ISSN 0254-2757

QI QUÆSTIONES INFORMATICÆ

Volume 5 • Number 1 April 1987

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereeneging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Subscriptions

Annual subscription are as follows:

	SA	US	UK
Individuals	R10	$ 7	£ 5
Institutions	R15	$14	£10

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

THE PROTOCOL SPECIFICATION LANGUAGE ESTELLE

Jan Roos
Computer Science Department,
University of Pretoria

ESTELLE is being developed by the International Standards Organisation (ISO), Technical Committee (TC) 97, Sub-committee (SC) 21, Working Group (WG) 1, Formal Definition Technique (FDT) Subgroup B to fulfil the need for a protocol specification language. FDT Subgroup C is working on LOTOS as an alternative protocol specification language and the International Telegraph and Telephone Consultative Committee (CCITT) has developed SDL for the same purpose.

The purpose of this paper is to briefly introduce the language ESTELLE and to discuss the following:

— The characteristics of the language.
  • The finite state machine orientation of the language.
  • The formal semantics of the language constructs.
  • The levels of abstraction provided by the language.
  • Features enhancing protocol specification and verification.
— The current level of maturity of the language.

Some indications of the limitations of the language are given and the appendix contains a very simple skeleton example of an ESTELLE specification.

1. INTRODUCTION

The need for formal description techniques (FDTs), to be used for protocol specification, has long been recognised by the international standards community. The International Telegraph and Telephone Consultative Committee (CCITT) produced SDL (Z.101 — Z.104) [1] as early as 1980 and the first FDT meeting of the International Standards Organisation (ISO) was also held in 1980 [2]. FDTs are directed towards a number of goals. For example:

- to specify a protocol in an unambiguous, clear and complete way,
- to provide a basis for protocol analysis, verification, conformance testing and implementation.

The FDT field is increasingly popular amongst computer scientists and a fair amount of research has been done over the last 5 years. The FDT produced by ISO Technical Committee (TC) 97, Sub-committee (SC) 21, Working Group (WG) 1, FDT Subgroup B is ESTELLE. FDT Subgroup C is working on LOTOS as an alternative. ESTELLE is based on an extended finite state machine concept and LOTOS on the temporal ordering of interaction primitives. Both these FDTs have already been circulated in draft proposal form and are very close to completion.

An FDT provides a way for describing the set of observations that can be made of the system specified. In principle, any well defined language could be used for specification (e.g. Temporal Logic, ESTELLE, LOTOS, SDL, CHILL, ADA, assembler). Programming languages generally have a limited degree of appropriateness to the protocol specification task. Special purpose FDTs like ESTELLE, LOTOS and SDL were therefore developed to provide for this need.

ESTELLE, LOTOS and SDL are languages which can be interpreted mechanically whereas implicit FDTs, like Temporal Logic, do not give an explicit model of the system being specified. Instead, a specification is expressed in terms of properties and invariant conditions of the system [3].

There are many possible specifications of a system. These specifications can be at different levels of detail. The more detailed a specification becomes the less freedom is left to the implementer of the specification. Ultimately a specification should not provide more detail than is absolutely required to fully specify the functionality of the system. Such a specification leaves maximum freedom of implementation.

This paper discusses the FDT ESTELLE and highlights its virtues as a protocol specification technique. The paper concludes with a brief skeleton example of the use of the language in Appendix A.

Quæstiones Informaticæ, Vol 5 No 1, PP 51-62, ISSN 0254-2757

2. THE LANGUAGE ESTELLE

ESTELLE is an extension of a subset of PASCAL which allows the components of a data communication protocol to be modelled as a hierarchy of modules each of which is specified as an extended finite state machine. Because of its PASCAL nature it is generally more acceptable to most protocol experts than is LOTOS, which is more mathematical in nature.

The language enforces a general specification structure which results in a fairly readable specification. The ability to partition a specification into nested modules and to interconnect them through channels allows for reasonable flexibility in modelling a protocol. Each of these modules forms a separately compilable unit.

In ESTELLE the module at the highest level is the specification itself. The specification can be one module or can be refined into a set of nested modules.

