Proposal for an interface system between the business and data processing systems

P. Stecher

Department of Computing, London School of Economics, Houghton Street, London WC2

This paper presents a new concept called the application controller which addresses some DPS/end user interface problems. It is a new way to see business activities, DPS activities and user/DPS interactions and is a proposal for a software package. The aim of the application controller is to involve the end user in the definition and design of his application to a high extent and to base DPS and user activities on a model of the application. This model accommodated in the application controller describes the application to the user and to the DPS not only during system analysis and design, but during the operation of the DPS as well. The application controller thus adds to the existing modes of DPS's, namely batch and real time, the state triggered DPS.

(Received July 1976)

1. Three levels of abstraction

An installed data processing system (DPS) is a mapping of a particular business system (BS) in the sense that certain activities in the BS, which previously have been accomplished manually or by some special purpose machine (e.g. desk calculator) or not at all due to the vastness of the activity (e.g. sales forecast by product, country and area), are now carried out by a computer using programs, files and a schedule of program runs.

A BS is characterised by the industry within which it operates, by its organisation, objectives, policies and procedures. When we talk of BS activities which we want to 'computerise', we have in mind repetitive activities with a stable structure that are carried out in a cyclic manner, e.g. calculate pay, sales performance analysis, inventory level calculation, or on demand, e.g. customer order processing, preparation of shipments of ordered goods. We are not so much concerned with activities that arise spontaneously and have an unstable structure, e.g. identification of an employee, who has been with the company for more than five years, holds a degree in aircraft engineering and speaks Russian.

A DPS is characterised by its hardware, its mode of operation (batch, real time, time sharing), its operating system, its programming languages and its file support system (simple files or DBMS). We map the BS onto the DPS by writing programs, designing files and setting up a procedure indicating when programs have to run (schedule in a batch system, triggering messages in a real time system). There are of course many BS activities which are not mapped onto the DPS for various reasons. After the DPS has become an integral part of the BS, we can still visualise it as a mapping of the BS in the sense defined above.

As there is no way yet to translate BS activities directly into programs, files and triggers, we have to do it in intermediate steps with the help of system analysis and design techniques, that is to say, we use an additional level between the BS level and the DPS level. This level shall be called application level. The application level is characterised by the procedures, techniques and forms which are employed to describe the business activities, the information and parts flow, and to specify requirements and design the application, i.e. what is carried out by a user, where the DPS comes in and of what nature the user/DPS interactions are. The simplest application level techniques are plain English, flowcharts and decision tables. By specifying requirements and designing a solution descriptively or by flowcharts and decision tables, we have actually done a mapping from the BS level to the application level and constructed an application model. By translating requirement specifications into programs, files, and triggers, we are mapping the application model onto the DPS level.

It is of course feasible to introduce any number of levels between the BS level and the DPS level to control the complexity of the transition, in a way similar to that suggested by Dijkstra (1969) for designing operating systems, but we will confine ourselves to three levels of abstraction. The highest level in the hierarchy is the BS and the lowest one is the DPS. On each level, problems of the next higher level are expressed by means of the level.

2. Some application modelling techniques

Literature provides us with a large number of concepts for building application models. Generally speaking each technique besides project control techniques used in system analysis and design generates an application model. This applies to techniques with a narrow scope, such as flowcharts and decision tables, as well as to tools which address all aspects of an application. To the latter belong those suggested by Young and Kent (1958), CODASYL Development Committee (1962), Langefors (1966), Bubenko (1973), Grindley (1966 and 1975) and Teichroew (1971). The rationale put forward by these authors has basically been to separate user requirements, the BS aspects, from implementation considerations, the DPS aspects. The user is to specify, elaborate and improve his application without being restrained by implementation considerations. The system analyst on the other hand can base the DPS design on a well defined application model.

One may ask why any of these application modelling techniques have not been incorporated on a large scale or commercially exploited. Two points in particular come to mind:

1. *Conditio sine qua non* for an application modelling technique of today is that the technique involves the end user to a high degree in requirement specification and application design. Some of the techniques having been developed in the nineteen fifties or early sixties, when user involvement was not yet an important criterion, are of a mathematical rigour and abstraction, which make them unattractive to end users.
2. The system analysts on the other hand seem to have disregarded them, because they considered the application model thus generated to be a mere stop on the way to the DPS, bearing no significant benefits. For them any application model has been a superfluous vehicle, whose construction would only waste time and misdirect their attention from the DPS. This attitude has been due to the lack of aids which would facilitate the mapping from the application level to the DPS level: A system analyst must develop the application model manually by interviewing the user and then translate it into programs, files and triggers. The latter seems to be too complex a task to be done automatically by a

compiler and projects to develop one have apparently not yet succeeded—for instance SODA (Nunamaker, 1971).

Any technique which tries to overcome the above-mentioned drawbacks has to provide a user interface for the application model and establish a close link between the application model and the DPS. The concept suggested in this paper abandons the compilation approach and sets in its place a software system, which accommodates the application model not only during the analysis and design phase, but, still more important, during the operation phase. We call this concept the application controller (AC).

3. Goals of the AC

The AC is:

(*a*) a new way to see things, e.g. BS activities, DPS activities, user/DPS interactions

(*b*) a proposal for a BS/DPS interface system.

