NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Fast Prototyping Of Communication Protocols

By

Mujeeb Haqqani

A M.C.S.. Thesis
Submitted to the School Of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of Master of Computer Science.

University Of Ottawa
Ottawa, Ontario, Canada

ISBN 0-315-60002-0

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

ACKNOWLEDGEMENT

The author wishes to thank his supervisor Dr. R. Prœbert for his guidance and instructions in this research. The author is particularly grateful to Dr. R. Probert for the patience and the understanding he showed towards the author who was a part-time student.

The author also wishes to thank his wife Meena Haqqani for her ongoing support and encouragement in completing this thesis.

ABSTRACT

In this thesis we present a methodology and a life-cycle system for rapid implementation of communication protocols. Our proposed approach includes a scheme for automated coding and decoding of protocol (single module) formal specifications in Estelle into (and from) a KnowledgeBase (KB). The KB rules and facts provide a single and useful authoritative reference during all phases of protocol development. The KB is used to interrogate the specifications, to generate an executable prototype of the protocol, and to develop a protocol conformance test plan.

An interactive experimental system, FPCP (Fast Prototyping of Communication Protocols), is designed and implemented to demonstrate the feasibility of this approach. FPCP provides an easy to use facility to aid in understanding and implementing protocol specifications by:

i) automatically generating an interrogatable KnowledgeBase representation of the specification, and

ii) automatically generating an executable prototype of communication software corresponding to the formal (single module, normalized) specifications in ESTELLE format.

In particular, FPCP first automatically encodes the ESTELLE specifications into a KnowledgeBase (KB). The FPCP system then creates a prototype of the protocol which is integrated with the KB. Once an executable prototype is created the FPCP system can then be used to:

- execute the protocol interactively;
- interrogate and debug protocol specifications;
- conduct user training;
- perform interactive testing of the protocol;
- validate protocol test cases, specified in TTCN format, for conformance to protocol specifications;

- validate test results against protocol specifications to assist in error diagnosis and correction.

In our trial application of the FPCP system , we have generated the KB from the formal Estelle specifications of transport(0) protocol. We also have generated an executable prototype of the transport(0) protocol from the KB, and validated the protocol on the FPCP system.

Our approach allows developers to maintain single, useful and authoritative "reference specifications" throughout the protocol development life cycle. The automated encoding of the specifications into a rule based KB allows users to take advantage of rule based system features [Mackert] and yet maintain the Estelle specifications as the single authoritative reference specifications.

Some possible extensions to FPCP system are also described in this thesis such as the ability to:

- develop protocol conformance test plan from KB;
- provide explanations of prototype behavior;
- interrogate the KnowledgeBase to
    - debug Estelle Specifications;
    - analyze IUT (Implementation Under Test) test results.

Fast Prototyping Of Communication Protocols

TABLE OF CONTENTS

ABSTRACT

ACKNOWLEDGEMENTS

1. INTRODUCTION TO PROTOCOL IMPLEMENTATION TOOLS AND TECHNIQUES

    1.1 Background
    1.2 Motivation For Fast Prototyping
    1.3 Related Work
    1.4 Objectives And Contributions Of The Thesis
    1.5 Thesis Outline

2. INTRODUCTION AND RATIONALE FOR A FAST PROTOTYPING APPROACH

    2.1 Overview Of Our Prototyping Directed Approach

    2.2 Protocol Specification In ESTELLE

        2.2.1 Normal Form Specification
        2.2.2 Syntax of Protocol Specification
        2.2.3 Channel Definition
        2.2.4 Other Estelle Components

    2.3 Automated Encoding And Use Of Protocol Specification In KB Form

2.3.1 Encoding Of Protocol Specification Into KB
2.3.2 The Scope Of Encoding From Estelle Specification
2.3.3 Encoding Estelle Specification - The General Concept
2.3.4 Interrogation Of Protocol Specification In KB

2.4 Development Of Executable Prototype From KB And Potential Uses

2.4.1 Development Of Prototype

2.4.1.1 The Design Of Prototypes in FPCP
2.4.1.2 The Execution Of FPCP Prototypes

2.4.2 Uses Of FPCP Prototypes

2.4.2.1 Debugging And Refinement Of Protocol Specification
2.4.2.2 Development Of Conformance Test Plan
2.4.2.3 Validation And Analysis Of IUT Test Results
2.4.2.4 User Training

3. FAST PROTOTYPING OF COMMUNICATION PROTOCOLS (FPCP): AN INTERACTIVE SYSTEM FOR PROTOCOL IMPLEMENTATION

3.1 FPCP System Design And Constraints
3.2 FPCP System Files And Objects
3.3 FPCP System Features And Limitations
3.4 Summary Of FPCP Design Rationale

4. AN EXAMPLE : TRANSPORT(0) PROTOCOL PROTOTYPING WITH FPCP SYSTEM

4.1 Automated Encoding of Transport(0) Into KB
4.2 Generating Transport(0) Prototype
4.3 Executing FPCP Transport(0) Prototype
4.4 Uses Of The Transport(0) Prototype

4.4.1 Debugging Transport(0) ESTELLE Specification
4.4.2 Validation And Analysis Of Test Results
4.4.3 User Training

4.5 Assessment Of Experience With Application Of FPCP to Transport(0) Protocol

5. EVALUATION AND CONCLUSIONS

5.1 Summary Of FPCP Approach

5.2 Suggested Extensions To FPCP System

5.2.1 Capture And Maintenance Of Prototype Execution Traces In KB
5.2.2 Interactive Interrogation Of KnowledgeBase
5.2.3 Development Of Conformance Test Plan And Test Cases
5.2.4 Conversion Of KB Into Estelle Specifications
5.2.5 Verifications Of TTCN Test Suites Against KB
5.2.6 Summary And Recommendations

5.3 Comparison Of FPCP to Related Approaches

5.4 Conclusions

6. REFERENCES

APPENDICES:

Appendix A - Sample Of Transport(0) Protocol Specifications In ESTELLE

Appendix B - Sample Of Transport(0) Test Cases In TTCN Format

Appendix C - FPCP System User Guide

C-1: System Design
C-2: How To Compile FPCP System
C-3: How To Start FPCP System
C-4: System Features
C-5: Sample Session

LIST OF FIGURES

Number	Description
Figure 2.1.1	FPCP Process Overview
Figure 2.2.1	Layers In OSI Model Of Communication Protocols
Figure 2.2.2	Interaction Points Between Two Entities
Figure 2.3.1	Estelle To KnowledgeBase Conversion
Figure 2.3.2	Pictorial View Of KnowledgeBase
Figure 2.4.1	Design Of FPCP Prototypes
Figure 2.4.2	Graphic View Of Protocol Execution
Figure 2.4.3	Debugging and Refinement of Specifications
Figure 2.4.4	Development Of Conformance Tests In FPCP
Figure 2.4.5	IUT Test Results Analysis With FPCP
Figure 3.1.1	FPCP Design Overview
Figure 3.3.1	FPCP Menu-Driven User Interface
Figure 5.2.1	Proposed Enhancements To FPCP
Figure 5.2.3	Verification Of TTCN Test Suites Against KB
Figure C-1	FPCP System Design Components

1. INTRODUCTION TO PROTOCOL IMPLEMENTATION TOOLS AND TECHNIQUES

1.1 Background

The International Standardization Organization (ISO) and CCITT have been developing and promoting standards for Formal Description Techniques (FDTs) for defining communication protocols. These standards for Open System Interconnection (OSI) have now became industry standards. One such formal description technique which has evolved is ESTELLE [ISO86a][Tenney]. ESTELLF is used for specification of both communications protocols and services. It is based on the extended finite state transition model.

The Tree and Tabular Combined Notation (TTCN) is a more informal description technique which is used for defining protocol conformance test suites. TTCN has recently been adopted by ISO as a draft international standard language for test specification [ISO86a] [ISO87d] [ISO87d].

Automated implementation of communication protocols from a set of formal specifications has been a subject of great interest over the last few years [Johnston] [Hansson] [Amalu] [Short]. Formal specifications tend to be precise and unambiguous but often lengthy and error prone.
However the development of accurate protocol specifications remain a time consuming and error prone task. At University of Ottawa, the Protocol Research Group has conducted studies to automate development of prototypes and generation of test cases from formal specifications [Boyce] [Probert88] [Amalu] [Short]. Similarly, for the implementer of protocols, automated tools have been developed to facilitate protocol testing and validation [Barbeau] [Bochmann] [Probert88] [Saqui] [Hansson] [Wiles].

In this thesis we present an approach (methodology) for fast prototyping of communication protocols. Our technique involves using a formal description technique, which is capable of documenting precise and unambiguous protocol specifications, to develop a KnowledgeBase (KB). The formal specifications remain the single authoritative source reference for the protocol development life cycle [Probert83]. The KnowledgeBase (derived from protocol formal specifications) provides the bases for developing an integrated set of tools to facilitate protocol development tasks. For example the KB can be used to develop an executable prototype, to provide a query facility against the source specifications, to generate test cases, and to validate conformance test suites against the specifications in the KnowledgeBase. We have designed and implemented an experimental computer system "Fast Prototyping Of Communication Protocols (FPCP)" to demonstrate the usefulness and the effectiveness of our proposed approach.

1.2 Motivation For Fast Prototyping

The objective of a formal description technique is to provide a mechanism for developing precise and unambiguous protocol specifications. Although the main objective has apparently been realized in many of these formal languages (such as ESTELLE, FAPL, and PDIL). In fact these languages have merely evolved into complex high-level programming languages. As a result the specifications coded are difficult to comprehend and debug. Nonetheless automated tools are under development to increase the usefulness of these Formal Description Techniques [Bochmann] [Hansson] [johnston] [Pappalardo] [Linn87] [Logrippo] [Probert88] [Saqui] [Vuong].

Automated Prototyping of communication protocols from these formal languages provides the developer a tool for testing and verification of specifications. The creation of a KnowledgeBase from the formal specifications of communication

facilitates automation of the protocol development life cycle processes. Furthermore these automated processes can be integrated by the use of one authoritative reference, namely the formal specifications of the protocol.

1.3 Related Work

Researchers at the University of Montreal have also developed a test design tool for functional analysis of communication protocols [Barbeau]. The tool generates protocol control data flow graphs and un-parameterized test sequences. These are generated by the tool from protocol formal specifications in Estelle. The generated control flow graphs defined the interaction sequences to be observed during testing. The tool was tested on an experimental basis on the File Transfer Access and Management (FTAM) protocol.

The Protocol Research Group at University Of Ottawa is very active in developing automated tools and techniques for implementation of communication protocols. In a recent study the researchers proposed a state transition-oriented approach to automatically generate a major component of standardized conformance test (CTS) suites [Boyce]. Under this approach formal protocol specifications are decomposed into behavior description modules. these modules are then used to generate and also verify conformance test cases against the specifications. The bulk of a standardized CTS can be derived from protocol state transition specifications in Estelle. The selective execution of protocol behavior description modules can be used to generate additional test cases and to validate protocol specifications.

In another related study the researchers at the Protocol Research Group are in a process of developing an interpreter to execute large LOTOS specifications [Logrippo]. LOTOS is an alternative standard formal description language for specifying protocols. The interpreter (ISLA - Inter-active

System For LOTOS Applications) first performs syntax and static semantics checks on LOTOS specifications. Subsequently ISLA automatically encodes the specifications into a prolog list. The Prolog list models the dynamic semantics specified in LOTOS. ISLA allows users to stepwise execution of LOTOS specifications. At each step the user is prompted with a menu of possible "next actions". At any point during simulation , a user can ask to see the current behavior which describes the current state of the LOTOS machine. These displays are in LOTOS format. The system maintains the internal representation of the machine transparently to users. The system also maintains a history of execution traces to provide users with displays to examine how the current state was reached.

In a joint project between Bell-Northern Research and University Of Ottawa researchers have designed and developed an integrated software environment, called the TTCN Workbench, for design and specification of conformance test suites in TTCN, and for managing the application and maintenance of conformance tests [Probert88]. The testing environment proposed includes tools to support:

- Importing and exporting test suites in TTCN notation;
- Editing of test suites in TTCN notation;
- Translation of TTCN test suites into machine processable format;
- Validation of TTCN test suites against formal specifications (Estelle or LOTOS);
- Interactive selection of test suite profile;
- Management of test results including test reports and analysis.

At University Of Ottawa , in an MSC thesis a technique for prototyping communications protocols in programming language C from protocol specifications in ESTELLE is presented [Amalu]. This approach involves manual coding of Abstract Service Primitives (ASPs) as procedures in C from the protocol specifications in Estelle. As a second step an automated

Estelle Compiler is generated. The parser for the Estelle compiler is generated from YACC (Yet Another Compiler Compiler) and the lexical analyzer generated by LEX which is a general purpose utility to generate lexical analyzers. The outputs of YACC and LEX are passed as inputs to a C Compiler which produces the code for the executable protocol prototype.

Also at University Of Ottawa, in another MSC thesis an Interactive Test Sequence Generator (ITSG) from logic specifications is developed [Short]. The logic specifications are coded in Prolog. ITSG provides users with a facility to control the traversal of execution paths and to generate test cases.

At the Swedish Institute Of Computer Science researchers developed an interactive TTCN (tree and Tabular Combined Notation) editor and executor (ITEX) [Wiles]. The tool provides designers a graphic oriented system to write, debug, and execute TTCN tests.

At the department Of Computer Systems , Uppsala University Sweden, a tool to translate protocol specifications into Pascal has been developed [Hansson]. The tool translates protocol specifications specified in ASYL/EFSM.

Within the scope of the European ESPRIT program researchers have developed an Estelle simulator prototype (ESTIM) [Saqui]. The prototype works with a subset of the Estelle language. The ESTIM simulation environment consists of:

- an Estelle compiler which performs specification parsing, static type-checking and translation into a machine language (ML) file;
- an interpreter to interpret the ML file;
- user interface to fire transitions in any one of the following four modes:
  1) step by step mode;

2) random mode;
3) user controlled mode;
4) "exec" mode (transition sequence stored in a file).

At Hewlett-Packard Laboratories work is being done to develop a graphical simulator for LOTOS (a formal description technique for communication protocols) [Johnston]. The system, SPIDER (Service and Protocol Interactive Development Environment) contains two types of interfaces with LOTOS. A textual interface to parse the specifications into a tree structure, and a graphic interface to edit the specifications. The SPIDER system performs static semantic checking of the specifications and develops a Labelled Transition System (LTS) to simulate the dynamic part. The simulation of LTS can be performed interactively or a pre-defined transition sequence is processed to guide the graphic simulation.

At the University of British Columbia, work has been done since 1984 to develop a an Estelle compiler [Vuong]. The latest version of ESTELLE-C accepts the 1986 version of Estelle. The compiler translates the protocol implementation dependent details into a compilable C program. In the second step the generated C program is compiled and linked with pre-defined run-time support, implementation independent C routines.

At the Institute of Computer Science and Technology of the National Bureau of Standards, USA, work is being done to to develop automated tools for generation of protocol test skeletons from protocol specifications in Estelle [Linn87].
One of the tools designed in France, LISE contains two modules. The first module (VADILOC) performs the validation of Estelle specifications. During this process it validates communication specifications between two entities, connectivity and existence of path between two states. This module creates descriptions of a FSM modeling the protocol.

The second module (GAST) uses expert system techniques to generate test sequences from the FSM descriptions generated by the first module.

Researchers at the Institute Of Electrical Communication, Tohoku, Japan, have developed a knowledgebased protocol development environment (IDESS) [Shiratori]. The IDESS environment consists of six components:

NESDEL: NESDEL is a protocol specification language. It uses graphic and program-like representation. It describes a protocol in terms of a Finite State Machine expressing the behavior of the protocol.

NESDEL Editor: This component allows users to create and modify NESDEL specifications of a protocol. This system also includes the specification syntax and consistency validation and a Help subsystem.

Protocol Validation System (EXPA): This system contains various algorithms to validate the specifications in NESDEL.

Knowledgebased Interactive Software Generator: This interactive system generates protocol detailed specification in IDL format (see below) from protocol NESDEL specifications.

Communication Software Specification Language (IDL): IDL is another specification language used in the IDESS environment. IDL is similar to Estelle language, but it supports hierarchical descriptions of state transitions. As mentioned above the IDL code is generated interactively from the protocol specification in NESDEL.

IDL Compiler: The compiler translates the IDL specification in machine language for protocol implementation.

1.4 Objectives and Contributions Of the Thesis

The main objective of this thesis is to present an automated approach to support fast prototyping of communication protocols. This approach will not only provide an executable prototype but also provide a base for automated support over all phases of communication protocol development [Mizuno].

In this thesis we describe and illustrate an easy to use automated process to encode (and decode) a protocol in Normal Form Estelle Specifications into a KnowledgeBase (KB). This allows us to maintain the formal specifications as the single authoritative "reference specifications" for the entire development life cycle. We also present a design for automated development of executable prototypes from the KB.

Our life-cycle support approach has most of the properties of simple knowledgebased systems:

- collection of rules and facts;
- execution engine;
& - explanation facility.

In our proposed design we maintain the advantages of rule-based systems and yet provide a fast and easy to use mechanism for generating a prototype for both training and protocol specifications development. We also have developed and implemented a system entitled "Fast Prototyping Of Communication Protocols (FPCP) to demonstrate the feasibility and to illustrate the main features of our proposed approach. Included in the FPCP system is a user friendly interface permitting interactive inquiry of the specifications, interactive execution of the prototype in Local or Remote modes, and trial execution of protocol tests in TTCN format on a file. Finally, a detailed application to a real protocol is

presented as a means of assessing the feasibility and usefulness of our approach.

Our design of the prototyping environment supports the integration of future protocol development tools. Such tools will be useful for interrogation and maintenance of KB of formal specifications, recording of protocol execution traces, development of conformance test suites, and verification of test suites (in TTCN format) against the formal specifications (in either KB form or in Estelle form).
Finally, our approach seems to be useful for incorporating rule-based techniques in fast prototyping. The pilot FPCP system illustrates the effectiveness of our approach in providing support for automated protocol implementation tools.