The general structure of a module is:

```
Header
   Parameter list
   Interaction point list
   Export variable list
end;
Body
   Declaration-part
      constant
      type
      var
      procedures and functions
      channel definitions
      module definition
      internal interaction point definitions
      state set definition
      use clause parts
   Initialisation-part
   Transition-part
   Termination-part
end;
```

A module consists of a header and one or more body-parts. This facility is useful to support different protocol classes from the same header. The module header may define a formal parameter list used to pass parameters to a module when it is initialised and may define a list of export variables to be shared between parent and child modules. The module header will also include a list of all the module's interaction points with the outside world. Each interaction point is a full duplex interface and is connected to an interaction point in another module through a channel. Interaction points, parameters and export variables are specified as lists in the module header definition and are similar to the formal parameter lists found in function and procedure headings of PASCAL.

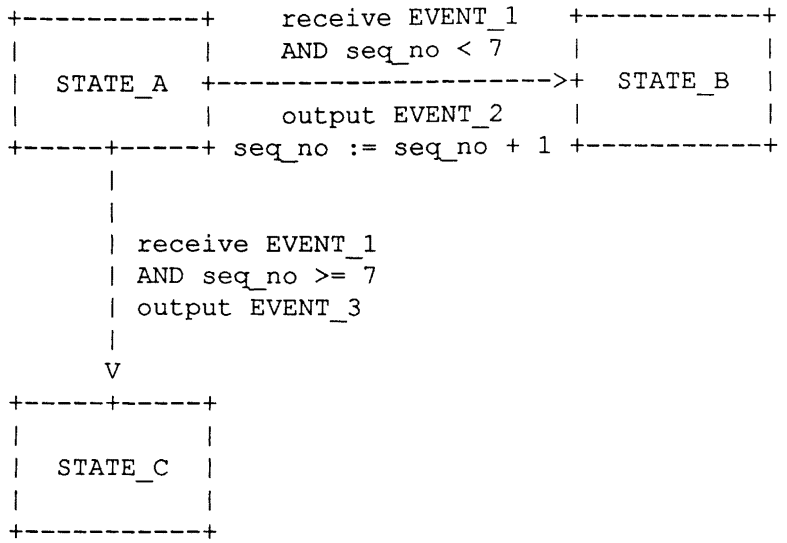
All parts of a body are optional. The declaration-part consists of parts such as constant and type definitions, variable declarations and procedure and function definitions, which are derived directly from PASCAL. Additional parts unique to ESTELLE are the channel and module definitions, internal interaction point definitions, state set definition and use clause parts. The module definition part can contain further child modules or reference them as external. (A module can be defined as a process or an activity. Processes may run in parallel with other processes on the same level of the hierarchy whereas activities which are children of the same parent may not run in parallel. The purpose of this distinction is to resolve synchronisation issues.) The state set definition defines all the states to be used in the Finite State Machines (FSM) and the use clause provides access to the export variables of child modules.

The initialisation-part and termination-part of a module are used to specify procedures to be executed automatically during creation of an instance of a module or during the termination thereof. The initialisation-part also sets the initial state of the module's FSM and connects the module's interaction points with other external or internal interaction points. (An interaction point is external to a module if it is defined in the module header and it is internal to a module if it is defined in the declaration part of that module. The attach operation is used to connect external

interaction points of a parent module to the external interaction points of a child module whereas the connect operation is used to connect all other interaction points.)

The transition-part is really the heart of the module and is used to represent the FSM of the module. Each module's FSM reacts to events received through channels from other modules. A FSM is specified in the following way:

In ESTELLE the FSM diagram:



is specified as:

```
from STATE_A                          (* Transition 1 *)
  when EVENT_1
    provided seq_no < 7
      to STATE_B
        begin
          output EVENT_2;
          seq_no := seq_no + 1;
        end;
    provided seq_no >= 7              (* Transition 2 *)
      to STATE_C
        begin
          output EVENT_3;
        end;
```

The from-clause specifies the current state, the when-clause the event, the to-clause the next state and the provided-clause specifies a boolean expression which must be true for the transition to take place. The from-, to-, when- and provided-clauses can be used in any order and can be nested as shown above. An example of such an FSM implementation can be seen in Appendix A.