The goals of the AC are:

(*a*) The application model generated by means of the AC describes the application to the user and the DPS.

(*b*) The application model must cover all aspects of the BS with respect to the DPS and be close to the user's concept of the BS.

(*c*) The application model must not be transient in nature, i.e. be a stage in the system life cycle, which is left behind when the DPS has once been set up, but must exist throughout the system life cycle and be the reference point of all BS and DPS activities.

(*d*) The user, to whose benefit the DPS is aimed, must be heavily involved in the specification and design of his application.

(*e*) Changes in the DPS ought not to affect the BS. The design of a DPS is a multigoal effort. To solve it analytically is almost impossible: time and costs are too high. And who can guarantee that the data on which the design is based, such as kind volume and frequency of user data or response time required, is complete and accurate? Often a clear idea exists as to what the DPS should achieve. The AC provides facilities to capture this idea and to develop, simulate and adapt a DPS to meet the requirements.

(*f*) If changes in the BS occur, it must be easy to adjust the application model and the DPS to the changes. This goal is difficult to achieve. As long as there is no direct translation from the application model to the DPS, any modification of the BS will cause a certain workload to the user and the analyst to modify the DPS. We can only hope to ease this burden by defining clearly the interface between the BS level and the application level and between the application level and the DPS level and by limiting the interface width.

The AC is presented below in two parts: We start with an example to illustrate the basic ideas of the application controller. Functions and components of the AC, pinpointed in this example, are elaborated in the second part with respect to a potential user and to a possible implementation.

4. The user's view of the AC concept: how to design and operate an application

An example

We shall make the assumption here that the organisation that intends to employ the AC is aware of what a DPS is and the kind of support the end users can expect from it.

The first step for a user is to record activities which are carried out by him or which are planned for implementation. The purpose of activities is to produce outputs or to accept inputs. Inputs and outputs are understood here not necessarily in their DP meaning: output may be a decision to employ a particular person and input may be the arrival of this person at a certain location and date. Let us consider the following example: 'Validating receipts'.

Table 1 The writeup of a business process

Application: receiving goods
Functional department: receiving office
Business process: validating receipts

1. The receiving office is sent the packing slip, which accompanied the receipt, by the receiving bay with any comments about the state of the goods, count variance, etc.
2. Using the packing slip the clerk in the receiving office assigns a unique receipt number to the receipt and records for each part:
   part no.
   part description
   supplier number
   quantity as indicated on the packing slip
   count variance
   purchase order number
3. The clerk will now decide whether to accept or refuse the received parts. Alternatively he may hold them for some time and ask the material control office for their view
4. If the clerk accepts the receipt, files are updated. If he refuses the receipt, the receiving bay is informed so that the receipt can be returned to the supplier. If he holds the receipt, a message is sent to the material control office. Within two days this office has to reply and accordingly the clerk in the receiving office accepts or refuses the parts.

After a receipt of purchased goods has been identified in the receiving bay, the receiving office is activated: it has to update files, see what has to be done with the parts, check the receipt against the purchase orders, sort out discrepancies between ordered and received parts, etc. **Table 1** shows the description of this business process 'validating receipts', which can be compiled directly by a user and put into the AC via a keyboard/CRT/printer terminal (we assume the use of this type of terminal for all communication with the AC).

The access to the AC is controlled by employee number or functional department number. A user can access only those parts of the AC for which he is authorised. From the description of the application a user can verify what his or his department's task is all about. He can have applications displayed using a multilevel index where he selects entries with a lightpen or function keys. Hence the AC may serve as a documentation and training tool.

The writeup of an application may be supplemented by diagrams, input/output formats, decision tables, flowcharts, etc. to detail the activities.

'Validating receipts' is a so-called business process (BP). We can visualise a BP in two distinct ways:

(*a*) as an action hierarchy

(*b*) as a state network.

An action hierarchy depicts into which activities by the user or the 'system' a BP can be split. The action hierarchy for a particular BP is not unique; much is left to the discretion and preferences of the user and the analyst. The hierarchy may have as many levels as it is thought convenient. A state network of a BP depicts important states which a BP can assume and their interrelationship.

Business process (Part 1)

All activities in an organisation can be viewed as business processes, with the following properties:

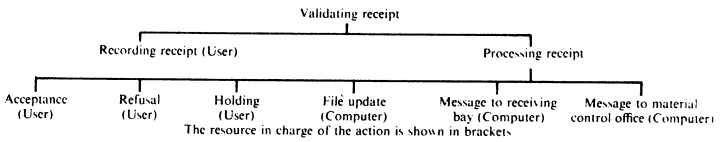
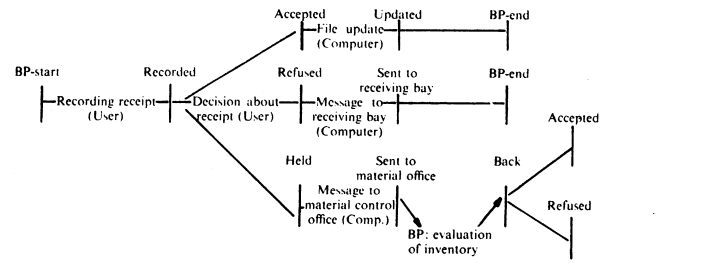


Fig. 1 BP as an action hierarchy



Legend: A bar represents a state, the text between two consecutive bars stands for the action, which leads from the preceding state to the succeeding one. The resource is shown in brackets.

