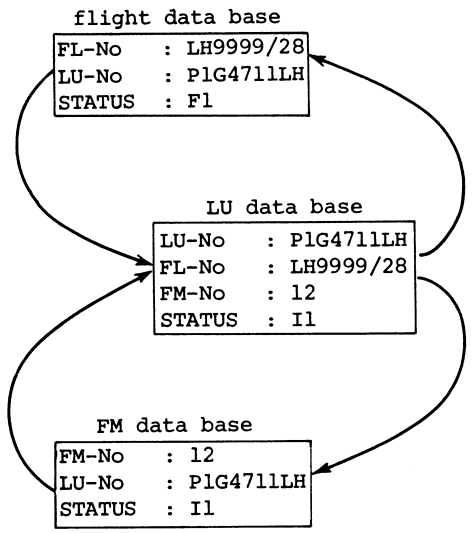

Fig. 7 Data base relationships after recording load unit/breakdown

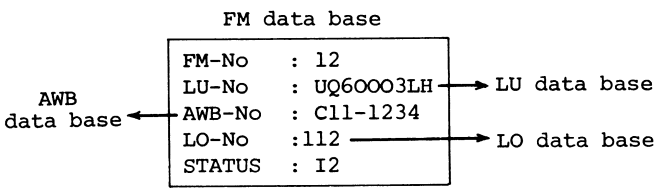

Fig. 8 Data base relationships which may exist for a single transport frame

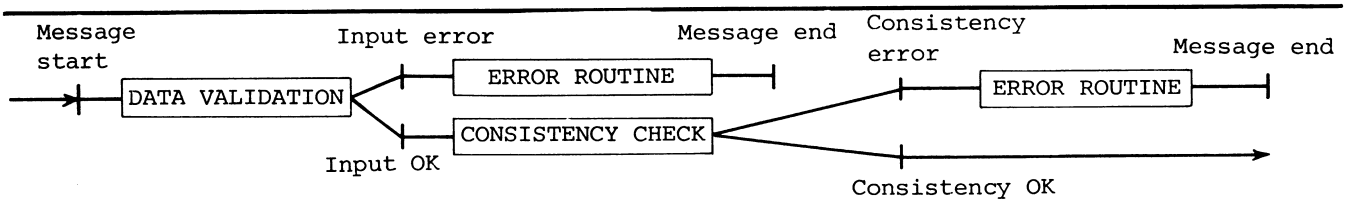

Fig. 9 The state network for recording load unit/breakdown

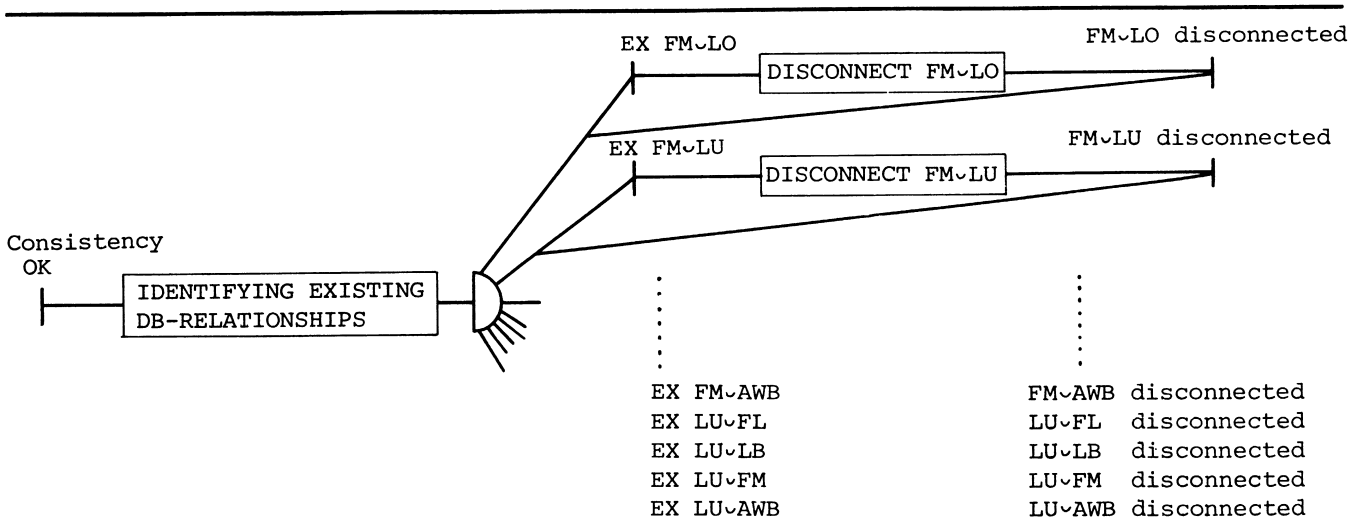

Fig. 10

Table 1

Activity	*Intention*	*State*	*Triggered subactivity*
RECORDING LOAD UNIT/ BREAKDOWN	CONNECT FM ∪ LU	EX FM ∪ LO	DISCONNECT FM ∪ LO, DISCONNECT LO(FM) ∪ FM —IF FM(LO) = LO, SET STATUS (FM) = I4 AND DELETE FM IN LO —IF FM(LO) ≠ FM, SET STATUS (FM) = I4
		EX FM ∪ LU	DISCONNECT FM ∪ LU, DISCONNECT LU(FM) ∪ FM —IF STATUS (FM) = I1 AND FM(SEG LU(FM)) = FM, SET STATUS (FM) = I4 AND STATUS (SEG LU(FM)) = I4 —IF STATUS (FM) = I2 AND FM(SEG LU(FM)) = FM, SET STATUS (FM) = I4 AND STATUS (SEG LU(FM)) = I4 AND ERROR MESSAGE —IF STATUS (FM) = I1 AND FM(SEG LU(FM)) ≠ FM, SET STATUS (FM) = I4 —IF STATUS (FM) = I2 AND FM(SEG LU(FM)) ≠ FM, SET STATUS (FM) = I4 AND ERROR MESSAGE
		EX FM ∪ AWB	DISCONNECT FM ∪ AWB, DISCONNECT AWB(FM) ∪ FM —IF STATUS (FM) = I1 AND THERE IS ANY AWB-LU LOCATION DATA WITH FM(AWB-LU LOCATION DATA) = FM, SET STATUS FM = I4 AND STATUS (AWB-LU LOCATION DATA) = I4 —IF STATUS (FM) = I2 AND THERE IS ANY AWB-LU LOCATION DATA WITH FM(AWB-LU LOCATION DATA) = FM, AS ABOVE AND ERROR MESSAGE —IF STATUS (FM) = I1 AND THERE IS ANY AWB-LU LOCATION DATA WITH FM(AWB-LU LOCATION DATA) ≠ FM, SET STATUS (FM) = I4 —IF STATUS (FM) = I2 AND THERE IS NO AWB-LU LOCATION DATA WITH FM(AWB-LU LOCATION DATA) = FM, SET STATUS (FM) = I4 AND ERROR MESSAGE
		THERE ARE NO RELATIONSHIPS BETWEEN ITEMS	ESTABLISH RELATIONSHIPS OF FM —STORE LU-NO. IN FM SEGMENT AND SET STATUS = I1

that the result of the 'basic design' phase, namely the relation between states and subactivities triggered by these states, is directly transferred into the 'detailed design' phase. Hence the state networking technique allows us to investigate and to analyse critical parts of the application early in the development process and to consider the remaining subfunctions later during the 'detailed design' phase.

5. The use of the state networking technique during the programming and testing phase

The results of the 'detailed design' phase, namely the final state network, the specification of the actions and the input/output formats are the basis for the programming and testing of the function 'Recording load unit/breakdown'. This program consists of a control section and a number of subroutines. The control section (**Table 2**) has been derived from the final state network (Figs. 9, 10 and 11) by directly translating a graphical representation into a programming language syntax: In Table 2 the left column corresponds to the states and the right column to the actions triggered by these states. The actions become subroutines in the program. Hence the structure of the program corresponds exactly to the final state network.

The development of the test case is the task of the end users. It starts right after the end of the 'detailed design' phase and uses the same documentation as programming. The test strategy is set up on the grounds of the final state network.