This section very briefly introduced some of the structural aspects of the ESTELLE FDT and the interested reader is referred to references [4] and [5] for more detail.

3. CHARACTERISTICS OF THE LANGUAGE

Generally the language was designed to favour the specification of sound, well structured, verifiable protocols. Although the eventual implementation of the protocol was not supposed to be of major importance, it is important that the concepts of the protocol are conveyed in the clearest possible way and that the resultant specification should enhance its analysis and verification. In order to reach these goals the following are some of the major characteristics of the language:

- The finite state machine orientation of the language.
- The formal semantics that exist for all language constructs.
- The support of levels of abstraction of the protocol.
- The features enhancing protocol specification and verification.

3.1 The finite state machine orientation of the language

For many years a finite state machine (FSM) approach has been used for modelling communication protocols. Especially in the areas of protocol verification and implementation FSM techniques are often used. It was therefore natural for a specification technique to use the FSM modelling approach.

The FSM basis of the language obviously makes it very useful in FSM-based protocol verification exercises, such as global state exploration. The FSM basis is also easily understood by the protocol implementer and although a protocol specification does not suggest any implementation detail, the FSM structure enhances the portability of the specification to an implementation.

3.2 Formal semantics of ESTELLE language constructs

The formal semantics of ESTELLE language constructs were not present in the first draft proposal of the language but its importance was soon recognised and they are now part of the language.

Formal semantics are required in order to provide a consistent language definition and to clearly determine the expressive power of the language. This provides for the proper definition of the language itself and forms the basis for the analysis and testing of a specification.

ESTELLE uses a subset of PASCAL as basis and extends this basis with the necessary statements to support its requirements. The full PASCAL subset as well as all additions are fully defined through a meta language. In doing so various proposed ESTELLE constructs or statements had to be changed, or even removed, because of the complexity involved in the definition of their semantics.

3.3 Support of levels of abstraction of the protocol

It is very important that a protocol specification technique allows the specifier to specify only issues of importance at the particular level of the specification. It thereby hides detail that is not of importance at that level. This divide-and-conquer technique implies that it should also be possible to fully specify all relevant components or sub-components of a protocol and to specify their interactions. The components or sub-components in ESTELLE are the modules as discussed earlier.

Experienced protocol specifiers/implementers will appreciate the importance of the explicit definition of all communication channels interconnecting the different modules and the outside environment. All interactions on each channel and their flow directions are also specified. Because of the dynamic nature of a protocol structure it is important to specify the effect of interactions to be queued or already queued, both for new instantiations and for terminated instantiations of modules.

The danger of too much refinement is that it tends to provide unnecessary detail which restricts clarity and freedom of implementation.

3.4 Features enhancing protocol specification and verification

Most of the features of ESTELLE already mentioned, like its finite state orientation, its rigid structure, its well defined interaction points and channels, etc. enhance sound 'programming' techniques when specifying protocols. These features generally enhance readability, maintainability as well as verifiability.

Other rules which serve the purpose of enforcing sound 'programming' practices are:

- The strong typed environment of PASCAL is retained providing many static specification errors to be discovered by the compiler.
- The fact that the major state of a module's FSM can only be changed by one construction namely the to-clause. This prevents the specifier from changing the major state in awkward places not clearly 'visible' to the reader, and it also simplifies protocol verification through reachability analysis.
- The fact that no interactions may be specified in a module's termination part. This prevents the specifier from finally specifying 'unnoticed' interactions in some 'hidden' portion of the specification.
- Any function referenced in the predicate contained in a provided clause must not have side effects. If the values of variables could be changed by functions referenced in predicates and if all the criteria for selecting one or more transitions were not satisfied during the selection process, then these side effects could influence the selection of transitions in an unpredictable or 'hidden' way. This feature prevents the specifier from specifying transition criteria in a non-obvious or faulty way.

4. THE LEVEL OF MATURITY OF ESTELLE

The second draft proposal on ESTELLE was scheduled for completion in June 1986. Because this draft proposal will include most of the comments made on the first draft proposal it is expected to be accepted.

Already many groups throughout the world are working on syntax checkers, compilers, test facilities, protocol design tools, simulation facilities, verification facilities, etc. all based on ESTELLE. Such products, although initially aimed at a specialised environment, are also soon expected on the market.