Fig. 2 BP validating receipts as a state network

1. A BP is associated with a functional department, no matter who actually executes the BP, e.g. an inventory level calculation can be carried out by a DPS and is nevertheless associated with the material control office. In most BP's more than one resource is required to perform the actions of a BP. The types of resources depends on the observer's standpoint: a user identifies a drilling machine as a resource, the AC just regards it as another user. On the other hand, for the AC the DPS is a resource (and the only one it can identify besides the user), for a user the AC and the DPS are viewed as the computer resource and ought not to be distinguished. Thus we allow the following resources to be involved in a BP:
   User
   Computer
   AC
   DPS
   Production machine.
2. A BP has a clearly defined start and end and may spread over many intermediate states.
3. A BP is moved from one state to another by an action involving at least one resource.
4. A BP refers to a primary object class, e.g. the BP 'calculate pay' refers to the object class employee, the BP 'inventory level check' to the object class part. A BP may be valid only for a subset of the elements in the object class.

Action hierarchy (AH)

One of the objectives of the AC is to have a description of the application in such a way as to allow a user and the DPS to act on this description. The writeup of the application (see Table 1) has to be formalised so that the AC can automatically access it and operate on it.

The first step towards formalisation is to construct the action hierarchy from the BP writeup (see **Fig. 1**). A user assisted by a system analyst has a free hand to translate the writeup into the AH. He may choose as many levels and entries as he wishes, as long as he observes the following rules:

1. An action must be a piece of homogeneous work.
2. An AH depicts actions ignoring their sequence and the conditions under which these actions can be carried out. The sequence of actions and conditions restricting the execution of actions does not appear on the AH. It is accounted for by the state network of a BP as we shall see below. The AH is akin to an organisational hierarchy, where the lower levels make up the higher ones.
3. There may be only one root entry, which must be identical to the BP name.
4. Entries on the lowest level must have assigned resources such as computer or user. The resource to be assigned is suggested by the kind of activity: data entry is mainly user performed, calculation is done by the DPS, even if a user provides parameters. For instance the action in paragraph 4 of Table 1 (File update) is a pure DPS action, the one in paragraph 2 (Assigning a receipt number and recording the receipt) is a user action, assisted by the DPS. The purpose of assigning a resource is to identify a resource as being in charge of seeing an action through, thus not precluding the involvement of any other resource.

   As it is the user who provides information about activities, we must relieve him of the burden to distinguish between the AC and the DPS. Hence for him the only visible resources are the 'computer' and the 'user'.
5. No other level than the lowest level may have resources assigned.

A user is free to change the writeup and the corresponding AH of a BP. For instance a user in the above-mentioned example wishes to examine the quantity of a part ordered, each time he records a receipt of a part (paragraph 2 in Table 1). The resulting AH is shown in **Fig. 3.** Please note that the activity 'recording receipt' has dropped its resource. This can be summed up by rule 6.

6. If more than one resource plays a major part in an action, this action is represented by a neutral action entry. New action entries with resources are created one level beneath.

The technique described so far is similar to top-down program development. The usual pattern of user or computer actions is not recorded in the application writeup or in the AH actions such as:

   User signing on to a terminal
   User requesting a BP or an action from the AC
   Message format display
   Syntax checking
   Data validity checking
   Error indication by the computer and corrective action by a user
   Authority checking.

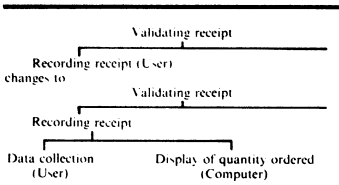


Fig. 3 Extension of an action hierarchy

Action hierarchy		*State network*
Validating receipt	corresponds to	Whole network with actions and states
Recording receipt	,,	Recording receipt + state recorded
Processing receipt	,,	All actions and states to the right of the state recorded
Acceptance	,,	Decision about receipt + state accepted
Refusal	,,	Decision about receipt + state refused

etc.

Fig. 4 The mapping of an action hierarchy on to a state network

State network (SN)

The state network of a BP is constructed based on the BP writeup and the AH (see Table 1 and Fig. 1 respectively). The arcs represent state transitions (=actions) and the nodes, states (**Fig. 2**). Each state transition is associated with a resource and corresponds to an action on the lowest level of the AH. If a BP is initiated, which is done either by a user or the AC, the SN for the BP is entered.

If alternative states can be reached from a particular state, it is up to the resource to decide which one is to be selected, for instance after the state 'Recorded' in Fig. 2 a user has to decide whether to accept, refuse or hold a receipt. By constructing the SN from the writeup and the AH we have actually done a mapping between AH and SN (see **Fig. 4**). Hence we can interpret a state in the SN as triggering the succeeding action or as triggering a subnetwork which correlates to an entry in the AH. For instance the state 'Recorded' triggers action 'Decision about receipt' in Fig. 2 and the action 'Processing receipt' in Fig. 1. It must be defined in the AH, which states in the SN 'terminate an action on the intermediate and upper levels'.