1.5 Thesis Outline

Chapter 2 presents our proposed communication protocol development life cycle approach. A brief discussion of the OSI reference model and protocol specifications in Estelle format are also included in this chapter. Our schemes for automated encoding of Estelle specifications into KB and the uses of the specifications in the KB are introduced as well. Finally, we include our proposed design of the FPCP environment and potential uses of the FPCP generated prototypes.

In Chapter 3 we describe the detailed design of the system FPCP (Fast Prototyping of Communication Protocols) which is designed and developed to demonstrate the feasibility and usefulness of the proposed approach. This chapter includes a brief description of the system design and a detailed review of system features.

In Chapter 4 we present an example of Transport(0) prototyping with FPCP system to illustrate our proposed approach and to demonstrate the usefulness of the FPCP system. In this example we used Transport(0) Normal Form Estelle specifications and we included the potential uses of the FPCP Transport(0) prototype.

Finally in Chapter 5 we discuss potential modifications to the FPCP system to incorporate all aspects of our proposed approach in an integrated system. This chapter also includes our evaluation of the proposed approach and the FPCP system.

2. INTRODUCTION AND RATIONALE FOR A FAST PROTOTYPING APPROACH

2.1 Overview Of Our Prototyping Directed Approach

The OSI reference model provides one common framework for protocol development standards. The OSI reference model consists of seven protocol layers namely:

Application Layer,
Presentation Layer,
Session Layer,
Transport Layer,
Network Layer,
Data Link Layer, and
Physical Layer.

This layering is the basic structuring technique used in the Open System Interconnection (OSI) Architecture [ISO86a].

Several Formal Description Techniques (FDTs) are appropriate to this model[Fleischmann]. ESTELLE is one such formal specification language [ISO86a] [Tenney]. An overview of our proposed approach to Fast Prototyping of Communication Protocols (FPCP) using a formal specification technique (ESTELLE) is shown in Fig 2.1.1.

Our approach requires first the development of the communications protocol specification in Estelle. The ESTELLE specification will subsequently remain the single authoritative reference for protocol implementation.

As a first step in FPCP , the ESTELLE specification is automatically converted into a KnowledgeBase (KB). The KB contains Facts and Rules derived from the ESTELLE specification. The KB design facilitates the design of capabilities for:

- interrogating the specification;
- developing an executable prototype of the protocol.

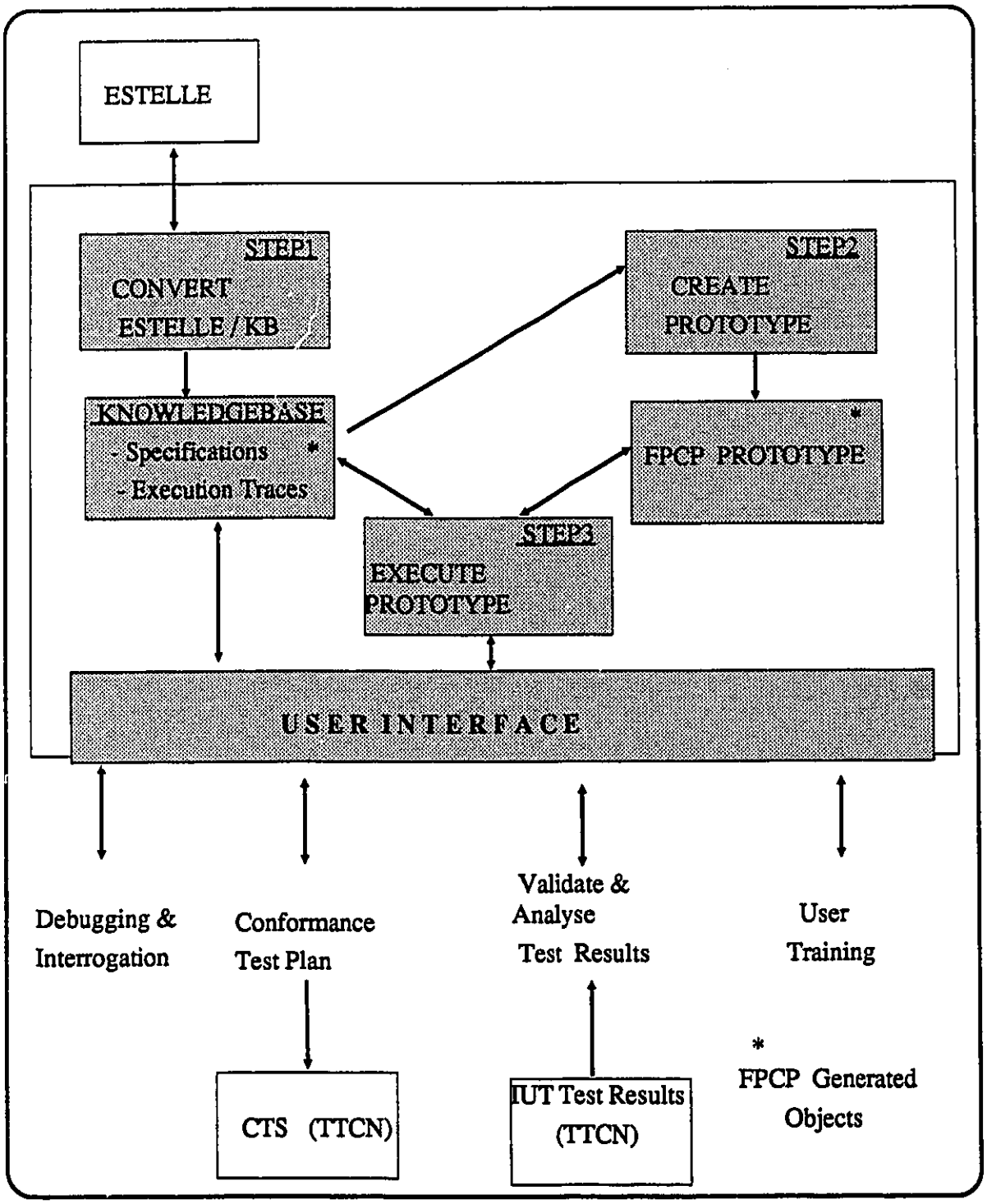


Fig: 2.1.1 FPCP PROCESS OVERVIEW

As a second step in FPCP, an executable prototype of the protocol can be automatically created from the KB. As shown in Fig. 2.1.1 this generated prototype is integrated with the KB. The execution of a protocol is controlled by a "User Interface" process. The "Execution Control" process interacts with the generated code (prototype) and the KB to execute the protocol. This entity also maintains an audit trail (traces) of executions. The audit trail is then added to the KB as additional facts. This integration allows sophisticated user capabilities. In particular, the KB and the interactive execution of the prototype provides users the ability to:

- Debug and refine the protocol specification;
- Develop a conformance test plan;
- analyze IUT test results;
- conduct protocol training sessions.

In the next section we present a brief overview of the Estelle protocol specification language. In section 2.3 we describe our approach for automated encoding of Estelle specification in KnowledgeBase (KB). This section also includes a discussion of potential uses of KB for debugging the protocol. Later in section 2.4 we present our scheme for automated generation of executable prototypes from KB.

2.2 Protocol Specification in ESTELLE

The OSI Reference Model as mentioned earlier views interconnection service providers as layers of intercommunicating modules (see Fig. 2.2.1). In any given implementation, these layers may not be physically present; however, the entire system must behave as though the layers are present.

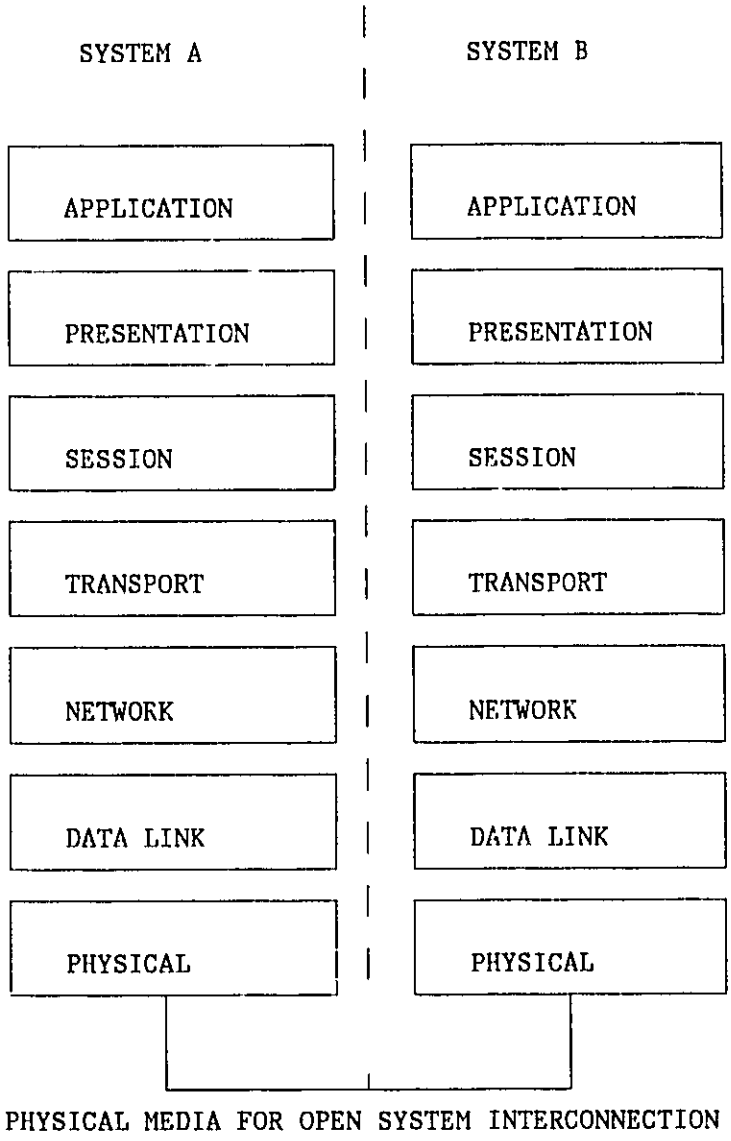


Fig. 2.2.1 : Layers in OSI Model Of Communication Protocols

The protocol entities in the same layer on different systems are called **peer entities**. Each entity utilizes the services of the layer below it. The communication between the peer entities is conducted using the higher layer protocol.

An **interaction point** between two entities is an abstract interface used to exchange data between the two entities (see Fig. 2.2.2).

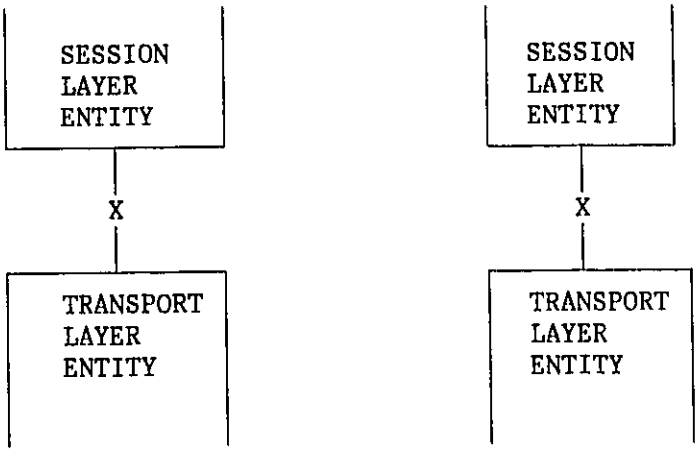


Fig. 2.2.2 Interaction Points (X)between Entities

A **channel** connects two directly (physically) adjacent modules and provides for bi-directional communication between them. The communication is in terms of interactions called **"Service Primitives"**. There are two categories of service primitives:

1 - **User Service Primitive** (input to the module);

2 - **Provider Service Primitives** (output by the module).

Protocol Specification in ESTELLE

Estelle is a Formal Description Technique (**FDT**) for specification of communication protocols and services. Estelle is based on the Extended Finite State Machine (EFSM) model [Linn] [ISO86a] [Tenney]. The Extended FSM model consists of one or more communicating modules and channels as described below.

Estelle is one of the three formal description techniques developed within ISO for use in describing OSI services and protocols. In Estelle, modules and channels are the fundamental building blocks. Modules may be decomposed into sub-modules. Each module is an EFSM with interaction points for inputs and outputs.

The modules are connected through channels. The channels form bindings from the outputs of one module to a potentially infinite queue associated with another module. Each channel has two ends which are attached at the interaction points of two modules. A module may initiate or receive interactions. Therefore, associated with each channel are two roles (initiator , responder) and a set of interactions is associated with each role.

Estelle provides an important language for design , analysis, and specification of protocols. As a formal language it is intended to provide complete, consistent, precise, concise, and unambiguous specification. Estelle supports decomposition of a system into smaller parts and the specification of the behavior of a part in terms of an extended finite state transition model.

2.2.1 NORMAL FORM SPECIFICATIONS

In Estelle a protocol may be specified in terms of externally observable behaviors of possibly more than one module. In general, a protocol may include inter-module interactions and relatively complex transition definitions. A protocol specification which has been transformed into single module specification is called a **Normal Form Specification** [Sarikaya] [Boyce] [Ural87].

A Normal Form Specification is a collection of **Normal Form Transitions (NFTs)** describing the externally observable behavior of a module (e.g. a protocol entity). An NFT consists of the following clauses:

- an optional WHEN clause specifying an external input interaction;
- a FROM clause containing a specific identifier for the present state;
- an optional PROVIDED clause specifying an enabling predicate which must be satisfied for the transition to take place;
- a TO clause containing a specific identifier for the next state;
- a BEGIN-END clause specifying a single path composed of assignment statements, procedure calls and possibly some output statements defining the external output interactions.

Throughout this thesis the initial Estelle specification is assumed to be in Normal Form.

2.2.2 SYNTAX OF PROTOCOL SPECIFICATIONS

A protocol specification in Estelle consists of the following parts:

CONSTANT DEFINITION
TYPE DEFINITION
CHANNEL DEFINITION
MODULE HEADER
MODULE BODY
MODULE VARIABLE DECLARATIONS
STATE DEFINITION
STATE SET DEFINITION
PROCEDURES AND FUNCTIONS

INITIALIZATION PART

TRANSITIONS DECLARATIONS

The Estelle syntax for CONSTANT, TYPE, VARIABLE, PROCEDURES and FUNCTIONS parts is identical to the syntax of the Pascal programming language. The other parts of an Estelle specification are discussed briefly in the following sections. A more detailed review of Estelle syntax, its features and facilities can be found in the references [Linn] [Tenney].

2.2.3 CHANNEL DEFINITION

This part identifies the module's interaction points. An interaction point is an abstract bi-directional interface through which a module may send or receive service primitives. This part also includes the definition of queuing discipline and module role (User / Provider).

EXAMPLE:

```
channel u-access-point (USER, PROVIDER);
        BY USER:
              t-connect-request (called-tsap-id :t-addr-typ);
                                (calling-tsap-id:t-addr-typ);
                                (exp-data-option:int-typ);
                                (qual-of-serv:int-typ);
        BY PROVIDER:
              t-connect-ind (called-tsap-id:t-addr-typ);
                            (calling-tsap-id:t-addr-typ);
                            (exp-data-option:int-typ);
                            (qual-of-serv:int-typ);
```

In the example above an interaction point (u-access-pt) is defined for the transport module. The interaction point defined takes the role of transport service user (BY USER) or the transport service provider (BY PROVIDER). In this example the Session is the user and the Transport is the provider. For each role of the interaction point the example contains a definition of a Service Primitive. The service primitive definition contains the specification of the parameters and their types. In the above example the transport connection request Service primitive (t-connect-request) requires four parameters:

called transport service access point;
calling transport service access point;
transport service data option (expedited data);
quality of service requested.

2.2.4 OTHER ESTELLE COMPONENTS

MODULE HEADER

This part defines the external visibility of a module in terms of its interaction points and exported variables.

EXAMPLE:

```
module transport0
        (U: u-access-point (PROVIDER) queued);
        (N: n-access-point (USER) not queued);
```

In the above example the Module Header part defines for the transport module two external interaction points used to exchange data with other modules (a user module and a network service provider module).

MODULE BODY

This part contains the definitions of module local variables, and procedures. Each module body definition and the associated MODULE HEADER definition constitute a single module definition. A module body definition may contain the following parts:

CONSTANTS
VARIABLES
STATES
STATE SETS
TRANSITIONS
PROCEDURES AND FUNCTIONS

STATES

The state definition part specifies the possible values of the control state variable.

EXAMPLE:

```
        state idle, wftr, wfcc, data;
```

STATESET

The stateset part provides a means to refer to several states at once.

EXAMPLE:

```
stateset waiting = (wftr, wfcc);
```

TRANSITIONS

The transition definition part specifies all possible instances of the module (Extended Finite State Machine). The syntax provides for specification of transitions enabling conditions and priorities. A transition procedure block provides a means of coding the required processing.

EXAMPLE:

```
trans from idle to wfcc when n.tcreq
                        provided tcreq.qts = ok-qts
                        begin
                             {processing code}
                             {for procedure block}
                        end;
```

In the above example of a transition of transport module, the transition is fired :

- if the current state is 'idle' and;
- when the transport connection request (tcreq) is received from the Network access point; and
- requested quality of service (tcreq.qts) is same as (a pre-defined) ok-qts.

When this transition is fired the transition procedure block containing the processing code is executed and the modules state is changed from 'idle' to 'wfcc' (waiting for connection confirmation). The transition block may include statements to output to other entities.

2.3 Automated Encoding And Use Of Protocol Specification

2.3.1 Encoding of Protocol ESTELLE Specification into KB

As stated earlier, a formal definition technique such as ESTELLE provides a means to develop protocol specifications which are concise and unambiguous. Recently, in several research studies, discussed earlier in Chapter 1, different approaches to development of automated prototypes from ESTELLE have been proposed and developed [Barbeau] [Saqui] [Hansson] [Amalu] .