5. ESTELLE LIMITATIONS

A question that arises is whether ESTELLE will fully provide for all the needs of a protocol specification FDT. Although it is still too early to definitively comment on this issue some initial concerns may be mentioned:

- Because SDL, the FDT of CCITT, is also based on the FSM model but uses a different language, it is felt that ESTELLE is not sufficiently different to justify its existence.
- Another concern is the length of ESTELLE specifications. It is generally felt that a specification of excessive length prohibits a clear understanding of its contents and is consequently error-prone and difficult to use for verification and testing.
- Because ESTELLE provides for only the FSM modelling concept and it is modelled so closely to specific implementation techniques, it is true to say that it restricts the specifier to a specific approach and can be regarded as an implementation-oriented style of specification. In this regard it has been suggested that ESTELLE should be regarded as an implementation description technique, instead of a specification-oriented formal technique.

6. CONCLUSION

The need for formal protocol specification languages is no longer contested. It is often concluded that the mere process of specifying a protocol in an FDT is in itself worthwhile, because it helps the specifier to fully understand the protocol being specified.

The development of ESTELLE is certainly a major accomplishment and it will be very useful in the data communication arena. It is however also clear that ESTELLE has its limitations and it should therefore be applied only where suitable. It should be accepted that other FDTs like LOTOS, etc. do play an important complementary role.

REFERENCES

1. Dickson Gary J., Chazal Pierre E., [1983], Status of CCITT Description Techniques and Application to Protocol Specification., *Proceedings of the IEEE*, **71**, 12, December.
2. Vissers Chris A., Tenney Richard L., Bochmann Gregor V., [1983], Formal Description Techniques., *Proceedings of the IEEE*, **71**, 12, December.
3. Hailpern Brent T., [1982], *Verifying Concurrent Processes Using Temporal Logic*. Lecture notes in Computer Science 129, Springer-Verlag.
4. Linn Richard J. Jr., [1986], A Tutorial on the Features and Facilities of ESTELLE. ISO/TC 97/SC 21/WG 1 Project 20, UMASS 2, Febr. 25.
5. ISO/TC 97/SC 21, [1986], Information Processing Systems — Open Systems Interconnection — ESTELLE — A Formal Description Technique based on an Extended State Transition model. Draft, UMASS 1, ISO/TC 97/SC 21 N DP9074, March 13.

APPENDIX A

ESTELLE example

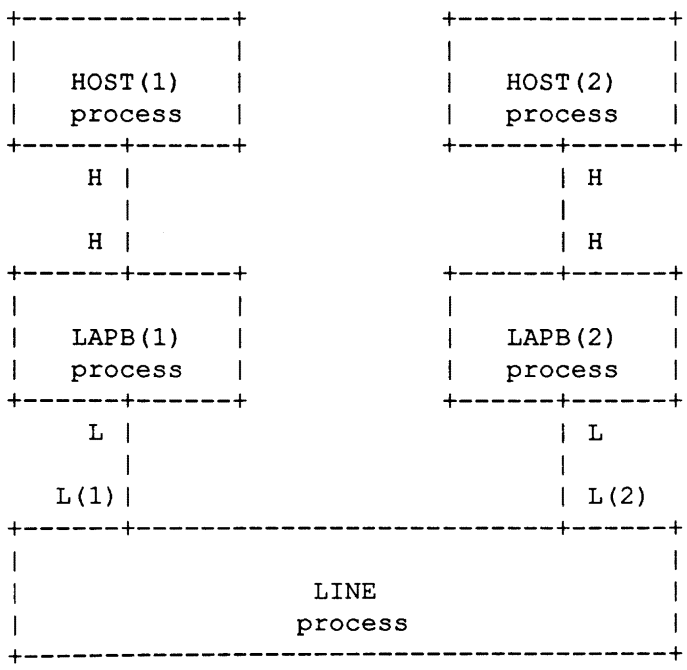
A portion of a subset of the layer 2 protocol of X.25, namely LAPB, will be specified as example.

LAPB is a variation of HDLC and is a data link protocol used in a point-to-point environment. A very basic knowledge of LAPB is assumed.