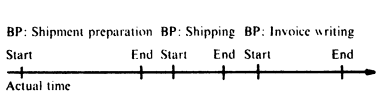
The same states can be reached through different actions, for instance the states 'Accepted' and 'Refused' after the BP 'Inventory evaluation at the material control office' are identical to the states 'Accepted' and 'Refused' after the action 'Decision about receipt'. That is to say, the BP 'Validating receipt' continues after 'Inventory evaluation' with either 'File update' or 'Message to receiving bay' until the state 'BP-end' is attained.

When it seems convenient we may use pseudo-actions in an SN. A pseudo-action is an action, where no resource is assigned. In Fig. 2 the actions between states 'Updated' and 'BP-end', 'Sent to receiving bay' and 'BP-end', 'Sent to material office' and 'Back' are all pseudo-actions.

Messages and connective entries

Fig. 2 illustrates how communication between BP's is established through messages. For instance a message is sent to the material control office, telling it that a reply is expected from it. This message gives rise to a BP 'inventory evaluation' at the material control office. In the SN for 'validating receipts' the BP 'inventory evaluation' is followed by a pseudo-action which leads from state 'message sent' to the state 'message back'. This pseudo-action can be employed to monitor that the reply from the material control office happens in a certain time lapse.

Messages may be verbal, handwritten, typed or computer printouts/displays. The function of messages in the AC concept can be further exemplified by the following:



Shipment preparation, done by the customer service department, is followed by the shipping of the assembled goods by the shipping department. The invoice writing can start as soon as the billing department learns about the shipping.

When a shipment has been prepared a message is sent to the shipping department telling it that parts are due to arrive there to be shipped. The sending of this message can be done automatically by the AC: the BP 'shipment preparation' contains an action reading 'Message to shipping department'. When a certain state is attained in the SN for the BP, the message with all relevant data is sent by a computer action.

Let us now assume that the message points out that the shipment should only be released after an advance payment has been received from the customer. We have now at least two distinct states, which must be reached prior to the completion of the BP shipping:

1. Parts must arrive from the stock
2. An advance payment must arrive.

This situation can be dealt with as follows:

Due to the message from the customer service department, a BP 'shipping' is opened, either by initiation of the AC or by a user. State 1 is signalled by a clerk in the shipping department who accepts the goods from stock. State 2 is attained either by the billing department sending a message 'Advance payment arrived from customer X' or the 'computer' has to test user files repeatedly for the advance payment.

The BP 'shipping' contains an action which sends a message to the billing department reading 'Within two days of shipping an invoice must be produced and forwarded to the customer X'.

We can summarise: If something happens outside the view of a resource (and a resource's view is defined by all actions which it is currently performing), about which the resource should be kept informed, the resource is told about it by messages.

In Fig. 2 and the example above we have seen the objective of messages to link BP's together and establish communication between them. We call an action entry in a SN, which causes messages to be sent, a connective entry. The message which is actually sent may contain parameters which are employed by the receiving resource for a subsequent action.

Business process (Part 2)

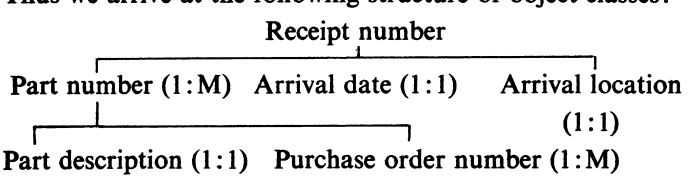
When we analyse the DP activities within a BP, both data requirements and process requirements are relevant. The AC is open for any language or procedure to describe processes and data relations, as long as they are compatible with the AC features outlined so far. Here we shall focus on one aspect of data requirements only: the data associations in a BP.

There is always a primary object class in a BP such as part, receipt or customer. The primary object class in Table 1 is the receipt number. There may be secondary object classes. For instance a shop order has usually a part number associated. It is the code of that part which is produced by the shop order. The part number is a secondary object class of the BP 'monitoring shop orders'. The objective of the primary and secondary object classes is to identify instances of a specific BP.

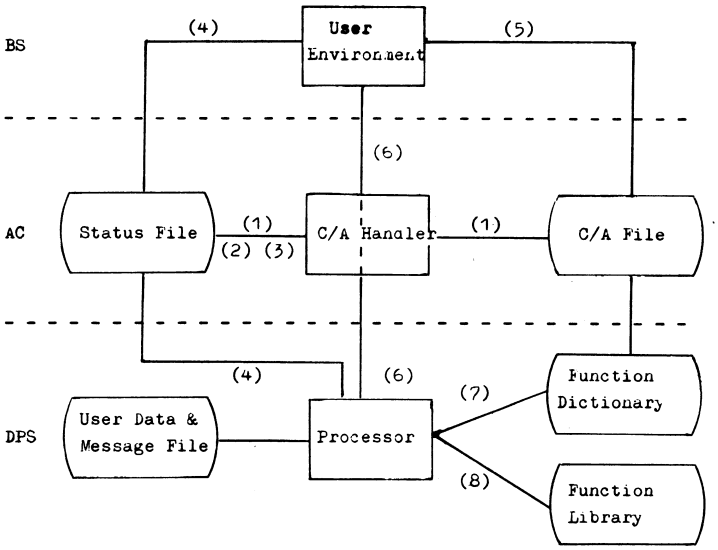
We structure the relationships between object classes in a particular BP as follows:

1. We ask which object classes are directly associated with the primary object class. In Table 1 we find part number, arrival data, arrival location, etc. We state whether the relationship is one to one, one to many, many to many.
2. Then we look for object classes which are associated with part number, arrival date, arrival location and define the kind of relationship similar to above. Part number is associated with part description in a one to one, with purchase order number in a one to many relationship, etc.