In our approach the objective of encoding an ESTELLE specification into a KnowledgeBase is not only to develop a fast automated prototype of the protocol but also to provide automated facilities for software design activities during the entire development life cycle. All of these facilities are based on the single authoritative reference (the ESTELLE specification):

- debugging of the protocol specification;
- developing executable prototypes;
- developing a conformance test plan;
- conducting protocol training sessions;
- interrogating the specification;
- verifying protocol conformance test suites, and test results.

The ESTELLE specification is converted into a set of "facts" and Pascal "procedures" [Isreal] [Dahl] [Hayes85] [Hayes84]. Although some components of the Estelle specification can be encoded as "rules", in our approach, for simplicity of decoding KB into Estelle (reverse engineering), we encoded all Estelle components as "facts". The relationship between the original Estelle specification and the converted specification in KB is maintained such that the ESTELLE specification can always be derived from the KB, i.e. the KB derivation is reversible. This facilitates maintenance of the Estelle specification as the "single authoritative reference" specification [Probert83]. The KB internal structure remains transparent to the users. The user's

interface with the system is through international standard languages (ESTELLE and TTCN).
Thus our unique contribution to prototyping is the encoding of protocol specification into a KB (The KB remains transparent to the users) which allows:

- users to work with a single authoritative specification (Estelle). i.e. we do not propose a new syntax or format for protocol developers;
- users to take advantage of rule based systems derived automatically from the formal protocol specification (in Estelle).

Furthermore, our design of the generated executable prototype (which is a generated PASCAL program making use of the KB at run-time) provides added flexibility to integrate such future features as:

- run-time explanation of protocol behavior in terms of formal protocol specification;
- development of user friendly interfaces with the generated prototypes in high-level languages.

Our approach for conversion of Estelle specification into KnowledgeBase is illustrated on the following pages. The implementation of this scheme in the FPCP system involved some system limitations as well as extensions to the Estelle syntax. These are described below.

2.3.2 The Scope of Encoding From Estelle Specification

As stated earlier, a protocol single module (Normal Form) Estelle specification is first encoded into KB. Therefore the protocol Module-Header-Definition part is ignored during conversion of the Estelle specification. The Type-Definition part and the Procedure-Definition part are assumed to be a valid Pascal format. These parts are generally used as is in generating the fast prototype in Pascal. The Estelle Procedure-Body parts may include calls to an FPCP built-in procedure **OUT**(service-primitive-name) to output a service request.

Some extensions to the Transition-Definition parts are also included. In summary the following limitations and extensions to the Estelle syntax must be observed (many of these limitations are removable. Further discussion is presented in the final Chapter):

- Maximum length of an identifier is 12 characters;
- "trans" keyword is followed by a transition name identifier;
- "delay" clause in transition-part is not converted;
- Estelle Transition-Procedure part should use the FPCP built-in procedure **OUT** to output a service request on a channel;
- The Module-Header-Definition part is ignored (without loss of information).

2.3.3 Encoding Estelle Specification - The General Concept

The conversion of Estelle Constants, Variables, Types, States, and statesets parts is simple one-to-one conversion into a set of facts(see Figure 2.3.1). The conversion of Channel and Transition parts is discussed in the following paragraphs using the Micro Prolog [Clocksin] syntax for rules and facts.

CHANNEL

The Estelle channel definition identifies the module interaction points and for each interaction point the possible roles (USER / PROVIDER) played by the interaction point. In the KB, these interaction points are represented as channels(chname). The roles are represented in KB as roles(role).

For each interaction point (chname) and each role, Estelle defines the list of applicable Service Primitives. This relationship is developed in KB as :

channel-services(chname, role, spname).

ESTELLE SYNTAX	KNOWLEDGEBASE
constant {cname = cvalue};	constants(cname,cvale).
type {tname = Type};	types(tname,type).
var {vname : tname};	variables(vname,tname).
state {sname};	states(sname).
stateset {ssname = {sname}};	stateset(ssname,sname).
channel {chname (role); BY role: {spname{(vname:tname) ;}}};	channels(chname). roles(role). chanel-services(chname, role,spname). sp-parameters(spname,vname). variables(vname,tname).
trans trname FROM f-state TO t-state WHEN chname.spname PROVIDED condition PRIORITY p-numb BEGIN procedure-code; END	transitions(trname, f-state , t-state, chname, spname, pc-name, p-numb, pro-name). provided(pc-name,condition). procedures(pro-name, procedure-code).

Fig: 2.3.1 - Estelle To KnowledgeBase Conversion

Furthermore the service primitive parameters (and their Types) are also defined in Estelle in the channel definition part. In KB this is converted to separate facts as follows:

```
sp-parameters(spname, vname).
variables(vname, tname).
```

The Service Primitive parameters are maintained in a "variable" database which also contains the variables defined in Estelle "Var" part.

TRANSITIONS

The Estelle syntax for transition defines all possible transitions of the module. This part also includes the specification of transition enabling conditions and the procedure code to execute the transitions.

As mentioned earlier, an extension to Estelle syntax for the transition part is assumed, namely the extended Estelle syntax for transitions requires a transition name which immediately follows the keyword "trans".

The Estelle transition part contain at most one of each of the following:

- from state;
- to state;
- when a Service Primitive is received,
  at an interaction point;
- PROVIDED condition;
- Priority;
- transition execution procedure.

In representation of the transition definition in KB this is broken down into three sets of facts:

```
transitions(trname,f-state,t-state,chname,spname,pcname,
            pro-name).
provided(pcname,condition).
```

procedures(pro-name,procedure).

This breakdown provides a good integration of the generated prototype with the FPCP build-in functions and the KB.

Note that the transitions are not defined as rules. The reason for this is that we had to develop our own inference system to interrogate the KB. This was due to the limited Hardware and Software capabilities of the micro computer on which the trial FPCP system was developed.

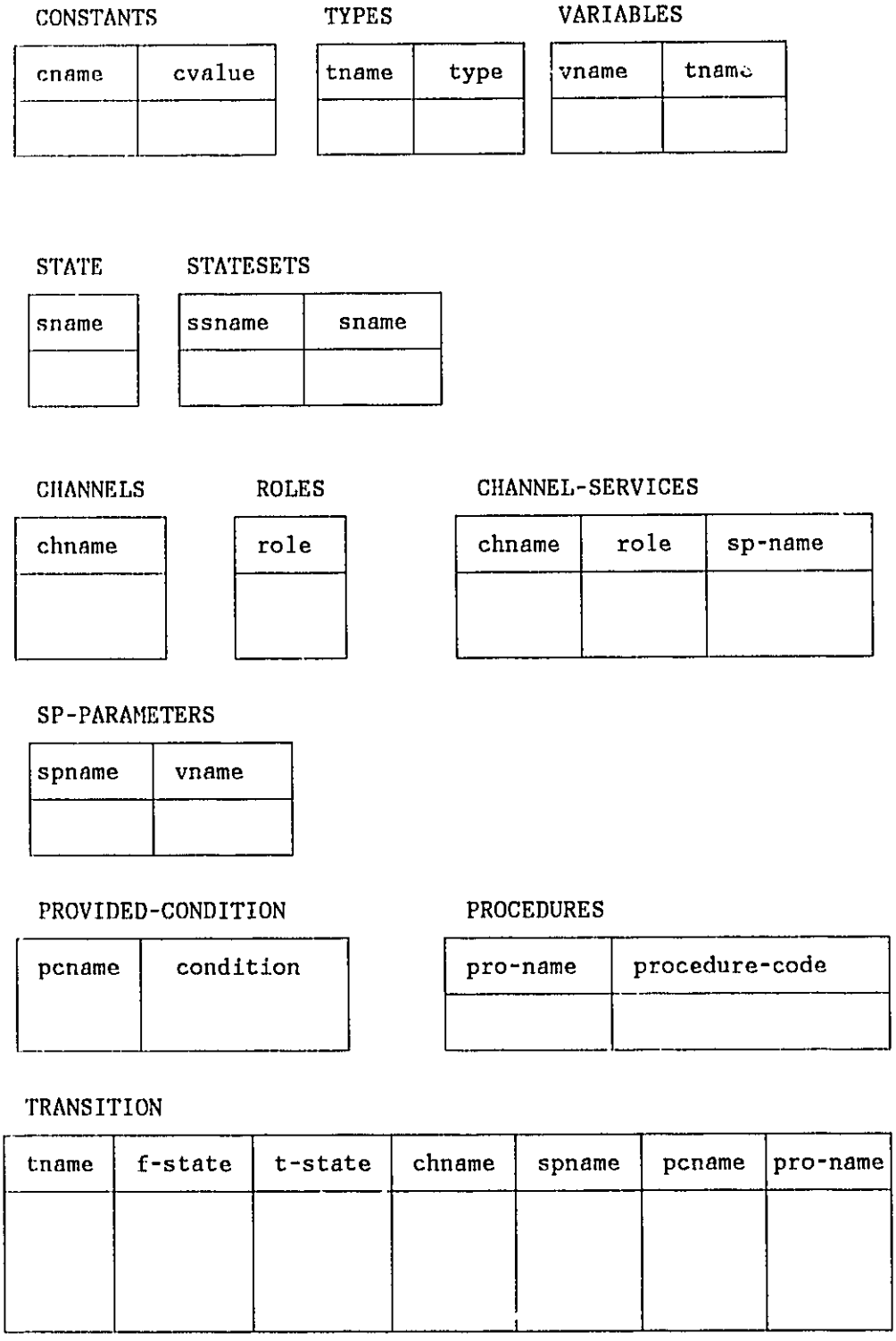
TRANSITION PROCEDURES

The transition procedures in Estelle describe the transformation of module variables, and the channel input and outputs. In the FPCP encoding step these procedures are converted as-is into Pascal procedures. The only exception to this is the translation of the references to "OUT" procedures into FPCP built-in "OUT" procedures with appropriate parameters.

In future enhancements to the FPCP system the process of encoding the Estelle specification into KB, can be modified to parse the transition procedure statements, such as assignment or arithmetic operations, and to maintain them on KB. The parsing may avoid the compilation errors when compiling the generated prototype.

A pictorial view of the entire KB is given on the following page.

Fig: 2.3.2 - Pictorial View Of KnowledgeBase



2.3.4 Interrogation Of Protocol Specification In KB

The design of the KB as shown in the previous section leads naturally to the capability of automated interrogation of the protocol specification [Cohen]. This facility is designed , but not implemented in the current version of the FPCP system. However the ability to automate the interrogation of protocol specification is an integral part of our proposed approach and methodology.

Under this proposed approach two types of interrogation facilities are designed:

i) interrogation of KB as a **static entity** (i.e. containing the protocol specification facts derived from the Estelle specification of the protocol; and

ii) interrogation of KB as a **dynamic entity** (i.e. containing not not only the facts derived from the Estelle specification but also the new "facts" (execution traces) as they are added to the KB during the protocol execution.

The later type of interrogation is discussed in section 2.4.2.1. As well, the uses of KB (as derived from Estelle specification) are illustrated.

In type i) interrogation, the KB can be treated as a database of facts as shown in Fig: 2.3.1. From this database a Natural Language Interface or logic programming language such as Prolog can formulate simple queries to interrogate the database [Hammond] [Hammond84] [Clocksin]. For example queries against the KB can be formulated to check for:

- variables and constants declared but not used;
- variables and constants used but not declared;
- service primitives declared in the Channel-Part but not referenced in any Transition_Part;
- ambiguities such as redefined variables;
- transition deadlocks or possible transitions;

- syntax of test suites.
- incomplete specification

For example the following query against the KB can be used to check for "unspecified receptions":

"Is there exists a service primitive (SPNAME) and a state (SNAME) such that there does not exist a transition (trname) from state SNAME to any state from any channel receiving SPNAME under any PROVIDED condition?"

2.4 Development Of Executable Prototype From KB And Potential Uses

2.4.1 Development of Prototype

GENERAL

As the second step in the proposed fast prototyping approach, an executable prototype is automatically generated from the KB. The generated prototype is integrated with FPCP KB and includes the necessary user interface to handle protocol external inputs and outputs. Chapter 3, Chapter 4 and Appendix C contain more details and examples of the development and use of the prototype in the FPCP system. In this section the design of the executable prototype and its integration with the KB is discussed.

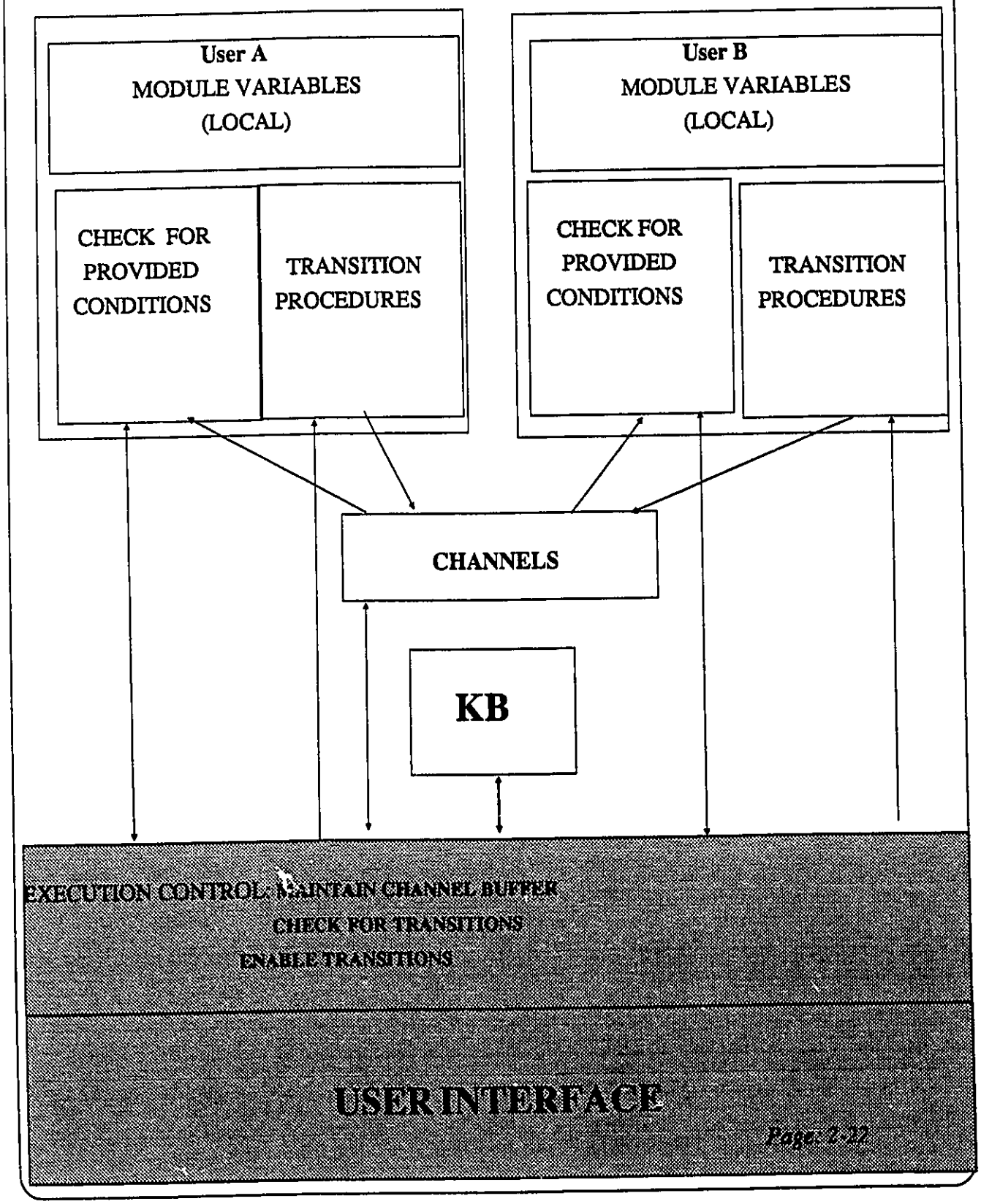
2.4.1.1 THE DESIGN OF PROTOTYPES IN FPCP

The KB contains sufficient reference information to generate an executable prototype. However instead of generating a stand-alone executable prototype, an executable prototype which remains linked to the KB is proposed. The automatically generated portion of the prototype contains the Pascal code for:

Constants and Variables;
Channel Buffer Inputs and Outputs;
Boolean functions of "PROVIDED" Conditions;
Procedures (User Defined) & Estelle transition procedures.

FPCP contains built-in procedures which interact with the KB and the generated Pascal prototype procedures. The FPCP system also includes a User Interface to allow users to interact with the executable prototype in different modes (local, remote, batch, and interactive).

Figure: 2.4.1 Design Of FPCP Prototypes



A pre-compiled generic module is linked with the FPCP generated code. The module includes the mechanism to select and fire a transition. The transition selection procedure is based on evaluating the "WHEN" and "PROVIDED" conditions. When more than one transition satisfies the conditions then the "PRIORITY" of each transition is checked to select one transition. In case of more than one transition with the same priority the system automatically select the first transition.

The transition firing procedure changes the current state of the module and retrieves the data from channels to update module variables, and then transfers the control to user defined transition procedure. The user defined procedures may contain calls to FPCP built-in "OUT" procedures to send data on channel buffers.

The input and output channel buffers are implemented as First-In-First-Out (FIFO) channels. In FPCP these buffers are "records" in virtual disk of the computer. Therefore the channel capacity is assumed to be limited only by the size of the available memory in the system.

The design of FPCP Prototype environment and its integration with the FPCP system KB is shown in Fig: 2.4.1. The FPCP prototype contains two separate copies of the generated code. This is used to simulate remote user interactions. The generated portion of the prototype is used to maintain the protocol module local variables , to check for transition enabling (PROVIDED) conditions, and to process the code to execute a transition.

The generic control portion of the prototype (which is build into the FPCP system) maintains full control of the prototype execution and channel buffers. This portion makes use of the generated code to determine which transition to be enabled and when required initiates the process to enable a transition.