The subset that will be specified allows only for the transmission of the frame types SABM, UA, I RR and RNR.

A.1 The structure of the LAPB specification

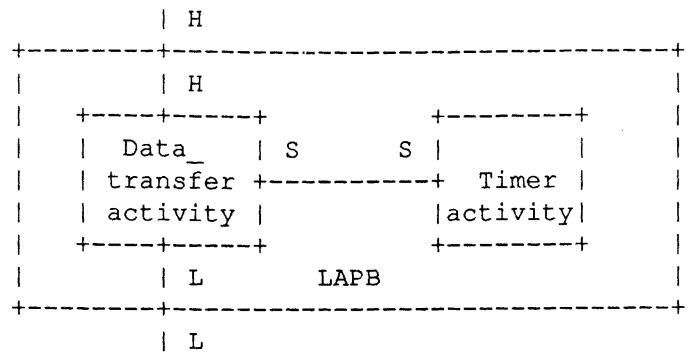
A.1.1 The environment structure



Each LAPB process has a H interaction point to interface to the HOST process (which implies the higher layer protocols) and a L interaction point to interface to the LINE process. Each HOST process has a H interaction point and the LINE process has two interaction points, L(1) and L(2), each to a LAPB process.

A.1.2 LAPB structure

Each LAPB process can be substructured as follows:



LAPB is substructured into a data_transfer and timer component, each with their shown interaction points.

A.1.3 Main specification structure

The main specification is a special type of module and its structure is shown below:

```
Specification (Name)
  Body
    Declaration-part
    OPTIONAL Initialisation-part
    EMPTY Transition-part
    NO Termination-part
  end;
```

A.1.4 Module structure

The reader is referred to section 2 of the paper for an example of the module structure. All modules are defined in the declaration-part of the main specification or in the declaration-part of other modules.

A.2 The specification of LAPB

A.2.1 Main specification

```
Specification LAPB;

  const
    window                  = 7;

  type
    entity_no_type          = 1..2;
    bit                     = 0..1;
    byte                    = 0..255;
    message                 = string;
    sequencenr              = 0..window;
    FrameKind               = (I,RR,RNR,SABM,UA);
```

```
    frame = record
      address:        byte;
      ControlField:   record
        kind:              FrameKind;
        seq:               sequencenr;
        NextFrameExpected: sequencenr;
        pfbit:             bit;
      end;
      info:           message;
    end; (* frame *)

  (* Notice that all messages that can flow on each   *)
  (* channel and the direction of flow are specified. *)

  channel H_channel(user,provider);
    by user:
      HostMessage_event(m: message);
      Busy_event;
      NotBusy_event;
    by provider:
      response_event(m: message);
      DisableHost_event;
      EnableHost_event;

  channel L_channel(user,provider);
    by user:
      request_event(f: frame);
    by provider:
      FrameArrival_event(f: frame);
      ChksumErr_event;
      InvalidFrame_event;

(* Module header definitions.                          *)
(* Note that each of these module headers has a        *)
(* corresponding module body and the interaction       *)
(* points listed in the module header are external     *)
(* to each particular module.                          *)

module HOST_type process(host_id: entity_no_type);
  inter H: H_channel(user)      common queue;
end;

module LAPB_type process(lapb_id: entity_no_type);
  inter H: H_channel(provider) common queue;
        L: L_channel(user)     common queue;
end;

module LINE_type process;
          inter L:array[entity_no_type] of L_channel(provider)
                                common queue;
end;

(* Module body definition.                             *)
(* All modules are defined as external and they are    *)
(* specified in later sections.                        *)

body HOST_body for HOST_type; external;

body LAPB_body for LAPB_type; external;
```

```
body LINE_body for LINE_type; external;

var  (* Declares all processes in the system. *)
  HOST: array[entity_no_type] of HOST_type;
  LAPB: array[entity_no_type] of LAPB_type;
  LINE: LINE_type;

(* Initialisation-part. *)

initialise
  begin
    (* Instantiate all active processes. *)
    init LINE with LINE_body;
    all i: entity_no_type do
      begin
        init HOST[i] with HOST_body(i);
        init LAPB[i] with LAPB_body(i);
        (* Interconnect the processes as required. *)
        connect HOST[i].H to LAPB[i].H;
        connect LAPB[i].L to LINE.L[i];
      end;
  end;

end. (* Specification. *)
```