It must be ensured that the test cases cover all branches of the state network. The test strategy generates a basic set of test

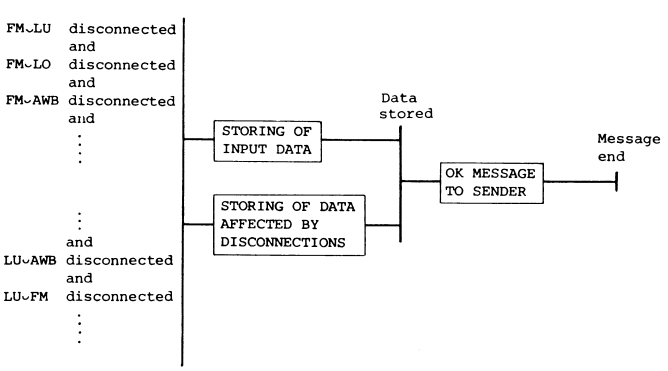

Fig. 11

cases. Each test case runs through certain actions of the state network. It has then to be varied according to the single steps of the actions in order to cover all existing branches. The test results are also analysed by the end users.

The approach for programming is top-down: a functionally related part of the control section is programmed, together with the subroutines called in this part of the control section. Then this program portion is tested before programming proceeds. If the program abnormally ends during testing, the history of the test run is captured by means of the states which have been reached. This significantly helps the programmer debug his program. Thus the state networking techniques allows us to program and test in a modular fashion.