This generic portion can (in an extended version of FPCP) maintain a complete history (traces) of the prototype execution [Hayes84] [Hammond] [Winston]. This history can be integrated with the KB as additional "facts". Such a KB can provide a useful "explanation" component to the KB interrogation facility. The Trace Maintenance and Interrogation facilities are not implemented in the current version of the FPCP system. Possible uses of such facilities are discussed in section 2.4.2.

2.4.1.2 THE EXECUTION OF FPCP PROTOTYPES

As stated earlier, the executable prototype consists of a set of generated Pascal code, the KB, and a set of generic built-in functions. The following paragraphs describe how a protocol is executed in FPCP to illustrate equivalence of the prototype with the original Estelle specification.

Before discussing the process of executing a FPCP prototype and the FPCP Execution Engine, a brief description of each entity in Fig. 2.4.1 is presented.

MODULE LOCAL VARIABLES

This entity contains the declarations of module entity constants, types and variables. They are derived as is from the Estelle specification. A separate copy for each one of the two users is maintained. These variables are used only by the the generated code for the same user.

CHECKS FOR PROVIDED CONDITIONS

These are a set of Boolean functions. Each function contains the PROVIDED condition as coded in Estelle specification. These functions have access to the module variables and the status of each channel. They are called by the EXECUTION CONTROL and they return TRUE if the condition is satisfied.

TRANSITION PROCEDURES

This entity contains the transition processing procedures coded in Estelle. It also includes the module initialization procedure, transition processing procedures, and "other" procedures coded outside the Estelle specification file, and referenced in some initialization or transition procedure block.

The "other" procedures if any are called by the initialization and transition processing procedures. The initialization procedure is called by the EXECUTION CONTROL entity to initialize the module variables. The transition processing procedures, which are derived from the Estelle transition part's BEGIN-END clause, have access to the module variables. These procedures are called by the EXECUTION CONTROL to process a transition. These procedures also have access to a built-in function OUT to output messages on channels.

CHANNEL

This entity contains the protocol module channel buffers and the necessary procedures to Load, Store, and Delete channel messages. The channel status is accessed by the PROVIDED CONDITIONS. The TRANSITION PROCEDURES may store messages on the channels. The

EXECUTION CONTROL stores user messages on these channels. This control entity also outputs the messages to users and where necessary deletes the messages from channel buffers.

KB (KnowledgeBase)

The KB contains the protocol specification as described in the earlier sections. The KB information is used to determine which transition to enable and to validate the format of user inputs to the prototype.

EXECUTION CONTROL (Partially Implemented)

This generic entity is the heart of the FPCP prototypes. It interacts with all other entities in the prototype subsystem. Its major functions are as follows:

1) Call module initialization procedures;
2) Validate user inputs against the KB;
3) Determine which transition to enable from the transition table in KB and the PROVIDED CONDITIONS Boolean functions (where necessary the PRIORITY clause is used to select a transition);
4) Enable a transition and call the TRANSITION PROCEDURES;
5) Delete messages from channels as required;
6) Maintains current State of each module ;
7) Add the audit trail (traces) in KB as additional facts to build a base for "explanation" facility. (This function is currently not implemented).

USER INTERFACE

This entity interacts with a user. It determines the type of interactions with a user, and formats user inputs and prototype outputs from the EXECUTION CONTROL. This entity provides features to execute the prototype in different modes such as :

- Interactive Execution;
- Protocol Testing (Local Mode) using a TTCN file;
- Protocol Testing (Remote Mode) using a TTCN file.

The User Interface displays a graphic view of the protocol execution as shown in figure 2.4.2.

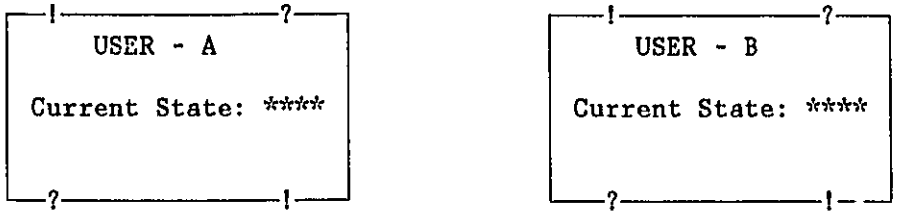


Figure: 2.4.2 Graphic View of protocol Execution

The two peer entities are displayed to simulate remote testing of protocol with two users. For each protocol entity the interaction points are labelled as "!" (provide) and "?" (use). The current state of the entity is displayed in the middle of the box.
During protocol execution , the messages flowing through the interaction points are displayed beside the interaction points.
The User Interface allows the user to place a message (Service Primitive) at any desired interaction point. This can be done interactively or through an input file.

The Process Of Executing A Protocol

The prototype of a protocol is executed by initiating the request from the User Interface. The Execution Control resets the current states of modules entities, and calls the module initialization procedures. Upon receipt of input from a user the Control entity formats the message according to the specification in KB and stores the message on an appropriate channel.
The Execution Control checks the channels periodically. If a message is present for a Higher Level (User) it removes it from the channel and passes it on to the User Interface.
The Execution Control sequentially checks the current state of a module against the transition table in KB to determine the next transition. To do this it checks each transition in the KB. For

each possible transition it calls the PROVIDED CONDITIONS to check that the Provided Conditions are satisfied. From the set of transitions for which the conditions are satisfied, the Execution Control select a transition based on the Priority as stated in the KB. If after these steps more than one transition can be fired then the system automatically selects the first transition.

To complete a transition the Control entity performs the following tasks:

- Loads the channel message into module variables;
- Deletes the message from the channel;
- Calls the transition procedures;
- changes the current state of the module according to the specification in the KB.

During this process the Execution Control entity could write the execution traces as additional "facts" in the KB. This feature however is not included in the current version of the FPCP system. A brief discussion of the implementation of traces in FPCP is included in section 5.2.1.

2.4.2 Uses Of FPCP Generated Protocol Prototypes

GENERAL

The FPCP prototypes as described in section 2.4.1 has many applications in the protocol development life cycle. Some of the major applications are described in this section. They include from refinement of protocol specification during the protocol specification phase to analysis of IUT test results during the implementation phase of a protocol.

2.4.2.1 Debugging and Refinement Of Protocol Specification

There are essentially three ways within FPCP to validate and debug an Estelle specification:

i) In FPCP environment a protocol Estelle specification is automatically converted into a KnowledgeBase (KB) which can be interrogated. The KB interrogation assists developers in detecting ambiguities at an early specification stage.

ii) FPCP system can execute a protocol from Estelle specification by generating an executable prototype from the KB. This facilitates developers in testing the Estelle specification. The FPCP system design includes a feature of maintaining (in the KB) an audit trail (traces) of prototype execution. (This feature is not implemented in the current version of FPCP).

iii) When the traces are available, KB interrogation can provide "explanations" of protocol entities behavior in terms of the original Estelle specification (Figure 2.4.3).

For example, FPCP could maintain the following "facts" when executing an external input interaction:
- current values of module variables and current state;
- external input interactions;

- the facts used to fire a transition;
- modified values of module variables and new State;

Upon a query to explain the prototype behavior , these facts (and the facts derived from the Estelle specification) are used to provide the sequence of Estelle specification followed to produce the output interactions.

This type of interrogation is referred as the "Interrogation of KB as a Dynamic Entity" in section 2.3.2. Although this function is not implemented in the current version of FPCP, a design to accommodate these features is included in section 5.2.1.

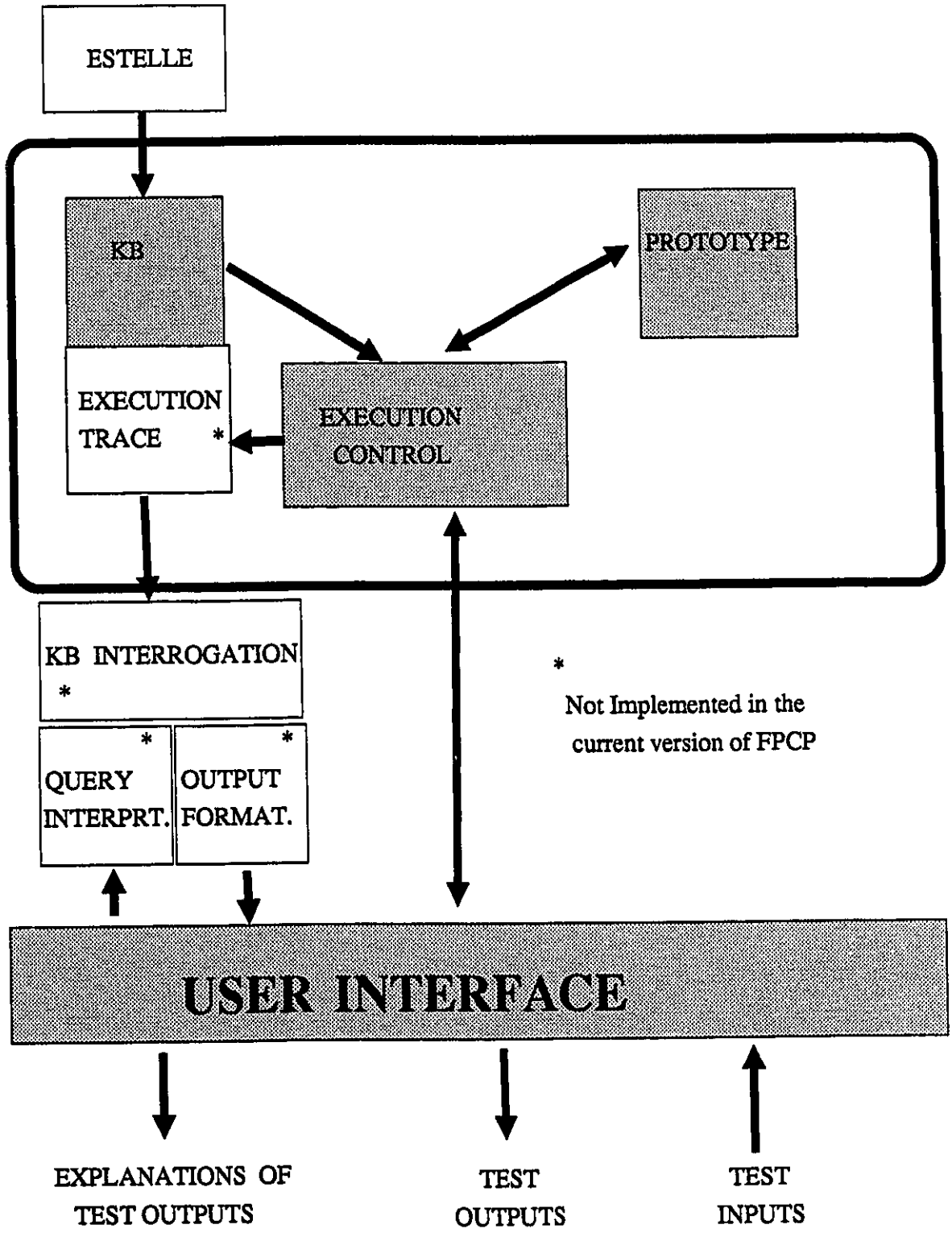


Figure: 2.4.3 Debugging and Refinement Of Protocol Specifications Page: 2-30

2.4.2.2 Development Of Conformance Test Plan

(Not implemented in FPCP)

Once a developer is satisfied with the protocol specification the next task is to develop a Conformance Test Suite (CTS) to test implementations [ISO86b] [ISO87a] [ISO87b] [ISO87c] [Knightson] [Probert83] [Probert86]. The selection of test sequences for adequate testing of protocol is an area of great interest in protocol research studies. Many approaches to the selection of test sequences to develop Conformance Test Suites have been proposed in research papers [Linn86] [Ural84] [Ural85] [Ural87] [Ural87a] [Short].

A common assumption in protocol testing is that the internal structure and code of an IUT (implementation under test) is unavailable. The 'black box' testing and the 'specification based' testing are more common in protocol testing.
The FPCP system which maintains the protocol specification in a KB can be used to generate specification based test cases. These test cases can be processed by the FPCP prototype to prepare well documented and formatted test inputs and outputs on a test file. An overview of FPCP approach for CTS preparation is given in Figure 2.4.4. A brief description of the steps in the test process illustrated in figure 2.4.4 follows:

GENERATE TEST SEQUENCES FROM KB

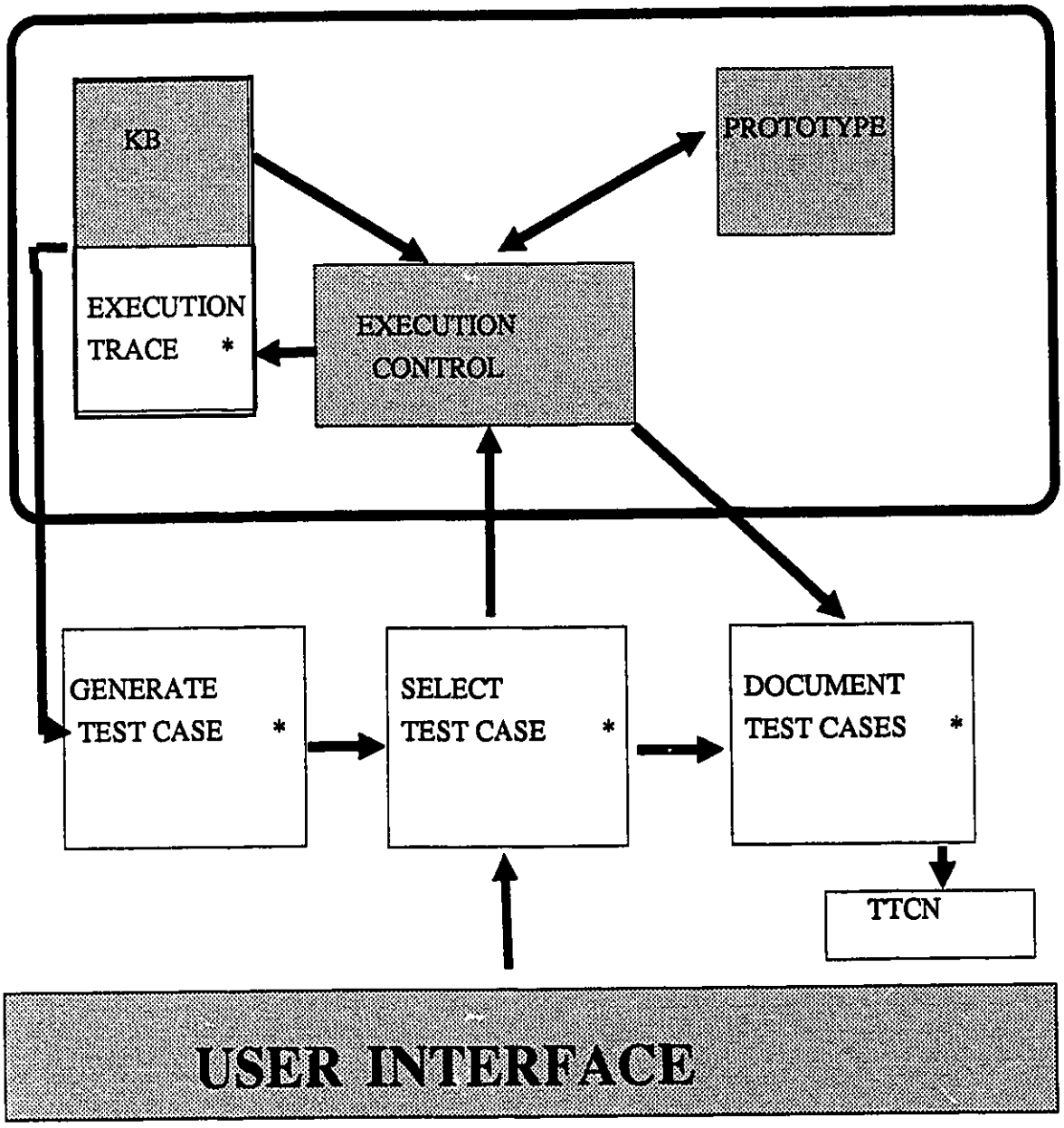
This process uses a specification-based test case generation strategy. The FPCP prototype and the KB can be used to generate these test cases. The test cases can be generated by a user controlled traversal of protocol execution paths (see chapter 4 for examples). The generated test cases are presented to the user for selection or rejection.

SELECT TEST CASES TO APPLY TO THE PROTOTYPE

This is an interactive process which allows user to select or reject a generated test sequence (interactively or by means of an automated test selection strategy). The selected test cases are passed to the FPCP Execution Control module for processing. This process also passes the test sequences as inputs to Document Test Suite process.

DOCUMENT TEST SUITES

This process receives test case inputs from Select Test Cases module and the test outputs from the FPCP Execution Control module. It assembles the test cases and documents the inputs and outputs in a standard format such as TTCN format (see examples on chapter 4).



* Not implemented in the current version of FPCP

Figure: 2.4.4 Development Of Conformance Test Plan With FPCP Page: 2-33

2.4.2.3 Validation and Analysis of IUT Test Results

The analysis of IUT test results involves validation of IUT behavior against the protocol specification. The "explanation" facility in FPCP provides an efficient tool to compare the FPCP prototype behavior with that of an IUT. For any specific test the FPCP prototype can not only provides the expected outputs but can also relate back each step of execution with the authoritative reference specification. An overview of this process is shown in figure 2.4.5.

2.4.2.4 User Training

The FPCP environment with a KB which can be interrogated, executable prototype and the execution traces provide useful and important tools in training protocol users.
The interactive execution with graphic display of module behavior allows users with a hands-on opportunity to learn the protocol. The interrogation of KB allows user to become familiar with the format of protocol Service Primitives.

Furthermore an FPCP user can develop Interaction Scenarios [Jakobson] based on the module entity states and transitions and store these interactions on a file. These interactions can then be selectively executed to demonstrate the protocol behavior to train users.