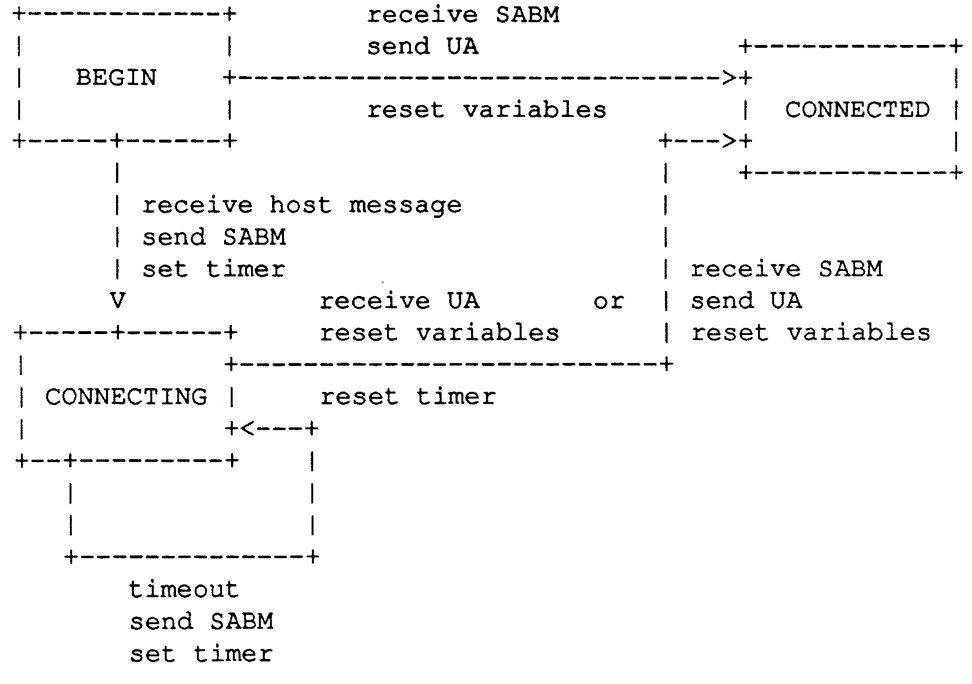
A.2.2 LAPB body specification

```
body LAPB_body for LAPB_type;

  const
    t1 = ...;
  type   ...;
   (* The ... indicates: to be  provided  at  implementation time. *)

  channel S_channel(user,provider);
    by user:
      set_timer_request_event;
      reset_timer_request_event;
    by provider:
      timeout_response_event;

  (* Module header definition. *)

  module timer_type activity(timeout_time: integer);
    inter S: S_channel(provider) individual queue;
  end;

  module data_transfer_type activity
                 (data_transfer_id: entity_no_type);
    inter H: H_channel(provider) common queue;
          L: L_channel(user) common queue;
          S: S_channel(user) individual queue;
    end;
    (* The interpretation of common and individual queues    *)
    (* as defined above is: the H and L channel interactions *)
    (* will share the same common queue while the S channel  *)
    (* interactions will have their own  individual  queue.  *)
```

```
(* Module body definition. *)

   body timer_body for timer_type;                  external;
   body data_transfer_body for data_transfer_type; external;

   var
     data_transfer: data_transfer_type;
     timer: timer_type;


   (* Initialisation-part. *)
   initialise
     begin
       (* Instantiate all active activities. *)
       init data_transfer with
            data_transfer_body(data_transfer_id);
       init timer with timer_body(t1);

       (* Interconnect the activities/processes as required. *)
       connect data_transfer.S to timer.S;
       attach H to data_transfer.H;
       attach L to data_transfer.L;
     end;

end; (* LAPB_body *)
```

A.2.3 Data-transfer body specification

The following FSM performs the connection function of the LAPB subset and will be illustrated in the specification of the data transfer module.



factor between males and females. From Table 5 it can be observed that the reduction in these differences is primarily due to strong changes in attitude within the male group. Male subjects showed a greater negative change on the first factor on first contact with computers and a strong increase on the negative views of computing between the second and third surveys. The male subjects evidently had higher expectations of the value of computers which were nullified by hands-on experience.