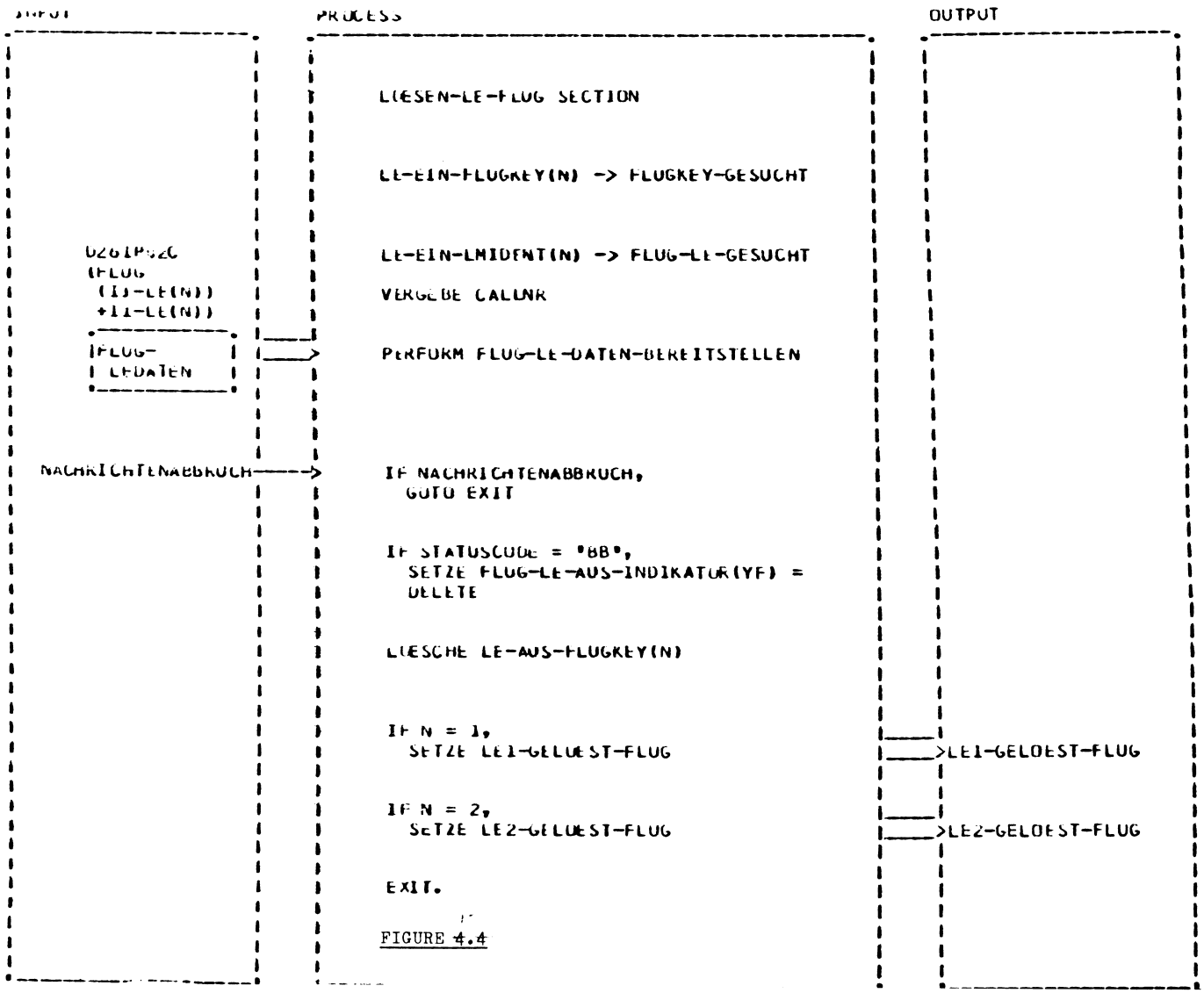

Fig. 12 The definition of an action

6. Assessment

As we have seen, the basic elements of the state networking technique for the specification of functions are state and activity. This technique can be employed in each phase of the application development process. By using graphical aids the application becomes easier to understand.

The state networking technique enables us to develop the application stepwise: we may start at a crude level and advance to more and more detailed design levels. The end users are able to participate in the project without any special DP knowledge. They may, at any stage of the project, verify the functional correctness of the application. As we employ terms which are familiar to the user to name states and activities, all participants of the project may use the same documentation. Hence the communication between end users and DP professionals will be improved.

Critical parts of the application can be analysed separately in an early phase of the project. Thus we may avoid costly redesign of the application at a later phase. The modularity of the application is increased, the benefits of which are particularly felt in the programming and testing phase. Test cases can be developed systematically.

To summarise we may say that the state networking technique

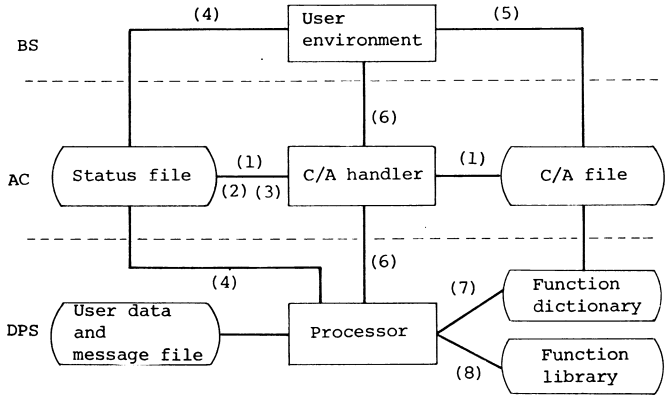

Fig. 13 The AC as an interface between the business system and DPS

is an application development method which is beneficial to both the end user and DP professionals.

7. The AC as a software system

The state networking technique is the methodology proposed

Table 2 Control section of the . . .

		PERFORM INITIATION
IF MESSAGE START		
		PERFORM DATA VALIDATION
IF INPUT ERROR		
		PERFORM ERROR ROUTINE
IF INPUT OK		
		PERFORM CONSISTENCY CHECK
IF CONSISTENCY ERROR		
		PERFORM ERROR ROUTINE
IF CONSISTENCY OK		
		PERFORM IDENTIFYING EXISTING DB RELATIONSHIPS
IF EX FM ∪ LO		
		PERFORM DISCONNECT FM ∪ LO
IF EX FM ∪ LU		
		PERFORM DISCONNECT FM ∪ LU
IF FM ∪ LU	DISCONNECTED AND	
FM ∪ LO	DISCONNECTED AND	
FM ∪ AWB	DISCONNECTED AND	
LU ∪ FL	DISCONNECTED AND	
LU ∪ LB	DISCONNECTED AND	
LU ∪ FM	DISCONNECTED AND	
LU ∪ AWB	DISCONNECTED AND	PERFORM STORING OF
	.	INPUT DATA
	.	
	.	PERFORM STORING OF
	.	DATA AFFECTED BY
	.	DISCONNECTIONS

in the AC concept to develop applications. In the FAS project we employed this methodology to develop programs and to design application functions which are to be performed by the end users, such as user procedures to handle items, to input data, to correct errors. As a matter of fact the AC concept advocates the integration of DP and user activities into one system and the construction of this system by means of one methodology: the state networking technique. The set of the state networks would make up the so-called *application model* and would be contained in the *Condition/Action File* in the shape of the state network tables. The control section of the program 'Recording load unit/breakdown' of Table 2 may be regarded as a kind of state network table: In the left column we find states (including Boolean expressions of states) to which actions are linked in the right column. An entry in a state network table is made up of a state and actions linked to it, representing a trigger condition and the activities triggered by the condition.