Thus we arrive at the following structure of object classes:



This structure is not necessarily a hierarchy, but may be a network structure. It does not intend to be a data representation of the real world but rather tries to capture the usage of data by a particular BP, for instance on the input and output format. As such it is close to data access paths. It is a local view of data associations. How each data element in the object classes is provided, is specified by the processing requirements and the input specifications.



Legend: The C/A handler polls the C/A file and status file to identify actions which are due and resources which have to carry them out (1). The BSR is then chained to the resource queue (2). Messages which link BP's together are inserted to the pertinent BP queue (3). User and DPS actions which are due for execution are transmitted to the user and DPS respectively (4). If a user wishes to learn what his task is all about, he gets the desired information from the C/A file (5). Data and messages which are relevant to the AC must be intercepted from the flow of data between the DPS and the BS (6). DP programs which are behind DPS actions can be inquired in the function dictionary (7) and called (8).

Fig. 5 The AC as an interface between the BS and the DPS

When we develop file structures later, we shall base this process on all local views and arrive at a total view of data relations. Doing so it may be necessary to alter one specific local view, as it may collide with the data associations of another BP. This anticipates the readiness of the user to accept another data association. In this sense the local view of our example is a preliminary view. How the total view is finally implemented by files is the task of the DPS builder, who has to take into account additional criteria such as performance requirements, volume, frequency and security.

We could assign to each data association a name, as it has been suggested by Langefors (1966) and others. It seems, however, we can do without it: if we allowed each user to select the names on his own, we should end up with such a variety of names, that they would become meaningless. If we restricted the selection of names to one person, the Data Base Administrator, he would not be helped either, as he must know the kind of relationship (in addition to the correspondence mentioned above, the correspondence in the reverse direction, occurrence and most important of all the semantics of the association), which requires interpretative description rather than a name.

Each action in a state network uses the object class elements, updates them or creates them. All elements are kept in a logical record, the business status record (BSR), which exists as long as the instance of the BP, to which it belongs. The BSR contains also all states which the instance of the BP has assumed.

An action in a state network is usually triggered by a state. Similar to this property of an action, we can visualise a BP as requiring certain states before it can be initiated. In the example in the previous section, a message is sent to the billing department reading 'Within two days of shipping, an invoice must be produced and forwarded to customer X'. This message relates an 'action' (in reality a BP), namely production of invoice, to a state, namely within two days of shipping. It is sometimes convenient to regard a BP as an abstract action with the resulting state 'completion of BP'. The transition from action to BP is somewhat fluid: If two BP's are put together, the result is a BP. If two actions are put together, the result may be an action or a BP.

The building of the application model in the AC concept can be divided into two major parts:

1. Production of local views in the form of BP's.
2. Development of a total view, which we exemplified by data association above.

For a user it is sufficient to be aware of the local views. The task of the analyst is to build a total view, which is compatible with all local views. It may be necessary to alter a particular local view to improve the consistency or performance of the model.

5. The AC components

The AC has three major components (see **Fig. 5**):

(*a*) C/A file (C/A meaning condition/action)

(*b*) Status file

(*c*) C/A handler.

The C/A file consists of the text for each BP, provided by a user and the formalisation of the BP in the shape of the action hierarchy and the state network. The C/A file is able to answer queries like 'What processes can occur in our organisation?' 'What is the task of a user and of the computer therefore?' and 'Where can the computer assist a user in carrying out his task?'

The status file contains all instances of a BP in the form of the business status record (BSR) and messages. It therefore can reply to questions like 'How far have things proceeded in our organisation?'.

The C/A handler initiates and advances BP instances in order to keep the DPS in pace with the business. It furthermore monitors states to identify undesirable states as soon as possible and to ensure that actions have been carried out in time.

C/A file

The C/A file describes what states can be reached in a BP and what actions have subsequently to follow. The C/A file has three chapters:

1. Application writeup
2. Action hierarchy
3. State network.

We shall elaborate on the state network structure only.

As the AC is presented here as an interface *machine*, system components such as the state network have to be represented in a way which allows automatic processing.

The state network (SN) for a specific BP can be viewed as a table. Each entry in the table relates to a state S_i, the argument, a functionvalue $(\{S_{i+p}\}, R_i, A_i)$, where S_{i+p}, $p = 1, 2, \ldots$ is a finite number of immediately successive states of S_i; R_i is the resource in charge of moving S_i to one or more of the states S_{i+p}; A_i is the action, by which R_i moves S_i to S_{i+p}.

The characteristic of the AC is, that all states and functionvalues can be represented in such a way as to enable the AC to perform operations on them automatically. This is to say, the AC does not need any semantic help, once the SN has been constructed. This is due to the fact that to the AC:

(*a*) state values are provided by resources, i.e. user or DPS, as the result of an action. State values are symbols such as ACCEPTED, S10. The only constraint for the symbols is their uniqueness within a particular BP,

(*b*) resources are user identifiers, e.g. employee number, computer number,

(*c*) actions are again symbols: they require meaning only for the resource which carries them out and not for the AC.