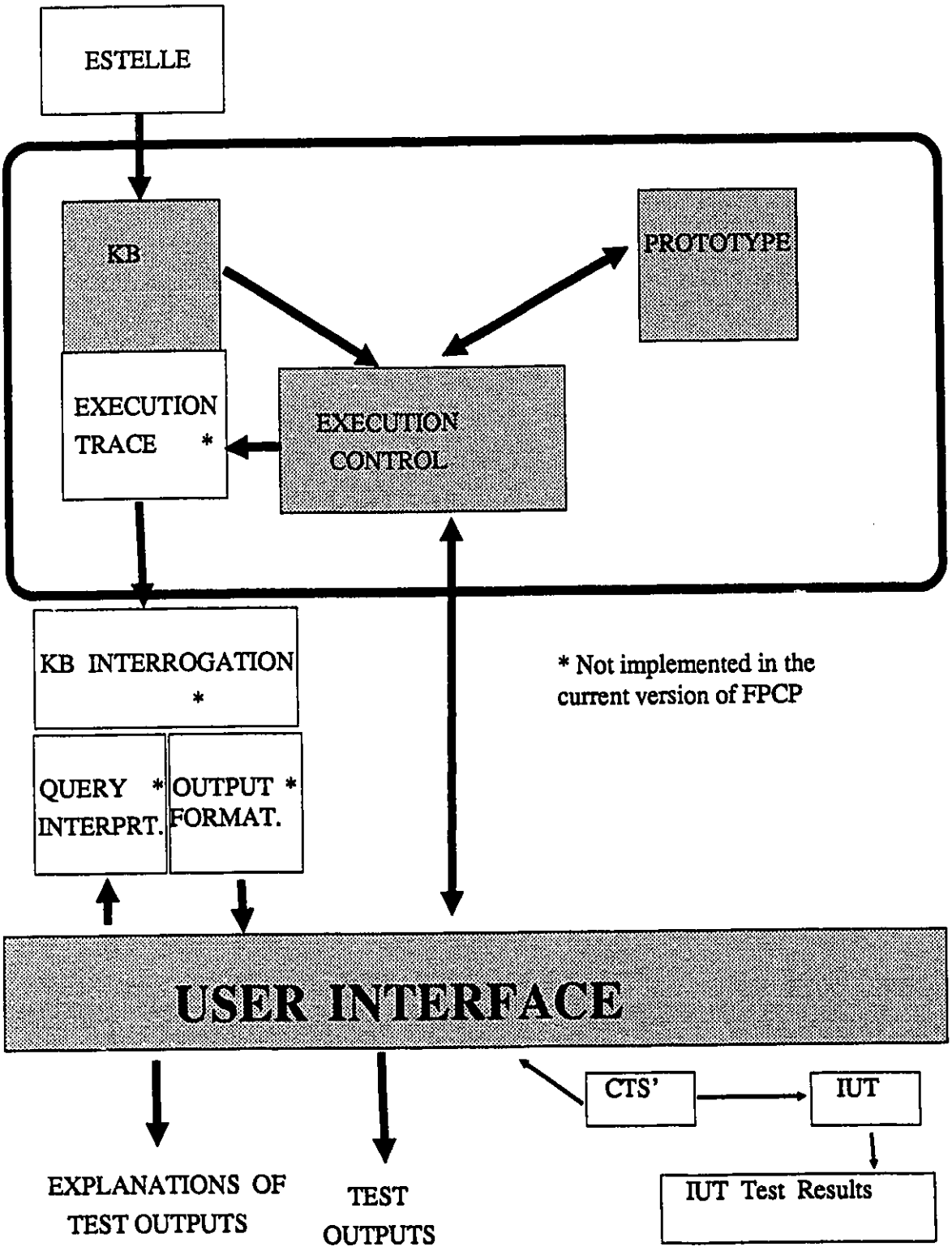


Figure: 2.4.5 Validation and Analysis Of IUT Test Results

3. FAST PROTOTYPING OF COMMUNICATION PROTOCOLS (FPCP) AN INTERACTIVE SYSTEM

In chapter 2 we described our approach to fast prototyping of communication protocols. To illustrate our approach we designed and implemented the FPCP system. The current version of the system does not include all aspects of our proposed methodology. In this chapter we describe the design and features of the existing version of the FPCP system.

The objective of developing FPCP system is to provide an integrated support environment for the implementation of communication protocols. The current version of FPCP system includes the following features as described in chapter 2:

- Automated encoding of formal specifications into KB;
- On-line inquiry of the KB;
- Generation of an executable prototype;
- Execution of generated prototype for debugging specification, conducting tests and user training.

The FPCP system does not yet include the following features:

- Decoding KB into Estelle Specification (reverse engineering of KB);
  Although we insist on the Estelle specification to be the authoritative reference, the decoding of the KB into Estelle specification would be useful in providing the explanation of FPCP prototype behavior in terms of the Estelle specifications.

- Interrogation of KB;
  This feature would indirectly provide the interrogation of formal protocol specification.

- Development of Conformance Test Plan.
  Development of specification-based test sequences in the FPCP environment would greatly increase the usefulness of the system.

In section 3.1 a brief overview of the FPCP system design is presented. In section 3.2 we give a complete list and description of the features available in the current version of the system. Appendix C (FPCP System User Guide) contains a detailed description of FPCP design and system components. Appendix C also includes procedures to compile and start the FPCP system.

3.1 FPCP System Design And Constraints

For reasons of system availability on a dedicated basis, the FPCP system (prototype version) was developed on a DEC RAINBOW microcomputer under the CP/M-86/80 Version 2.00 (1.1) operating system. The entire FPCP system was developed in TURBO PASCAL Version 3.00A. The DEC system had only 64K main memory, two 360K five and a quarter inch floppy disk drives, and 2 blocks of virtual disk memory, each 64K.

The conversion of FPCP into a more current hardware/software environment would increase the usefulness of the FPCP approach. The existing limits such as number of variables and the size of variable names can be increased if the system is implemented on a hardware with more memory. Similarly, FPCP implementation using a higher-level language such as Prolog, with built-in inference engine, would make the implementation of FPCP interrogation facilities easier.

The run-time FPCP system organization is illustrated in Figure 3.1.1. Disk drive A contained a working copy of the operating system, and the TURBO PASCAL Compiler. The second drive (drive B) was used for FPCP system programs and data files. The two blocks of virtual memory were used to maintain the KnowledgeBase in memory. This reduced the number of I/Os and improved the performance of the system significantly.

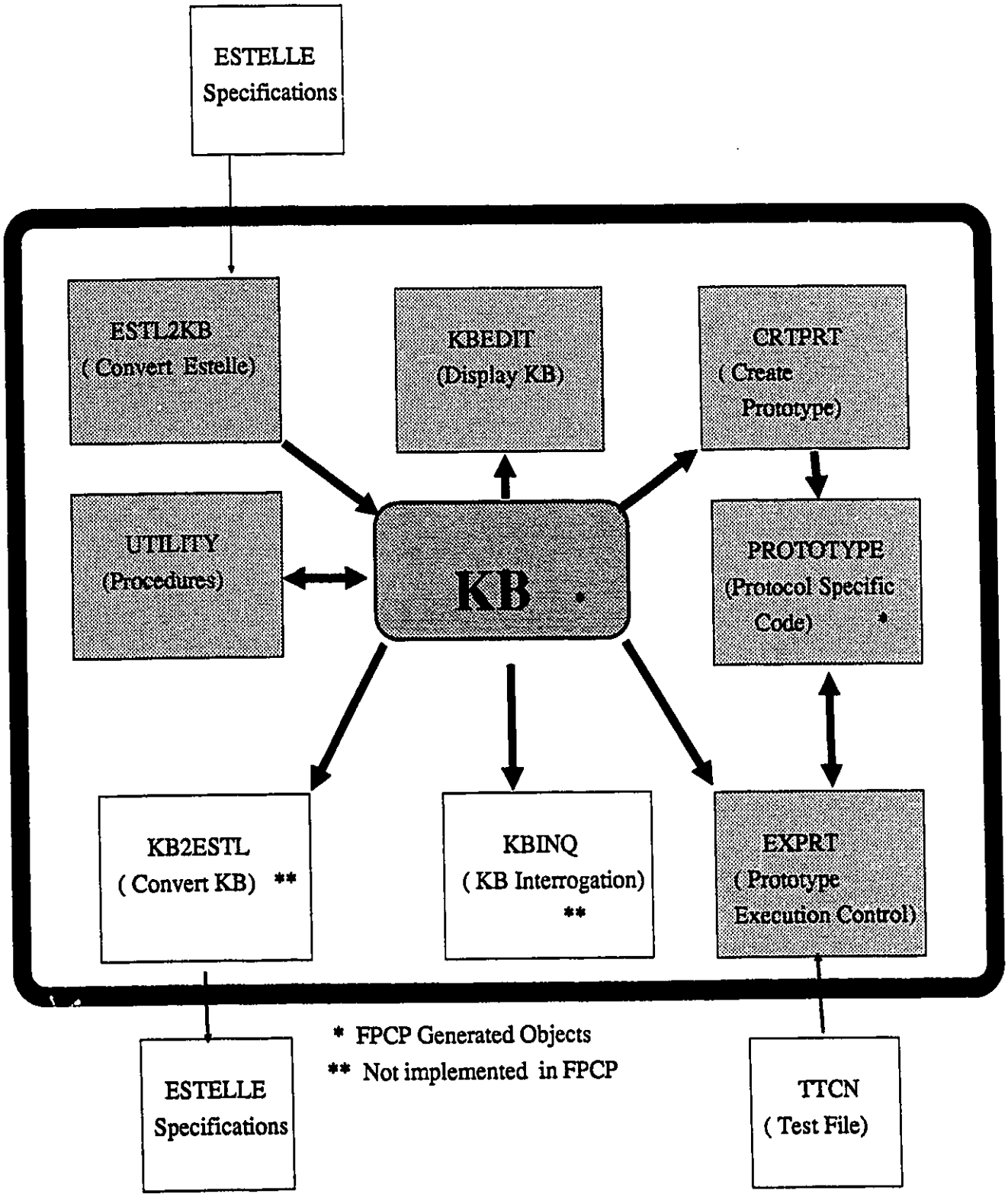


Fig: 3.1 - FPCP DESIGN OVERVIEW

The FPCP system file size exceeded the limits of the Pascal Editor. Therefore the program file had to be divided into a series of smaller files. Each program file contained a subsystem or series of FPCP procedures. The main program file contained the necessary **include** instructions to include all program files when compiling the system.

The FPCP system design is divided into seven processing subsystems as shown in Figure 3.1.1. Five of these subsystems (dark rectangles) are implemented in the current version of the FPCP. A brief description of these subsystems follows:

ESTL2KB - This subsystem performs the **encoding** of KB. It converts the protocol specification from ESTELLE to KB.

KBEDIT - This provides a facility to **browse** the KB. The current version of FPCP does not include the facility to edit (change) the KnowledgeBase interactively. However, the KB can be changed at a high level by changing the source Estelle specification and re-converting the Estelle specification into KB. This type of change mechanism is actually preferred from the point of view of control over versions of the protocol specification.

UTILITY - This subsystem contains utility procedures to perform such tasks as:

- Screen Inputs and Outputs
- Save KnowledgeBase
- Load KnowledgeBase

CRTPRT - This subsystem **generates** an executable prototype of a protocol from KB, denoted PROTOTYPE in Figure 3.1.1.

EXPRT - This subsystem provides the facility to **execute** the prototype interactively. EXPRT consists of four components:

EXPRT0 - Prototype Execution Control;

EXPRT1 - Implementation dependent variable declarations;

EXPRT2 - Generic transition firing module;

EXPRT3 - Protocol implementation dependent PROVIDED conditions and transition procedures.

Two of these four components (EXPRT1 and EXPRT3) are generated by the CRTPRT subsystem and are tailored to the specific protocol under implementation.

KBINQ - This subsystem is not implemented in the current version of FPCP. This subsystem when implemented will provide users ability to **interrogate** the KB.

KB2ESTL - This subsystem is also not implemented in the current version of FPCP. It is intended to **generate Estelle** specifications from KB. The feasibility of KB2ESTL is directly related to the degrees of editing allowed by the KBEDIT subsystem. If unrestricted editing is allowed, it may not be possible to design and develop fully automated KB2ESTL.

3.2 FPCP Files and Objects

There are two files which are input to FPCP. One contains the Estelle specification of the protocol (**ESTL.PAS**) and the other contains test suites in TTCN format (**TTCN.PAS**). The file ESTL is used by the ESTL2KB subsystem when converting the Estelle specification into KB. The second file (TTCN) is used by the EXPRT subsystem in interactive execution of the prototype.

As documented in chapter 2, the object KB (KnowledgeBase) consists of facts automatically encoded from the Estelle specification of a protocol. For example an Estelle transition clause :

```
" trans t1 from idle to wfcc when n.tcreq provided
        tcreq.qts_req = ok_qts
        begin .... end; "
```

is represented in the KB as the following facts/rules:

```
transition(t1,idle,wfcc,n,tcreq,pro-condition1,
            priority,procedure1);
provided(pro-condition1, 'tcreq.qts_req = ok_qts');
procedures(procedure1, 'begin ....end;');
```

The object PROTOTYPE consists of implementation dependent (automatically generated from the protocol Estelle specifications) module variable declarations, PROVIDED conditions, and transition PROCEDURES. Please refer to Appendix D (listing for files EXPRT1 and EXPRT3) for a complete list of FPCP generated prototype Pascal code for the transport(0) protocol example described in chapter 4.

3.3 System Features And Limitations

In this section we give a brief description of the options which can be selected by a user in an FPCP session. A detailed User's Guide is given in Appendix C.

In Figure 3.3.1 we give a tree diagram representing the menu-driven user interface of FPCP.

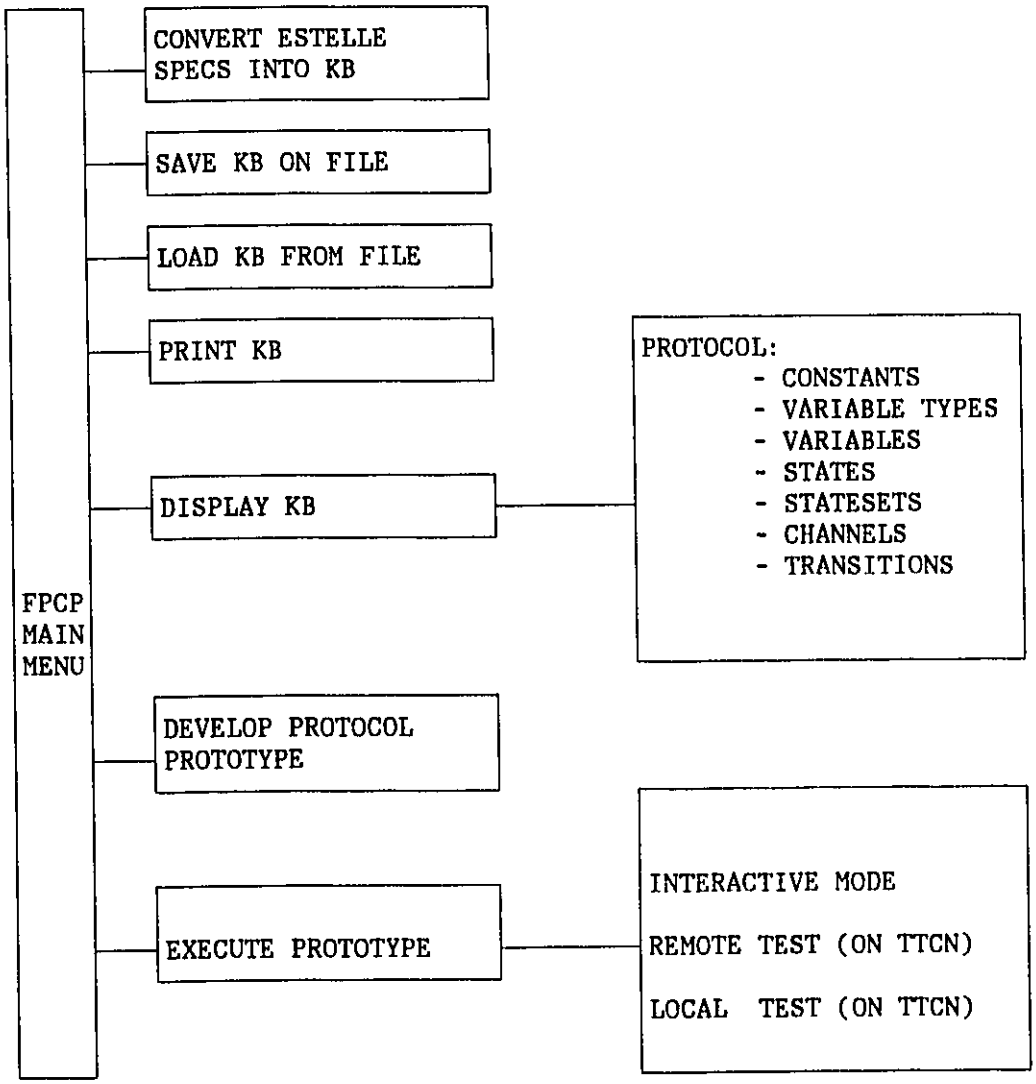
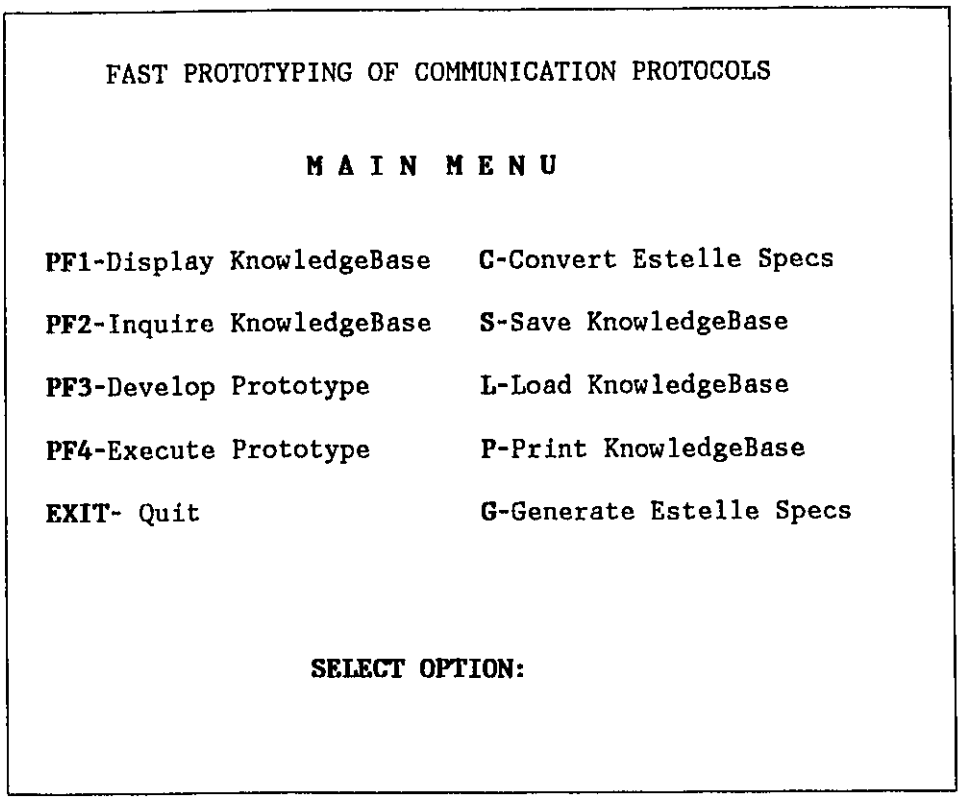


Figure 3.3.1 FPCP Menu-driven User Interface

THE FPCP SYSTEM MAIN MENU

The system main menu displays the functions available to the users. The FPCP system Main Menu screen layout is shown below.