Hence we may view the application model as consisting of control sections such as in Table 2 for programs as well as for user procedures. We extract the control structure from the application and have it reside in a special file, the Condition/Action File.

The *Status File*, on the other hand, would contain the instances of programs and user procedures in the form of data and states which occur in each instance. Thus the Status File is complementary to the Condition/Action File insofar as the structure of application activities are described in the latter one, while their occurrences, i.e. their data and states reached so far, are kept in the former one.

A third component, the *Condition/Action Handler*, 'arbitrates' between the Condition/Action File and the Status File by accessing the Status File to determine how far things have proceeded in the operation of the application and by comparing it with the application model of the Condition/Action File to find out what has to happen next in the application. The pattern by which the Condition/Action Handler accesses Condition/Action File and Status File is defined in the *polling list* of the Condition/Action Handler. This polling list links time conditions to activities of the Condition/Action Handler.

The Condition/Action File, the Status File and the Condition/Action Handler make up the AC as a software system, (see **Fig. 13).** Fig. 13 depicts three levels of mapping in a computer supported business system (BS): The highest level is the BS level which represents the organisation where the application is to run, its objectives, policies, procedures. The application is placed on the intermediate level, and the lowest level is taken up by the DPS. The square boxes in Fig. 13 represent a device or human operator with processing capability, the boxes with round edges represent files. The Condition/Action Handler prompted by its polling list polls the state network tables in the Condition/Action File and the pertinent data and states in the Status File to identify activities which are due and resources to carry them out (1). There are two categories of resources: the DPS and the end user. An end user may be a person, a department or a machine in the user environment. The Condition/Action Handler marks instances of actions for execution by chaining the pertinent data and states to action queues in the Status File (2). To confine all initiative for performing activities to the application model and the Condition/Action Handler would be unacceptable to the organisation and furthermore completely unrealistic. An end user or the DPS express their wish for activities to be executed by means of messages which flow between them (6).

The Condition/Action Handler is capable of intercepting these messages and inserts them to the Status File (3), similar to the way it processed data and states. These data and states as well as the messages receive a priority number when they are linked to the action queues. The priority number reflects the urgency of the activity to be attended to by the resource and specifies how quickly the resource should react. Activities which are due for execution and messages are made available to the user and DPS respectively (4). Being made available to the user or DPS means that both resources may inspect queues and entries in the queues which are assigned to them. They may do this usually at their own discretion. However, the AC may also cater for a prompting facility to notify a resource that action is needed. The sequence of actions to be executed is left to the resources.

If a user wishes to learn what his task is all about he gets the desired information from the Condition/Action File (5). In doing so a user employs the AC as a documentation facility. DP programs which constitute DPS actions can be interrogated in the function dictionary (7) and called from the function library at run time (8). Both function dictionary and function library are part of the DPS. The function dictionary specifies the composition of each DP action, such as the DP program modules and data involved, while the function library contains the object code of these modules.

8. Some benefits of the AC as a software system

What would be the benefits of having the AC as a software system?

1. There is an application model which is the reference point for both user and the DPS and which remains in existence during the whole of the life cycle of the application. User and DP activities are actually based on it. It is recommended to include in the application model also procedures of the BS which have no bearing on the DPS. Thus the application model may become a model of the whole of the BS.
2. The logical simplicity of an application and hence its comprehension by a user and systems analyst are greatly enhanced by the AC advocating (*a*) a separation of the general case of activities in the Condition/Action File from the individual case, i.e. the occurrence of activities, messages, data in the Status File; (*b*) the isolation of the time aspect of an application in the Condition/Action Handler. This component contains a 'timetable' of the application in the shape of its polling facility; (*c*) a separation of triggering functions from processing functions. The triggering functions are the domain of the Condition/Action Handler, while the processing functions are the domains of the user or the DPS.
3. Through the AC we would be able to simulate the behaviour of the DPS and the application as a whole, while we implement them, and to predict their performance under various load conditions. The state networks in the Condition/Action File represent a model of the application on which simulation can be directly performed. To this end actions in the state networks are assigned resource consumption values. The frequency of activities and messages can be derived from the usage pattern of the application. The Condition/Action Handler is the driving device in the simulation runs, which can take place as the development work progresses. Thus the simulation capability of the AC would ensure at various stages of the development process that the application design is not only functionally sound, but also meets the performance requirements as laid down by the user.
4. We may visualise any BS as consisting of a subsystem of physical activities, such as the production of goods or the provision of services, and of a control subsystem. The control subsystem comprises the decision procedures, data files and the information flow between resources. Tuning this control system means finding the 'optimum' time pattern for making decisions and then constantly adjusting it to the changing requirements of the BS. The AC has a range of tools for setting and adjusting the control cycle in a BS. The main tool is the polling list of the Condition/Action Hander, as it specifies, by linking Condition/Action Handler processes to time conditions, how quickly activities are moved on in the user environment and the DPS.