Thus the intelligence of the AC is reduced to table look-up.

States can become predicates. All Boolean expressions are allowed among states. The simplest case of the $S_i - (\{S_{i+p}\}, R_i, A_i)$ relationship is that S_i just triggers A_i, i.e. S_i is a sufficient condition for A_i, e.g. the state 'receipt accepted' triggers 'updating files' in Fig. 2. States may, however, have additional properties, two of which are:

1. A state can be not only a sufficient condition, but also a necessary one for an action.
2. A certain state must be reached by a point in time.

For instance the BP 'daily sales summary' may only be initiated if all sales offices have transmitted their daily sales records. A user who tries to initiate the BP, oblivious of this condition, is made aware of it by the AC. This is catered for by the C/A file, where an action A is specified as requiring a necessary state S.

The point in time, by which a state must be reached can be formulated as an absolute time or as a relative time, e.g. two days after shipment. If a state in a SN has a time constraint attached to it, e.g. it must be reached by 5 p.m., we may introduce progress checks on states prior to the crucial one. These aspects of states are taken care of by the polling list in the C/A handler, as we shall see.

Business status record

If a BP is carried out, we speak of an instance of the BP. When a BP instance is initiated, a BSR is created. It exists as long as the pertinent BP instance. Every BSR has an identifier, object classes and values, and state values. The BSR is a logical record. Its elements are derived from physical files through computation or are copied from physical files through a mapping process.

The BSR's identifier indicates what kind of BP it represents, for instance the BP 'validating receipts'. A BSR contains all object classes and their values which are relevant to the particular BP instance, thus in Table 1 we have

Object class	*Value*
Receipt number	.....
Part number	.....
Part description	.....
Supplier number	.....
etc.	

A structure of the object classes can be recognised as noted in the section on Business process (Part 2).

If a BSR has been initiated by a user, the user's employee number is inserted. If it has been initiated by the AC automatically, the condition which gave rise to it is indicated, for instance a message from a certain department. The resource which carries forward a BP instance by an action is responsible for signalling the state after the action has been terminated. For instance a user who accepts a receipt makes the AC insert the state 'accepted' in the BSR. The AC takes care of the insertion through the C/A handler, a user merely selects the option 'accepted' on his CRT terminal with a lightpen or function key.

Actions which are triggered by a state are not indicated in the BSR, but in the C/A file. The C/A handler determines the next action(s), based on the states as recorded in the BSR. This gives us the greatest liberty of changing a BP without being obliged to change any action entries in the BSR's for the BP.

A user has normally no direct access to a BSR, because he is not interested in the BSR as such, but has its content displayed through queries like 'Where are the goods with receipt number . . . at this moment?'. Each DP program knows how to interpret the identifier, object classes and state indicators in a BSR. This is due to the environmental knowledge which programs have by their very nature.

If a BP instance is completed, the BSR ceases to exist. There are, however, means of recreating BSR data, e.g. using physical files.

Range of the BSR

The BSR is composed of user data and state values. If a BP assumes only 'few' states, its BSR is distorted to a user data file or is even negligible. When does this happen? Generally we can say:

1. If there is little or no interaction between resources, be it different end users or user and DPS, and
2. The BP can be carried out in a continuous time period. For instance in the case of:
   (*a*) pure computational programs like statistical analysis
   (*b*) field research like interviewing customers utilising forms
   (*c*) report generation.

The BSR may also be distorted to a state file, i.e. no user data may be present; for instance the AC can be employed to monitor a production machine on the shop floor. Each time the machine is affected by an unplanned stop, a restart procedure is entered, e.g. the shop supervisor is notified.

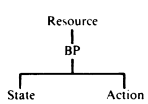
Status file

The C/A file deals with the general case of a BP in the form of an application writeup, action hierarchy and state network, the status file on the other hand looks after the individual case of a BP in the form of the BSR and messages.

BSR's and messages join queues in the status file. Queues may exist for each:

1. Resource, be it employee, department or computer unit. This queue is mainly used by messages, if the BP is not known to the AC.
2. BP, for instance 'validating receipts'. This queue is for messages.
3. State within a particular BP, for instance the state 'recorded' in the BP 'validating receipts' has a queue. This type of queue is for BSR's or messages, e.g. 'message back'.
4. Action. For BSR's only.

Resource, BP, state and action form a three level hierarchy of queues:



C/A handler

To investigate the role which the C/A handler plays in the AC concept, we shall consider the initiation of a BP instance.

A BP can be initiated either by a user or the AC:

1. A user initiates a BP instance on the grounds of messages he has received. A message indicates for example that parts have arrived in the receiving bay. These messages can be verbal, handwritten or computer generated. Computer messages of this kind are chained by the C/A handler to a resource queue or to a BP queue. The user in the department inquires from time to time what messages have arrived.
2. The AC can initiate a BP instance on its own. The C/A handler prompted by its polling list, polls first the C/A file and is led from here to corresponding BP queues in the status file, where messages may be 'waiting'.