Once a file of Normal Form Estelle specifications is created the first step is to convert the specification into KnowledgeBase by using the option C on the main menu. The automatically converted KB is then saved (option **S**) , displayed (option **PF1**) for debugging or verification purposes, or used to automatically develop an executable prototype (option **PF3**).

During the step to develop the executable prototype (option PF3) the system generates Pascal procedures containing the protocol specific code. Before the generated prototype can be executed (using option **PF4**) the user is required to re-compile the entire FPCP system linking the generated code and generic (implementation independent) Pascal routines. These functions are described in more detail in Appendix C, and are only briefly reviewed below.

Option C - Convert Estelle Specs

This option allows users to convert the given Estelle specification into a KnowledgeBase (KB). A file named 'ESTL.PAS' must be supplied containing the protocol specification in Estelle.
If an Estelle Syntax error is detected the error is displayed and the conversion is terminated. Upon completion of this task the system returns to the Main Menu.
This step is required at least once, and every time the Estelle source specification is changed.

Option S - Save KnowledgeBase

This option is used to save the contents of KB on diskette. This step is necessary when a new version of the Estelle specification is converted into KB during the current session.

Option L - Load KnowledgeBase

This option is used to load an existing KB (File named 'KB.PAS') from diskette.

Option P - Print KnowledgeBase

This option allows users to create a print file of the KB in memory. The function is available as FPCP system debugging tool.

Option G - Generate Estelle Specification

This option is not available in the current version of the FPCP system. It is intended to convert the KB back into the Estelle specification file.

Option PF1 - Display KnowledgeBase

This option lets the user browse through the KB. The system displays a series of screens containing the information in the KB. The top two lines of each screen are:

```
OPTION: 1=Const 2=Type 3=Channel 4=Module 5=State 6=Stateset
        7=Variables 8=Transitions 9=Procedures
```

Each option can be selected for display by pressing the 'ADDITIONAL OPTIONS' key followed by the option number (1 to 9).
The screen containing the information on the module constants declarations is presented first. Then a user can select any one of the nine screens as mentioned above.

From each screen a user can select next or previous screen by pressing 'NEXT SCREEN' or 'PREV SCREEN' keys respectively. Where applicable, the PgUp and the PgDn keys can be used to browse through the previous and next page of a screen. The Up and Down arrow keys are used to move the cursor up and down to select an item on the screen.

The format of each screen and the use of some function keys in each screen is different from screen to screen. A detailed description of each screen and the applicable function keys is given in Appendix C.

Option PF2 - Inquire Of KnowledgeBase

This option is not available in the current version of the FPCP system. This option combined with a natural language interface is expected to allow users the capability to interrogate the KB. Furthermore when this function is integrated with the executable prototype it could provide better explanations of protocol behavior (from the prototype execution trails).

Option PF3 - Develop Prototype

This option allows user to develop an executable prototype of a protocol from KB. This function when executed generates two Pascal program files containing protocol module constants, variable declarations, channel input / output, code tailored for the protocol, and the necessary CALLs to user defined procedures.
The user interface to execute the prototype in the interactive mode is part of the common built-in generic functions in the FPCP system. A detailed example of the use of FPCP system to build a Transport(0) protocol prototype is given in chapter 4.

Option PF4 - Execute Prototype

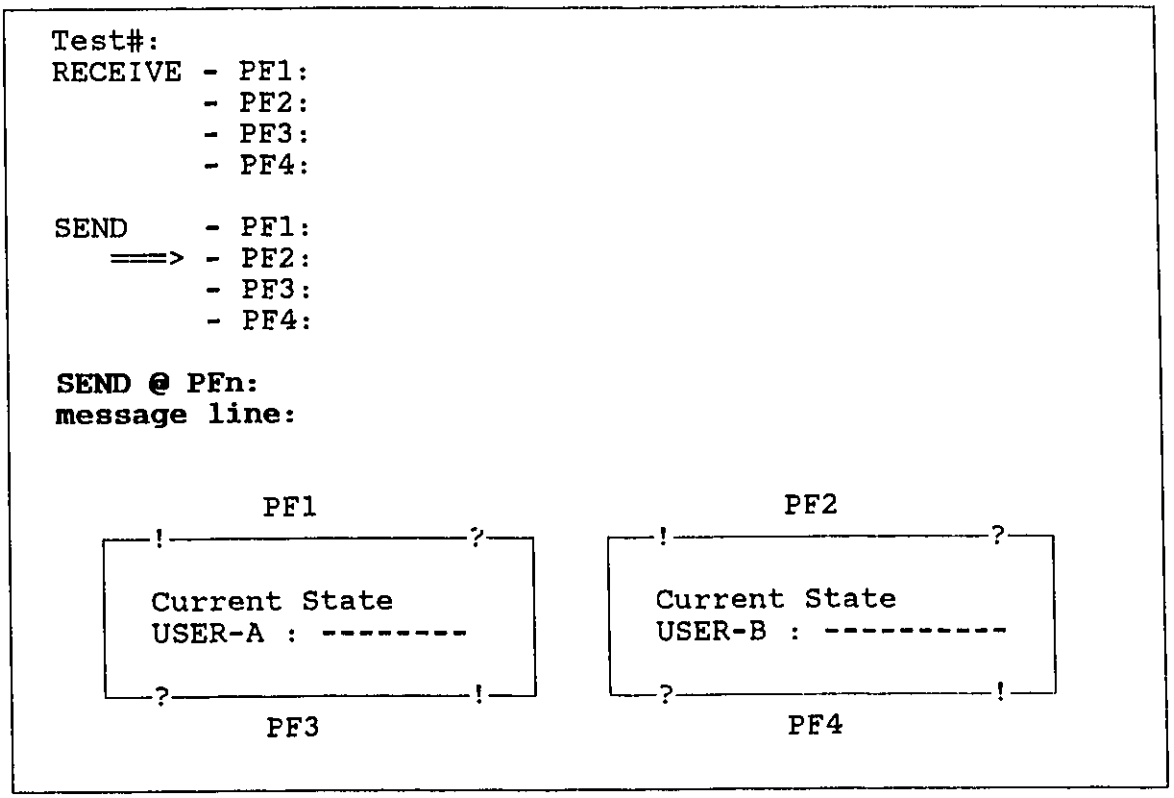
This option is available once a prototype of the protocol has been developed (using the option PF3 as stated above), and the FPCP system has been re-compiled. This option allows the user to execute the protocol interactively. The user is presented with a choice of three different execution modes(see the screen sample below).

PROTOTYPE EXECUTION - OPTIONS MENU

```
        EXECUTE PROTOTYPE

I - Interactive Execution
R - Remote Test Method - TTCN File
L - Local Test Method  - TTCN File
    Enter Option:
```

For each one of the three prototype execution modes the system displays a screen as shown on the following page. The screen lets users interact with the protocol and displays the protocol behavior.

PROTOTYPE EXECUTION SCREEN



This screen displays the status of two communicating protocol entities. Each box represents a protocol entity. In the center of each box, the current STATE of protocol entity is displayed.

The two entities interaction points and the data channel buffers are annotated by PFi, '!', and '?' indicators.

PF1 is the protocol entity A interaction point with the higher layer. PF3 is the entity A interaction point with the lower layer. Similarly PF2 and PF4 are the protocol entity B interaction points. The '!' and the '?' indicators indicate the protocol entity interaction point channel input(?) and output(!) buffers respectively.

During the execution of the prototype, the current STATE of each entity is displayed in the appropriate box. The names of Service Primitives sent or received are are displayed along with '!' and '?'. The top portion of the screen shows the actual messages which are sent and received in each buffer.
The 'SEND @ PFn' line is used in the 'INTERACTIVE' mode only to select an interaction point for sending messages. To select an interaction point one of the PF keys 1 to 4 is pressed. To send messages after selecting the interaction point press SEND. A new line will appear for the user to enter the desired message.

A brief description of each one of the three execution modes is given in the following paragraphs. Details are given in Appendix C.

Option I - Interactive Execution

The user can select any interaction point and then send a message (Service Primitive) by pressing the 'SEND' key followed by the message on the 'message line' area.

The user input interactions are verified against the service primitives specification in the KB. If the input sequence is incorrect, an error message is displayed.

This feature is used to debug and refine the protocol specification. The input sequences are entered on this screen. The system displays the module entities output sequences and the contents of each channel. The outputs are validated against the specification by the user. If any change is required the user is required to make the changes on the protocol Estelle specification file.

This screen can also be used to validate IUT test results. The tests conducted on IUT can be repeated here and the outputs of the prototype compared to that of the IUT results.
Please refer to transport(0) example in section 4.

Option R - Remote Test Method - TTCN Test File

The Remote Test method allows the user to conduct testing of the prototype using protocol test suites coded in TTCN format (see Appendix B for a sample of transport(0) test sequence in TTCN format). This option expects a disk file in drive B named 'TTCN.PAS' containing the test suite.

Under this mode the prototype gets its inputs from the TTCN file.
At each step of testing, a complete picture of the protocol entities states and contents of the buffers are flashed on the screen. The prototype output is compared with the expected results specified in TTCN file. The results of this comparison is displayed as each test sequence is executed. At the end of each test the system pauses for the user to press any key to continue testing until all TTCN file test inputs are processed.
Please refer to the transport(0) example in chapter 4.

Option L - Local Test Method - TTCN Test File
This mode is similar to the "Remote Test Method" discussed above. Under the "Local Test Method" the prototype is executed as a single (local) entity as opposed to two remote peer entities as in Remote Test Method.

3.4 Summary Of FPCP Design Rationale

The FPCP design objectives were two-fold:

i) Use of Rule-Based KB:
The FPCP design provides for automated encoding of protocol specification into a KnowledgeBase (KB). The KB, as an internal representation of the protocol specification, remain transparent to users. This allows users to take advantage of rule-based techniques and yet maintain the Estelle specification as the single authoritative reference specification during all phases of protocol implementation.

ii) Automated Generation Of Executable Protocol Prototype:
The FPCP prototype implementation environment includes generic implementation independent Pascal procedures. These procedures include user interface with the prototype, transition firing system and channel input/output procedures.

These procedures are linked and compiled with the generated code to yield a prototype of the protocol. This approach provides a fast and efficient tool for protocol prototyping, and provides greater flexibility in developing user friendly interfaces with the prototypes.

In this chapter we have overviewed the structure of the FPCP system and the generated prototype. We have also reviewed the operations of the FPCP system and the generated prototype.
In the next chapter we illustrate the usefulness and the feasibility of these FPCP features using the transport(0) protocol as an example. The automated encoding of the specification into KB, generation of executable prototypes, and the user interface for executing the generated prototypes are there described in detail.

4. AN EXAMPLE: TRANSPORT(0) PROTOCOL PROTOTYPING WITH FPCP

In this chapter we present an example of protocol implementation with the FPCP system with a view to demonstrate the viability of our proposed approach and the effectiveness of the FPCP system as a tool. For our trial we employed the transport protocol class(0). The transport(0) is the most complete protocol and is defined formally in Estelle [Ural84]. We will assume some familiarity with that protocol in our discussions. An Estelle specification (a sample of which is shown in appendix A), is used to illustrate the system details. As stated earlier, not all aspects of the proposed approach are implemented in the current version of the system. This section includes discussion of only those features in the current version of FPCP. Thus we will discuss:

- Automated encoding of Transport(0) Estelle Specification into KB;
- Generation Of FPCP Transport(0) Prototype;
- Execution Of FPCP Transport(0) Prototype for debugging, analysis of IUT test results, and User Training.

In section 4.1 we discuss and illustrate the procedures to encode specification into a KB using the FPCP system. The steps required to generate an executable prototype are discussed in in section 4.2. We illustrate the FPCP User Interface and the execution of the generated prototype in section 4.3. In section 4.4 we we examine uses of FPCP generated prototypes. Finally in section 4.5 we include our assessment of the FPCP generated Transport(0) prototype.

4.1 Automated Encoding Of Transport(0) into KB

A formal specification of transport(0) protocol has been developed in normalized Estelle format [Ural87]. A partial listing of this specification is included in appendix A. The transport(0) protocol contains access points for two channels :

- u-access-pt (the channel between Transport and Session entities);
- n-access-pt (the channel between Transport and Network entities).

For each channel a set of service primitives is defined in the channel specification part of the Estelle specification. The service primitives for u-access-pt are:

- By User: tcreq, tcres, tdreq, and tdatr;
- By Provider: tcind, tdind, tdati, and tccon.

Similarly the service primitives for n-access-pt are:

- By User: cr, dr, dt, cc, and ndreq;
- By Provider: nrind and ndind.

To employ FPCP to generate a KB for transport(0), we proceed as follows:

i) **PREPARE A COPY OF FPCP DISKETTE**

First, a working copy of FPCP system is created. This is done by coping all files from the FPCP system diskette to a blank formatted diskette. All data files such as the protocol specification file and protocol test files will be maintained on this copy of the system. Detailed procedures for preparation of a copy of the FPCP diskette are included in Appendix C.

ii) **CREATE TRANSPORT(0) ESTELLE SPECIFICATION FILE**

An Estelle specification consists of identifiers and transitions. In developing the protocol specification file, make sure that identifiers (variable names, interaction point names etc) are not more than 12 characters long. In the transition part we included a transition identifier after the keyword "trans". This extension to the Estelle syntax is required by the FPCP system.

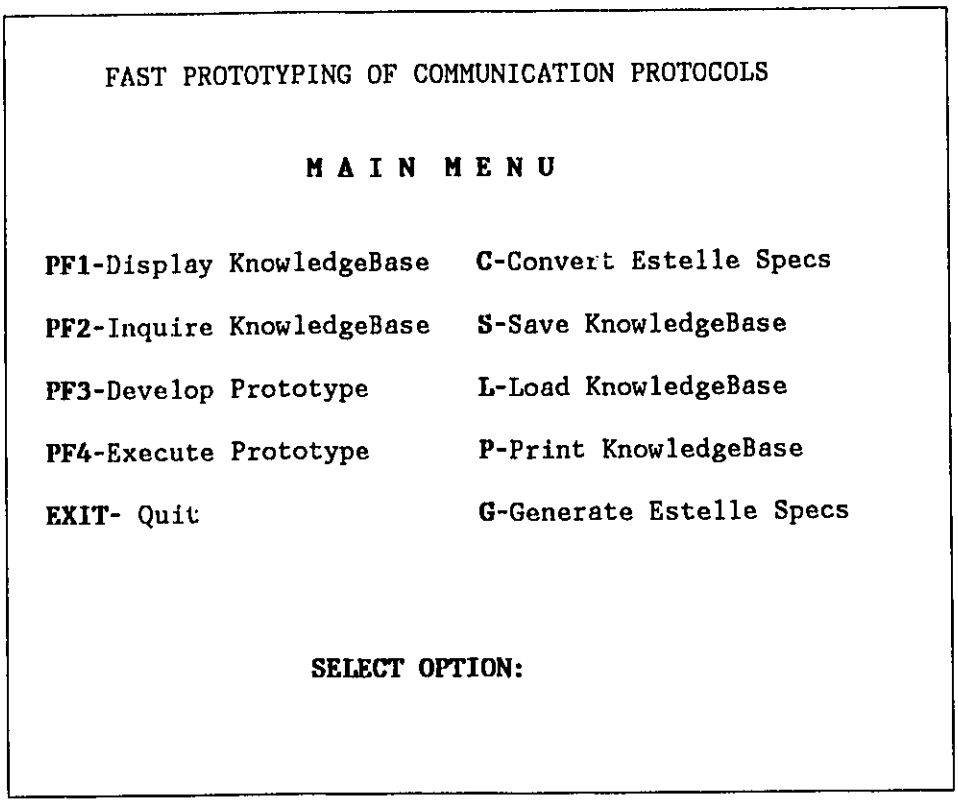
In the transition processing code we modified the syntax of the "out" clause to only include the name of the service primitive output in transition processing.

Once these changes to the transport(0) Estelle specification have been completed, we create a sequential text file (ESTL.PAS) containing the transport(0) modified Estelle specification. Any standard text editor could be used, for example, we used the Turbo Pascal editor.

iii) **START THE FPCP SYSTEM**

Start the FPCP system with the copy of the FPCP system diskette. Detailed procedures for starting the system are included in the User Guide in Appendix C.

Upon starting the system the system main menu as shown on the following page appears. When we start the system for the first time from a copy of the system diskette, only option C - Convert Estelle Specification is applicable.