	Short Term (1st/2nd Survey) n = 27		Rest of Course (2nd/3rd Survey) n = 22		Long Term (1st/3rd Survey) n = 26	
females	$\bar{X}$	t	$\bar{X}$	t	$\bar{X}$	t
Fact or 1	-.23	1.55	-.09	0.62	-.31	2.17*
Factor 2	-.14	0.84	+.12	0.69	-.08	0.15
Fact or 3	-.57	3.11**	+.13	0.65	-.46	2.01*
Factor 4	+.28	1.65	-.13	0.75	+.08	0.45

	Short Term (1st/2nd Survey) n = 65		Rest of Course (2nd/3rd Survey) n = 62		Long Term (1st/3rd Survey) n = 74	
males	$\bar{X}$	t	$\bar{X}$	t	$\bar{X}$	t
Factor 1	-.52	4.47**	-.05	0.44	-.46	4.03**
Factor 2	-.21	1.37	+.43	3.53**	+.13	1.18
Factor 3	-.40	3.16**	-.01	0.03	-.33	2.39*
Factor 4	+.23	2.00*	+.08	0.81	+.24	2.86**

* : $p < 0.05$
** : $p < 0.01$

table 5

Attitude Change in Males and Females

CONCLUSIONS

As noted in the introduction, attitude can play a significant part in the successful implementation and use of computer systems. Kerlinger [9:495] defines attitude as "...an enduring structure of beliefs that predisposes the individual to behave selectively towards attitude referents" and it is probable that attitudes formed by first contact with computers within an educational setting could well influence later acceptance and use of computers in business.

This study identified four components in the attitude towards computers construct, the structure of which agrees with earlier work. Although the dimensions extracted in a factor analysis are dependent on the items available in the survey instrument, it is apparent that attitude towards computers consists of both negative and positive aspects. For this reason, a simple pro-con attitude scale is unlikely to provide sufficient insight for studies of the role of attitudes in computer use.

The lack of a control group in the study of attitude change makes it impossible to confirm causal relationships and these findings should be considered as possible indicators within the context of a case study. Although a control group would have been eminently desirable, finding a suitable group of similar students and providing the course input without hands-on computer use is not feasible within the undergraduate commerce course structure. Discussions with the students involved suggest that actual computer use is a major factor in attitude change, but there is no empirical evidence to directly support this. The strong negative changes in attitude observed in this student sample are perturbing from an educational viewpoint. It would appear possible that hands-on use of computers, even with the more user-friendly systems, could have significant negative influence on an individual's perception of the value of computers. If such direct contact is required, it might be necessary to consider new approaches to cushion the computer shock experienced by novices or, at least, provide as gentle an introduction as possible. The reversal of changes in the negative dimension (Factor 2) suggests that the teaching of conventional programming languages such as COBOL might be counter-productive. Most aspects of commerce do not require such programming ability and, unless students intend to follow careers in computing, it could be that introductory computer courses for business students

```
      to CONNECTING
        when S.timeout_response_event
          begin
            f.ControlField.kind = SABM;
            output L.request_event(f);
            output S.set_timer_request_event;
          end;

  ....

end; (* data transfer body. *)
```

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review artilces and exploratory articles of general interest to readers of the journal. The preferred languages of the journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:

Prof. G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0105
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. Manuscripts produced using the Apple Macintosh will be welcomed. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name and affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil and the authors name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Drawings etc., should be submitted and should include all relevant details. Photographs as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:

1. Ashcroft E. and Manna Z., The Translation of 'GOTO' Programs to 'WHILE' programs., *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255, 1972.
2. Bohm C. and Jacopini G., Flow Diagrams, Turing Machines and Languages with only Two Formation Rules., *Comm. ACM*, **9**, 366-371, 1966.
3. Ginsburg S., Mathematical Theory of Context-free Languages, McGraw Hill, NewYork, 1966.

Proofs

Proofs will be sent to the author to ensure that the papers have been correctly typeset and *not* for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Only orginal papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.