In both cases the C/A handler proceeds by creating a BSR and identifying the first action and its resource in the SN for the particular BP. It then passes the BSR together with the action due to the resource. These information elements join the resource/action queue. If the queue is under the control of a

user, it is up to him to inquire for any entries. The AC may 'hint' the kind and number of actions due, but must not take the initiative. To ensure, however, that actions are carried out at all, the C/A handler can monitor queues. If actions have not been carried out in a certain time interval, the user's supervisor is notified. If the queue is under the control of the DPS, the action entries join the job queue of the DPS to be scheduled for execution. Whichever resource is in charge, the action is executed and one or more of the states S_{i+p} is attained. A BP instance is returned to the C/A handler by making the BSR and the new states S_{i+p} available to the C/A handler. It inserts the symbols representing the states to the BSR and chains the BSR to the S_{i+p} queue(s). For instance when a user in Fig. 2 has made a decision about the receipt, one of the states 'accepted', 'refused', 'held' is forwarded to the C/A handler. The C/A handler updates the states in the BSR and chains the BSR to one of the state queues. The next action after the state is not initiated immediately, as the C/A handler has to follow its polling list.

It may well happen that during A_i another resource than R_i is needed. R_i has then to call the assisting resource, but remains in overall charge of A_i and has to indicate S_{i+p} to the C/A handler. In Fig. 2 for instance a user may need the support of a DP program to make up his mind whether to accept, refuse or hold a receipt. A receipt may have arrived too late to be of any use to the company, or a certain tolerance level of quantity variance may have been exceeded.

Polling list of the C/A handler

The polling list is a constituent part of the C/A handler. It links C/A handler actions to *time* conditions to serve three purposes:

1. To initiate and advance BP's to keep the DPS in pace with the BS. States are tested on a regular or irregular basis. If they are true, actions and resources are identified and BSR's and messages are linked to resource queues as noted in the previous section.
2. To monitor states to make sure that actions have been carried out (if true, no action; if false, message to supervisor or corrective action) or to identify undesirable states as soon as possible (if true, message to supervisor or corrective action; if false, no action).
3. To make a resource determine a state. If the C/A handler on its own is unable to determine a state, it may initiate a state check program which is carried out by a resource. The C/A handler is told about the result of the state check program.

Examples:

Item 1: Five times a day the BP 'validating receipts' is advanced. It is checked how far BSR's have proceeded and which actions have to be initiated.

Item 2: A sales summary is generated daily. To be able to do this, customer orders from sales offices throughout the country must have been transmitted by 5 p.m. The C/A handler therefore checks at 5 p.m. daily whether this has happened. If it has, the BP 'sales summary' is initiated; if it has not, a sales supervisor is notified.

Item 3: The DPS may be prompted by the AC to browse through customer files to find out whether an advance payment has been made. If true, the shipment is released.

Initially the polling list and polling characteristics were derived during system analysis and design: each user specifies for his application which BP's and states are important, by assigning weights. Important BP's and states are polled more frequently. The user is free to change the weights during the operation of his application and thereby change the polling characteristics. Furthermore a monitoring device can check whether these weights are reasonable, by matching the number of pollings against the number of subsequent C/A handler actions.

The C/A file is always polled first, then the status file is accessed. So as not to overload the C/A handler through excessive polling, polling may be switched on/off for a particular BP or state.

6. Benefits of the AC

The benefits of the AC can be identified:

1. The application model, generated by means of the AC remains in existence during system analysis, design and operation of the DPS. Thus we have avoided the main drawback of most application modelling methodologies, namely to dump the application model once the DPS has been installed. The AC provides the necessary incentive for the system analyst to undertake the task of constructing the application model.
2. The AC is Janus-faced, i.e. it looks to the user and the DPS. This property bears the following advantages:
   (*a*) it permits a user to get involved in requirement specification and application design, as the AC terms are business oriented rather than DP oriented
   (*b*) it improves communication between user and DP professional during analysis and design
   (*c*) it provides documentation standards and enforces documentation, as the DPS is based on the application model and all access to the DPS is through the AC.
3. We are now able to simulate the behaviour of the DPS before implementation and predict its performance. This can be done by assigning consumption values to user and DP actions. The existence of well defined states in the C/A file is ideal for simulation. The C/A handler is the driving device in the simulation runs, which can take place as the development work progresses. Modification of the DPS after cutover can be tested by the simulation capability of the AC, before they are implemented.
4. The polling lists in the C/A handler allow easy tuning of the application and imposing the processing rhythm of the application on the DPS.
5. The general case of a BP is described in the C/A file, while the individual case, i.e. the occurrence of a BP, is represented in the status file. This distinction facilitates the comprehension of the application by a user and a system analyst.
6. It is left to a user to specify the states in a BP which he wants to be controlled by the AC. Although it might be argued that this liberty may disconcert a user, it is our belief that it is a real asset in adjusting the DPS to the user's requirements: in a crucial application it may be wished that many states should be tested by the AC; on the other hand, in a less important application only a few states may be controlled by the AC. As importance is subjective and prone to changes, the AC allows a smooth transition from a model with few states to one with many states and back.
7. The concept of BP is openended: this is due to the similarity between a BP and an action. An action may be made a BP; a BP may be enlarged by an action; two BP's, put together, result in a BP again. This property helps a user change his application and reduces the number of routines which process BP's and actions in the AC.
8. The AC improves the synchronisation between the BS and the DPS by pinpointing business states in the C/A file and supervising them by the C/A handler. Through these charac-

teristics the AC goes beyond a real time system and introduces a new DPS class: the state triggered DPS.