The other options are meaningful only after the specification is converted into KB form. The option PF4 - Execute Prototype can only be selected after a prototype is developed (using option PF3) and the FPCP system is re-compiled with the generated code. (In future enhancements to the system, the menu could bemade more dynamic, displaying only those options for which the prerequisite steps have been completed by the user).

iv) **CONVERT ESTELLE SPECIFICATIONS INTO KB**

The Estelle Specification file we created is named "estl.pas" and is saved on the protocol FPCP diskette. We selected Option C from the main menu to start the conversion of transport(0) Estelle specifications into KB. The system displays Estelle specification lines as they are converted to KB form. On any

conversion error the error message is displayed and conversion process terminates. When a conversion error occurred we corrected the error on the specification file and restarted the system to try the conversion again. We repeated these steps until all specification syntax errors were eliminated.

The generated KB consists of facts automatically encoded from the Estelle specification of transport(0) protocol. For example the following transition specification in Estelle:

```
 "trans t1 from idle to wfcc when n.tcreq
  provided tcreq.qts_req = ok_qts
  begin ..... end;"
```

is represented in KB as:

```
  transition(t1,idle,wfcc,n,tcreq,pro-condition1,
             priority,procedure1);
  provided(pro-condition1, "tcreq.qts_req = ok_qts");
  procedures (procedure1, "begin  ...   ");
```

As shown in chapter 2, the following channel definition in Estelle:

```
    "channel u_access_pt by user :
                tcreq (to-t-adr :adr;
                from_t_adr      : adr;
                qts_req         :int);
```

is represented in KB as :

```
    channels(u_access_pt)
    roles (user)
    channel-services (u_access_pt,user,tcreq)
    sp-parameters(tcreq,to_t_adr)
    variables(to_t_adr,adr)
    sp-parameters(tcreq,from_t_adr)
    variables(from_t_adr,adr)
    sp-parameters(tcreq,qts_req)
    variables(qts_req,int);
```

Once an error free conversion is completed we were able to use other options from the Main Menu. Next we used the Option "Save KnowledgeBase" to save the KB on disk. This avoids the need to convert the Estelle file every time the FPCP system is started.

v) **SAVE KNOWLEDGEBASE(KB) ON DISK**

We selected the option "S - Save KnowledgeBase" to save the KB on disk. The KB is saved on disk in drive B as a file named "kb.pas". A KB once saved can be reloaded in subsequent FPCP sessions by selecting the main menu option "Load KnowledgeBase".

The current version of the system works with only one version of the KB file (kb.pas). Like the Estelle specification we may maintain different versions of KB by copying the KB file (kb.pas) with different file names. When required to process a newly desired version of the KB we renamed the desired version as "kb.pas".

Now that the KB has been created, there are several operations which can be carried out on the KB prior to developing an executable prototype from KB, for example:

LOAD KNOWLEDGEBASE (KB) FROM DISK

Once a KB is saved on diskette it can be loaded in subsequent FPCP sessions by selecting the option "L - Load KnowledgeBase" from the main menu.

PRINT CONTENTS OF KNOWLEDGEBASE (KB)

We used the main menu option "P - Print KnowledgeBase" to create a print file. The print file is named "kbprt.pas". The file is printed by using the CP/M PIP command. The print version of the KB is used for debugging the FPCP system and the automated encoding.

BROWSE THROUGH THE KB

We use the main menu option "PF1 - Display KnowledgeBase" to browse through the contents of the KB. As described in Appendix C, the FPCP system provided easy to follow displays for browsing through all parts of transport(0) Estelle specification. A complete description of all inquiry screens is given in Appendix C-4

4.2 GENERATING TRANSPORT(0) PROTOTYPE

i) Selecting the option "PF3 - Create Prototype" provided on the main menu is the first step in creating an executable prototype from the KB. We select this option after completing the error free conversion of the specification into KB.

This step creates two files ("exprt1" and "exprt3") on the disk in drive B. These two files contain transport(0) protocol implementation specific processing code.

The file EXPRT1 contains the Pascal declarations of protocol variables. The generated PROVIDED Boolean procedures and the transition processing procedures are generated on file EXPRT3.

Once the processing of this option is completed we terminated the FPCP session (by pressing the EXIT key on the main menu) in order to perform the second step of creating the prototype.

ii) The second step requires the recompilation of the FPCP system with the two generated files included, as well as the pre-coded generic procedures. These pre-coded procedures include the prototype user interface, channel input/output procedures, and transition firing procedures.

The steps required for this task are documented in Appendix C-2. In summary it requires: - starting the Turbo Pascal from drive A;

- loading the FPCP main program (fpcp.pas);
- compiling with compiler option for a .COM file.

The compiler instructions to include the two generated files are in the FPCP main program file (fpcp.pas).

The FPCP system with the generated code included is expected to compile error free. The Estelle "transition" processing code part of the specification is copied as is onto the generated portion. In our tests of FPCP, this resulted in some compilation errors. To correct this situation we had to go back to the Estelle specification file (ESTL.PAS) to make corrections and repeat the above steps to re-generate the prototype. We performed several iterations before a prototype which compiled error free was created.

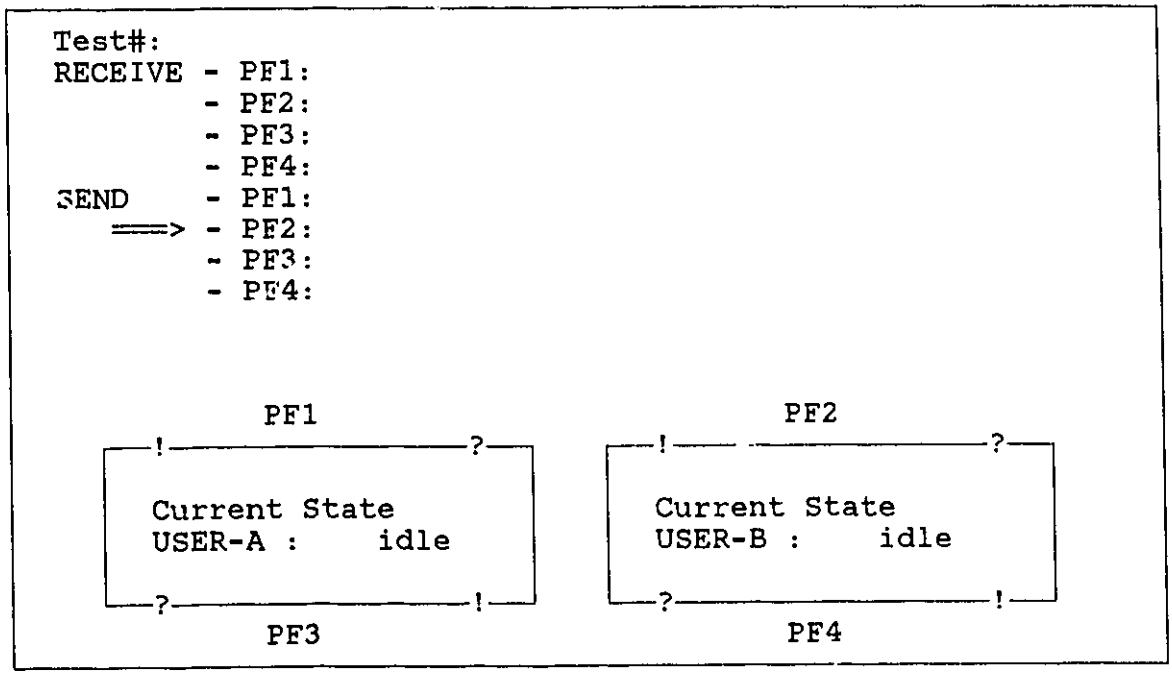
4.3 Executing FPCP transport(0) Prototype

The main menu option "PF4 - Execute Prototype" is selected to execute the prototype (of the transport(0) protocol). Upon selection of this option the system displayed the Prototype Execution Options menu. This options menu allows for selection of one of the three different execution modes.

PROTOTYPE EXECUTION - OPTIONS MENU

```
        EXECUTE PROTOTYPE

  I - Interactive Execution
  R - Remote Test Method - TTCN File
  L - Local Test Method  - TTCN File

      Enter Option:
```

For each one of the three prototype execution modes the system displays a screen as shown on the following page. The screen lets users interact with the protocol and displays the protocol expected behavior.



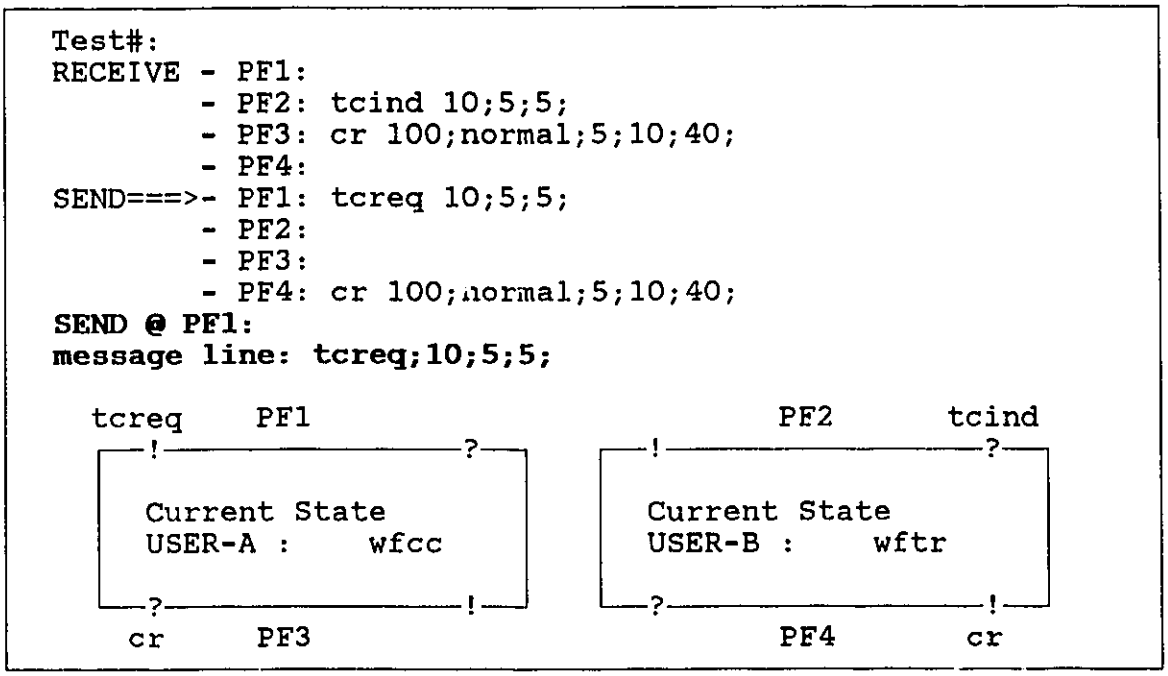
Prototype Display at Initial Stage

For our first example we selected option I - **Interactive Execution** from the prototype execution options menu. Upon selection of this option the display is as shown above. Our first example contains three input sequences and the proper prototype output after each interaction.

The first input sequence is the connection request from user A(site address 10) to user B (site address 5) with quality of service 5. To send this request from user A to user B the following steps are performed:

1) Enter "PF1" to select the user A interaction point;
   The cursor (===>) moves to SEND PF1 line.
2) Enter "SELECT" to display the message line;
   A message line appears for user to enter the input sequence.
3) Enter the input sequence. (tcreq;10;5;5;).
   The input sequence consists of the service primitive name followed by the parameters. Each item in the input sequence is separated by a semicolon (;).

Upon receiving the input sequence the prototype will execute the sequence and display the outputs as shown below.



Prototype Display after Sending tcreq at User A Upper Level

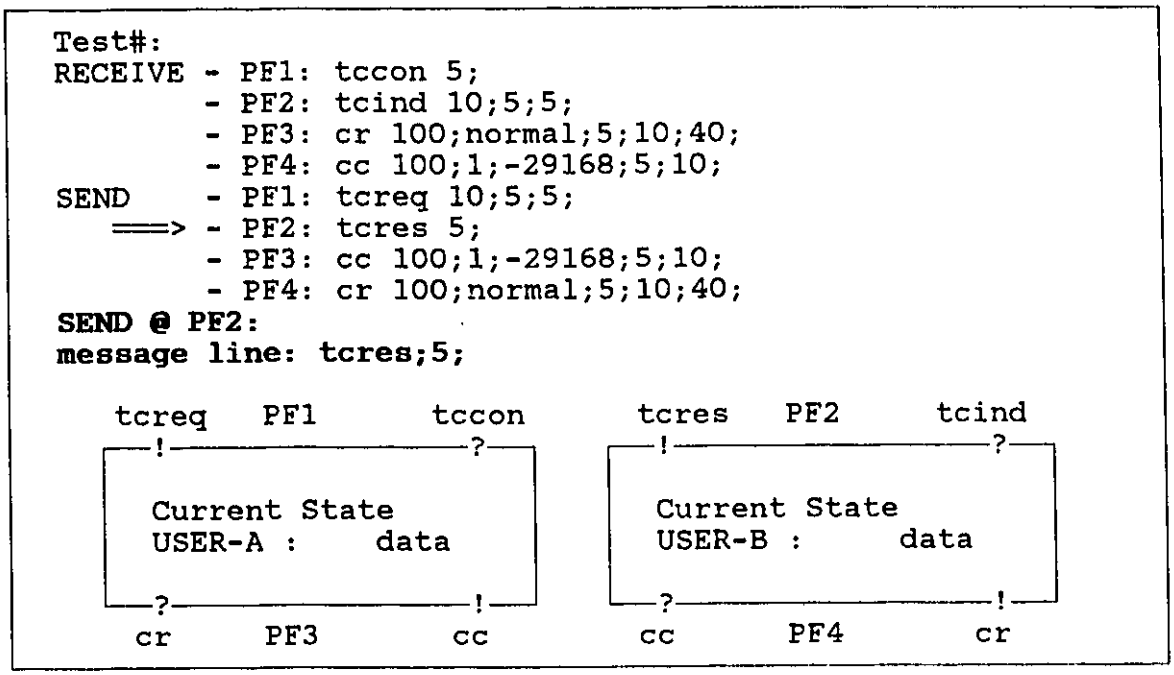
The second input sequence is the connection response from user B indicating the quality of service 5. The following steps are performed to send this sequence:

1) Enter"PF2" to select user B interaction point.

2) Enter "SELECT" to display the message line.

3) Enter the input sequence (tcres;5;).

In response to this sequence the display changes as follows:



Prototype Display after Sending tcres at User B Upper Level

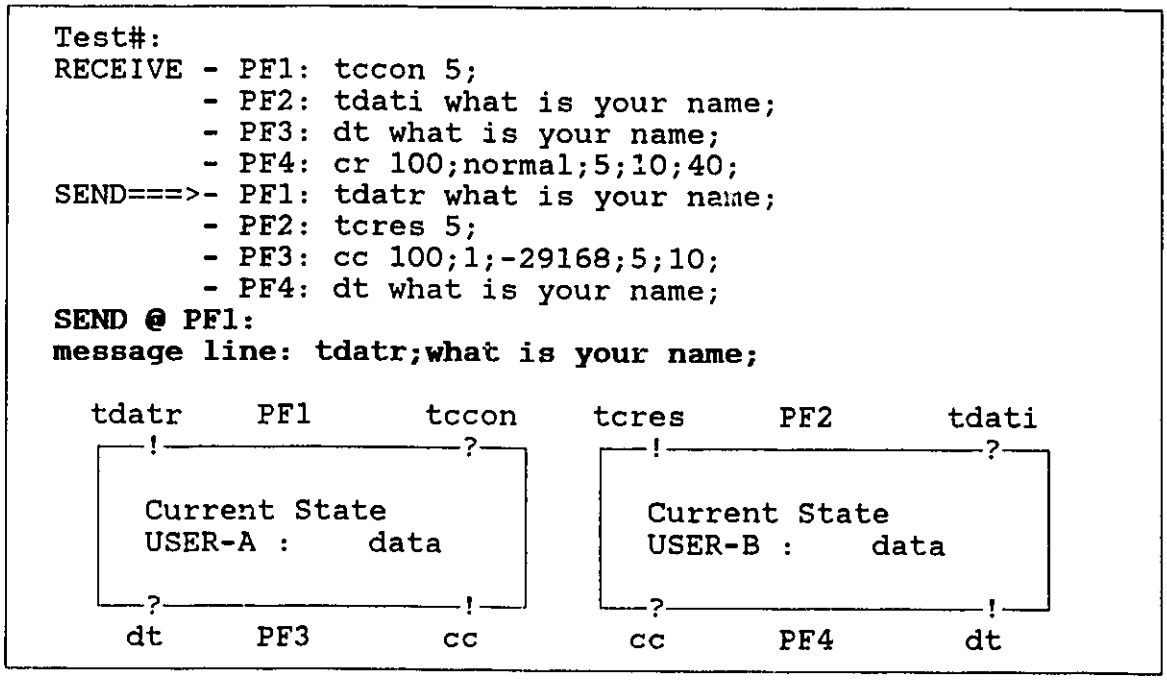
The third sequence is the data ("what is your name" exchange from user A to B. The following steps are performed to process this sequence:

1) Enter "PF1" to select user A interaction point.

2) Enter "SELECT" to display the message line.

3) enter the input sequence (tdatr;what is your name;).

This input sequence results in the changes to display as follows:



Prototype Display after Sending tdatr at User A Upper Level

For our second example we illustrate the FPCP User Interface with a TTCN test file. For this we selected **Option R - Remote Test Method - TTCN Test File** from the Prototype Execution Options Menu.

The Remote Test method allows the user to conduct testing of the prototype using test suites coded in TTCN format. This option expects a disk file in drive B named 'TTCN.PAS' containing the test suite.