Hence the AC as a software system would perform for the application what a DBMS performs for data and an operating system for computer resources: It would control the application more efficiently than systems have hitherto done.

References

STECHER, P. (1977). Proposal for an interface system between the business and data processing systems, *The Computer Journal*, Vol. 20 No. 3, pp. 194-201.

STECHER, P. (1978). On the Interface Between Business Systems and Data Processing Systems, Ph.D. Thesis, University of London.

IBM (1). HIPO-Draw, Program-no. 5796-BFF.

IBM (1974). HIPO, A Design Aid and Documentation Technique, GC20-1851.

IBM (1977). TSO-3270 Display Support and Structured Programming Facility (SPF), Version 2, General Information Manual, GH20-1974.

IBM (1978). Information Management System/VS, System/Application Design Guide, SH20-9025.

Book reviews

Logic for Problem Solving, by Robert Kowalski, 1979; 287 pages. (*The Computer Science Library/North-Holland*, $18.95, $9.95 paper)

One of the interesting attributes of artificial intelligence is that it has stimulated the creation of novel formalisms that are both fundamental and applicable to a wider area. Robert Kowalski's work in the field is a good example of this, and for several years he has seen formalism in logic as a fundamental way of expressing notions in problem solving. The central theme of the book is a form of 'clausal' logic which is related to top-down reasoning rather than the more conventional inferential schemes as found in classical texts such as Quine's.

The introductory chapter centres around the way that one expresses statements in clausal logic, while Chapter 2 begins to relate clausal forms to traditional AI topics such as data base searches and semantic networks. Chapter 3 discusses the parsing problem in the framework of clausal logic, while Chapter 4 arrives at tree searching in problem solving applications. This appears to be the intellectual epicentre of the book, which then goes on to consider computational procedures and plan formation, heading resolutely towards provability and ending with a brief consideration of the dynamics of informational systems which '. . . attempts to combine the traditional rôle which logic plays in epistemology and the philosophy of science with its new rôle in computing'.

For the browser, the book may be somewhat heavy going, while for the teacher of AI or Computer Science Theory it is an essential addition to the bookshelf.

I. ALEKSANDER (Uxbridge)

Simulation: Principles and Methods, by W. Graybeal and U. W. Pooch, 1980; 249 pages. (*Prentice-Hall*, £12·95)

Simulation is a technique which involves the application of an assortment of mathematical tools to the solution of a problem. It is beset with pitfalls for the inexperienced and unwary. Graybeal and Pooch have described the assortment of mathematical techniques and the pitfalls, together with various approaches and simulation languages in one single volume. In consequence, the treatment of any topic can at best be described as superficial. However, there are virtues in this approach. It illustrates a methodical approach to simulation in outline and, provided the simulator is already acquainted with various branches of mathematics, or is prepared to work hard at becoming so acquainted, could act as a guide and catalyst. It must be emphasised that it is not a practical how-to-do-it book. In particular the last chapter raises the problems to be encountered after a simulation model is completed and validated, namely the design of experiments using the model. This is a much neglected area in the world of simulation.

The presentation of material is clear, with mathematical results in general being assumed. Concepts are both explained and described mathematically and there are a number of simple arithmetic examples illustrating these and other ideas.

This book is designed to support a graduate course (in America) for potential users, which explains the inclusion of three pages on analogue computing (completeness?) but not the absence of activity diagrams.

R. E. SMALL (London)