7. Final remark

Although the AC has been presented here as a system with real time capabilities both during application design (the application is being developed interactively) and operation (the C/A handler makes the AC an abstract machine), the concept of the AC can be 'flattened out' to a batch system. In this case the states would not be tested real time, but in special batch runs.

Acknowledgement

The author wishes to thank Professor A. S. Douglas for the many valuable discussions.

References

BUBENKO, J. (1973). Contributions to Formal Description, Analysis and Design of Data Processing Systems, Ph.D. Theses, Stockholm.

CODASYL DEVELOPMENT COMMITTEE (1962). An Information Algebra, *CACM*, Vol. 5, No. 4, pp. 190-204.

DIJKSTRA, E. W. (1969). Complexity Controlled by Hierarchical Ordering of Function and Variability, in *Software Engineering*, ed. by P. Naur and B. Randell, Nato Science Committee, Brussels.

GRINDLEY, C. B. B. (1966). Systematics—A Nonprogramming Language for Designing and Specifying Commercial Systems for Computers, *The Computer Journal*, Vol. 9, No. 2, pp. 124-128.

GRINDLEY, K. (1975). *Systematics—A New Approach to Systems Analysis*, McGraw-Hill, London.

LANGEFORS, B. (1966). *Theoretical Analysis of Information Systems*, Studentlitteratur, Lund, Sweden.

NUNAMAKER, J. F. (1971). A Methodology for the Design and Optimization of Information Processing Systems, *SJCC*, AFIPS.

TEICHROEW, D. (1971). Problem Statement Analysis: Requirements for the Problem Statement Analyzer, ISDOS Working Paper No. 43, University of Michigan, Ann Arbor.

YOUNG, J. W., and KENT, H. K. (1958). Abstract Formulation of Data Processing Problems, *Journal of Industrial Engineering*, Nov-Dec 1958.

Book review

Algorithms in SNOBOL4, by J. F. Gimpel, 1976; 487 pages. (*John Wiley*, £10·60)

J. F. Gimpel has ten years experience of programming in SNOBOL4 and was responsible for the SITBOL implementation for the PDP10. This is an excellent book on software engineering techniques and applications, even in comparison with Griswold's 'String and List Processing in SNOBOL4,' or 'Software Tools,' by Kernighan and Plauger. Gimpel deals with much the same applications as both of these, but differs from the latter in choosing SNOBOL4 as the medium of communication for describing the algorithms.

Some knowledge of SNOBOL4 syntax is necessary to understand the algorithms, in spite of the copious comments. Fortunately, the language has a simple structure, which is very easy to learn. In theory an algorithm is independent of the particular language in which it is expressed, but in practice the facilities available in the language exercise an unavoidable influence on the techniques employed. SNOBOL4 is comparatively rich in facilities, but its forte is pattern matching and string operations, an area in which most other languages are weak. This makes quite a difference to the presentation of the algorithms.

As an example of the kind of technique possible with SNOBOL4, take the three statement procedure for converting Arabic numerals to Roman, which is done entirely by string manipulation and pattern matching. Or consider the method by which a facility equivalent to the FORTRAN statement function is added to the language; the execution time code compilation and function definition facilities enable this to be done in four statements.

Some techniques have not been described before. Examples are syntax-directed compilation by means of semantic routines embedded as unevaluated expressions in patterns, dynamic loading and compilation of external functions stored in source form, and determining the statement numbers of statements compiled at execution time.

Most of the algorithms are much more concisely expressed in SNOBOL4 than they could be in another language. Strachey's general macro processor, for example, takes approximately forty statements. It is a measure in fact of the power of SNOBOL4 that so much has been packed into this volume. There are functions for various conversions, string and array manipulations, list processing, document formatting, pattern construction, and input/output. It is useful to have numerical functions such as SQRT and the trigonometric functions. The subjects of sorting, permutations, and stochastic strings are also covered. One chapter deals with games, including a complete poker program using a previously unpublished algorithm. There is also an assembler, a compiler, and a macro processor.

Besides the algorithms themselves there is a discussion of their theoretical background and performance, where appropriate. A whole chapter is devoted to the theory of SNOBOL4 patterns, and another to their implementation. Other aspects of SNOBOL4 implementation appear in a separate chapter, with functions for collecting timing information. There are exercises with every chapter, and solutions are provided to half the questions.

The book has been machine-formatted by a sophisticated version of one of the programs it describes. Its appearance is quite acceptable, but could have been improved with computer typesetting. There are a number of misprints in the text, but not in the algorithms themselves, which have all been tested.

SNOBOL4 has its disadvantages, as the author himself points out. Perhaps the greatest is the lack of control structures, which is only partly alleviated by the iteration implicit in pattern matching. And in the present vogue for structured programming SNOBOL will not commend itself to everyone by permitting two gotos on every statement.

Nevertheless these algorithms demonstrate that well structured programs can be written in SNOBOL4 by adopting a modular approach in conjunction with a disciplined use of the language. The significance of this book lies not only in the functional building blocks which it supplies but also in its methods of interfacing modules, which deserve to be adopted as standard conventions by the SNOBOL4 programming community.

A. SHAW (London)