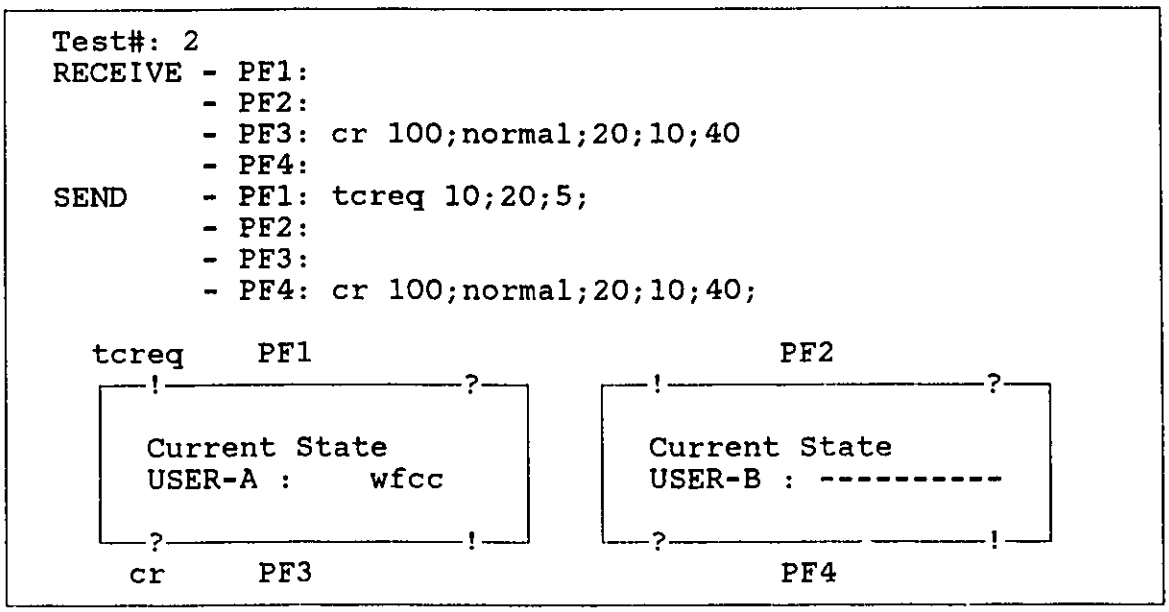
Under this mode the prototype gets its inputs from the TTCN file. At each step of the testing a complete picture of the protocol entities states and contents of the buffers are flashed on the screen. The prototype output is compared with the expected results specified in TTCN file. The results of this comparison is displayed as each test sequence is executed. At the end of each test the system pauses for the user to press any key to continue testing until all TTCN file test inputs are processed.

This example shows prototype testing with test suite coded in TTCN format. In this example the sample of transport(0) test suite (on file TTCN.PAS) in Appendix D is used. The FPCP prototype outputs for the following four steps is illustrated.

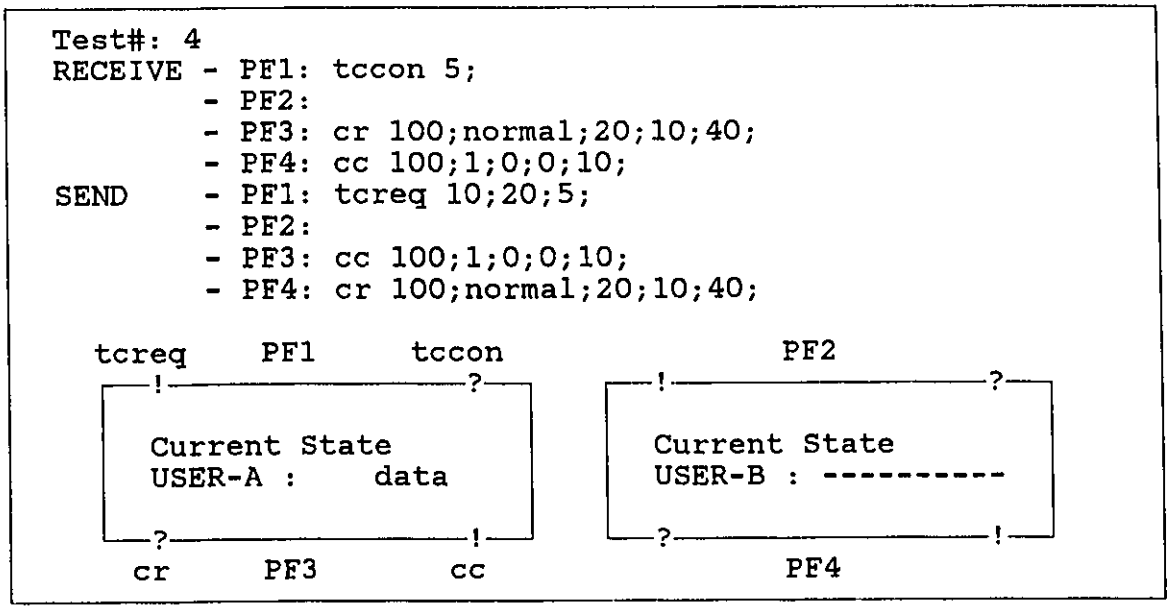
```
Step 1: U! tcreq d1 pass  (d1=10;20;5;).
        (Send tcreq at Upper Level with data d1).

Step 2: L? cr d2 pass (d2=100;normal;20;10;40;).
        (Receive cr at Lower Level with data d2).

Step 3: L! cc d3 (d3=100;1;0;0;10;).
        (Send cc at Lower Level with data d3).

Step 4: U? tccon d4 pass   (d4=5;).
        (Receive tccon at Upper Level with data d4).
```

For each test step which requires a "SEND" the FPCP User Interface prepares a tpdu (transport protocol data unit) from the given test sequence. The Interface also performs the necessary validation of the input sequences against the specification in the KB. The User Interface then sends this tpdu at an appropriate interaction point as specified by the test case. The Protocol Execution Control displays the protocol execution as shown with this example.

For each test step which requires a "RECEIVE" the FPCP User Interface compares the contents of the specified interaction point with the expected output data specified in the test case. This step does not result in any changes to the Prototype Display.



Prototype Display after Remote Test TTCN Steps 1 and 2



Prototype Display after Remote Test TTCN Steps 3 and 4

The User Interface for **Option L - Local Test Method - TTCN Test File** is similar to the "Remote Test Method" discussed above. Under the "Local Test Method" the prototype is executed as a single (local) entity as opposed to two remote peer entities as in "remote Test Method".

4.4 Uses Of FPCP Transport(0) KB and Prototype

4.4.1 Debugging Transport(0) Estelle Specification

The FPCP system can be used as a tool to debug formal protocol specification. Initially as the formal specification are encoded into KB, the system checks for and reports Estelle syntax errors. In addition to this the system performs some basic checks to ensure the consistency and integrity of the specification. This includes checks for undefined and redefined variables.

The KB display feature allows users to review the specification on screen. The FPCP displays provide more structured view of the specification and provide with different levels of details of the formal specification.
During the process of creating the executable prototype the system detects more types of specification errors. These errors are reported during the compilation of the generated Pascal program.

The interactive execution of the prototype allows users to debug the specification. More on this feature is mentioned in the following sections.

4.4.2 Validation and Analysis Of Transport(0) Test Results

The FPCP system generated prototype can be used to validate Transport(0) test suits against the formal specification. This can be achieved in one of the following two modes:

1) Interactive mode;
2) Batch mode.

In the interactive mode individual test sequences can be entered by the user for any of the module interaction points. The FPCP system verifies the format and the validity of the input. For this verification the system checks the current state of the module (from current values of the module variables) and the allowable Service Primitives at the given interaction point (from KB). Furthermore the format of the input Service Primitive is verified against the specification in the KB.

In the batch mode the user develops a test file in TTCN format containing protocol input (and expected output) sequences. The system performs the same set of verification checks as described above with the interactive mode. In the batch mode the system also verifies the actual output sequences against the expected output sequences in the test file.At each step as the output sequence is compared, the system indicates PASS/FAIL and pauses for user (to hit enter) to continue with the next test sequence.

4.4.3 User Training With FPCP Transport(0) Prototype

The FPCP system implementation environment can be used effectively for user training. The system provides for different types of support for training.

First, the KB Display feature allows users to walk-through the formal protocol specification. The KB displays structure is more user friendly then the formal specification in Estelle. With this feature a user can familiarize with the module states, transitions, interaction points, and the format of the interactions (Service Primitives).

The executable prototype allows users to enter module interactions at any of the module interaction points. The Execute Prototype display provides users with graphic view of module current state, contents of the channels, and processing of a transition.

The FPCP system also supports building of training scenarios on a file. The file is build as module valid (and invalid) input sequences and the correct output sequences in TTCN format on a batch file. The file can then be processed as batch of test input sequences (see batch mode in section 4.3.2). At each input step the FPCP system provides a graphical view of module external inputs, internal processing, status of current state, contents of channels, and module external outputs.

4.5 Assessment Of Experience With Application Of FPCP To Transport(0) Protocol

Using the FPCP system as if we were implementing the Transport(0) protocol has demonstrated the feasibility of our proposed approach. As well we have shown the potential effectiveness of the FPCP system to produce fast prototypes of communications protocols for debugging and training. A more detail discussion of our experience with FPCP follows.

As the first step of prototyping with FPCP system we created a text file containing the Transport(0) Normal Form Estelle specification. During this process we had to slightly modify the specification to accommodate the FPCP system limitations and extensions. More specifically we:

- truncated the names and identifiers to 12 characters;
- added transition names for each transition;
- simplified the Estelle OUT clause to OUT(asp)
  where asp is the appropriate abstract service primitive name.

The subsequent steps of encoding the specification into KB and creating the transport(0) prototype were completely automated. These steps normally took five to ten minutes each.

During the automated encoding process the system detected and reported a number of Estelle syntax errors.

The KB Display feature, which displays the specification with different levels of detail, proved to be a useful tool in understanding the formal protocol specification.

The system also detected some specification errors and ambiguities during the automated creation of the prototype. These errors were reported by the Pascal Compiler during the compilation of the generated Pascal code. However, this type of

error reporting was not user friendly. It required careful analysis of the specification to find the bugs in the Estelle file.

In summary, in our trial application to transport(0) protocol, the FPCP system provided a reasonably fast and efficient tool to automatically generate protocol executable prototypes. The built-in user interface provided the user with appropriate functions for debugging Estelle specification, validating test cases against the specification, and perhaps for building training scenarios, although we did not try to build training scenarios in our trial. The experience we gained during this trial implementation of FPCP allowed us to compare our approach with the related work in this area. In the next chapter we present an evaluation of the FPCP system and a comparison of proposed approach with the related work. This later application of FPCP needs to be investigated further, although in a sense, the TTCN file could be thought of a training scenario input sequences.

5. EVALUATION AND CONCLUSIONS

5.1 Summary Of FPCP Approach

In this thesis we demonstrated uesfulness of single authoritative reference specification (REFSPEC) approach to development and testing of communication software. In our approach the Estelle formal specification of a protocol remains the reference specification. These specification are automatically converted into a KB and are used in FPCP automated tools for debugging, testing, training, and interrogation of the specification. The automated conversion of the specification into KB allows users to take advantage of simple rule-based techniques without manual translation of formal specification into rules and facts.

Our objective was to demonstrate the feasibility of our approach as an integrated environment for protocol implementation. With the implementation of FPCP system we showed:

- automated encoding of formal specification into KB;
- inquiry against the KB(formal specification);
- automated generation of executable prototypes from the KB (formal specification);
- uses of FPCP generated KB and the prototypes in protocol implementation.

As an integral part of our approach , we also discussed:

- interrogation of KB (formal specification);
- interrogation of KB (formal specification plus the prototype execution traces);
- development of conformance test plan.

The above features are not implemented in the current version of the FPCP system. However, the design for these enhancements to FPCP system is included in section 5.2.

5.2 Suggested Extensions To FPCP System

This section describes some extensions to the FPCP system to enhance the usefulness of the system. A design of the proposed extensions is shown in figures 5.2.1 and 5.2.2. Details are described below.

5.2.1 Capture And Maintenance Of Execution Traces in KB

During the system debugging and system testing phases it is extremely useful for developers to obtain quick explanations of protocol behavior, i.e. to learn what in the source specification caused a specific protocol behavior. For example if a test conducted on the prototype did not produced the expected results, the test execution traces, if maintained by the system, can be interrogated to explain the prototype behavior and to debug the specification.

The FPCP KB interrogation facility suggested in section 5.2.2 would be able to provide this feature if the FPCP system design is enhanced to include traces of prototype execution in KB. The execution traces can be represented in KB as facts describing a test execution as series of transitions processed. For each transition processed the KB would include:

- current state;
- module variables;
- service primitives with parameters (interactions);
- transition selection trace (PROVIDED conditions satisfied);
- state after the transition (Next State);
- module variables after the transition.

In FPCP environment a prototype execution is controlled by an 'Execution Control' module. This control module executes a

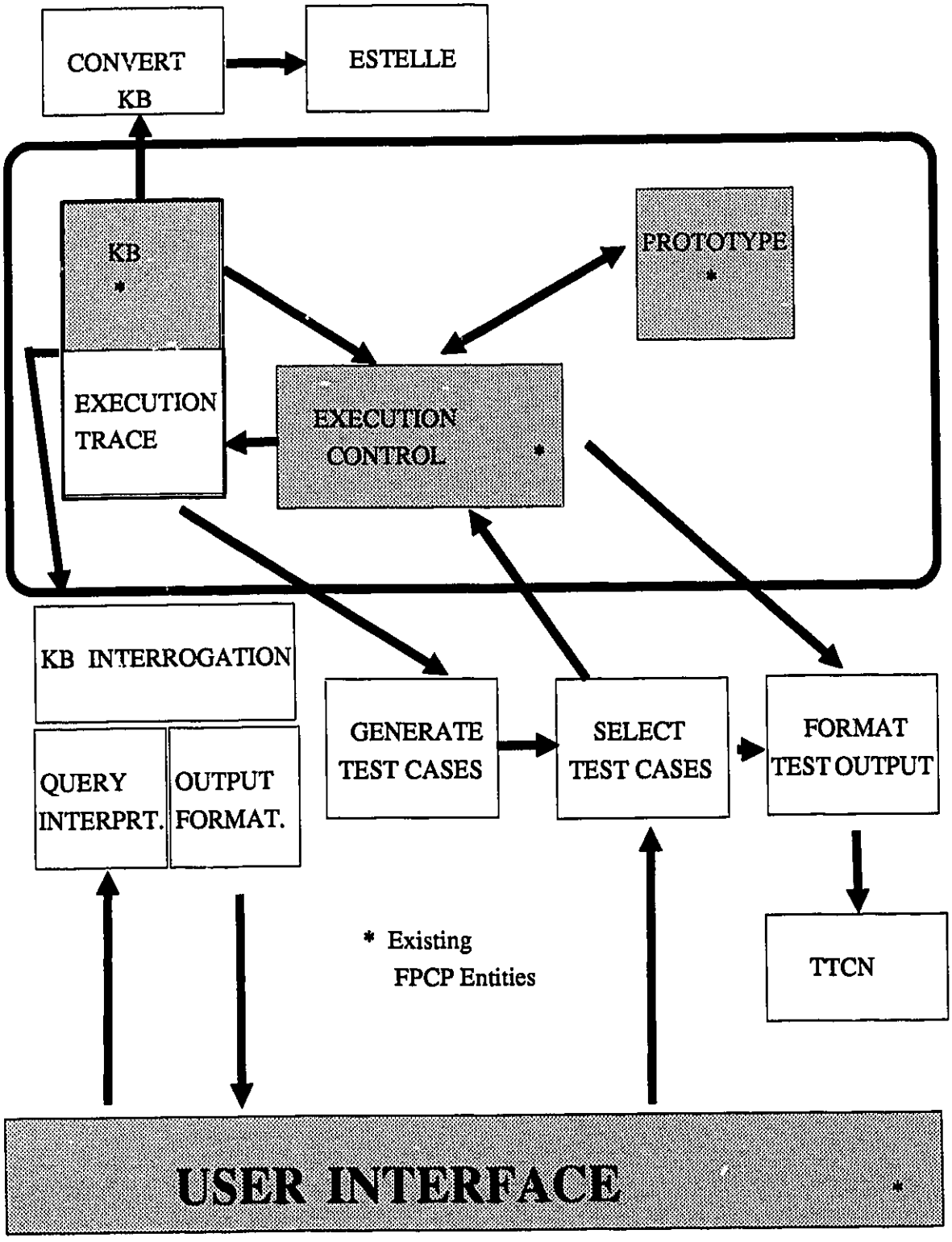


Figure: 5.2.1 PROPOSED ENHANCEMENTS TO FPCP SYSTEM

In FPCP environment a prototype execution is controlled by an 'Execution Control' module. This control module executes a protocol by using the protocol specification in the KB and the procedures generated by the FPCP system. This design makes it possible to capture the execution traces (as described above) as the prototype is executed. The paths along with the associated data can be maintained in the KB as execution traces.
The implementation of trace facts would allow developer a useful debugging tool. After executing an interaction sequence, the developer would be able to determine what in the formal specification is used to select the execution paths, and the values of module variables at each step of the execution.
Similarly, a test designer can interrogate the trace facts after executing a test sequence for conditions covered and not covered by the test sequences.

5.2.2 Interactive Interrogation Of KnowledgeBase

The interrogation of KB as mentioned in the earlier sections is an important addition to the FPCP system. The KB contains protocol specification in fact form which can be converted back to formal Estelle specification. Thus the interrogation of KB provides a means of interrogating the formal specification. In Figure 5.2.1, the interrogation mechanism proposed is as follows:

User queries are translated by a "Query Interpreter" into a prolog-like query against the KB. The translated query is processed by an inference process (shown in figure 5.2.1 as "KB Interrogation"). The results of the query is formatted into Natural Language by an "Output Format" process.

The KB interrogation mechanism proposed processes queries not only against the protocol specification but also against the protocol execution traces in the KB to provide explanations of protocol behavior.

5.2.3 Development Of Conformance Test Plan And Test Cases

The FPCP KB which contains the protocol specification can be used to develop specification-based test cases. The existing FPCP system features such as execution of test cases, and the suggested feature for capturing and maintaining execution traces provide support functions needed for the development and documentation of conformance test suites. For example specification-based test cases can be generated from the protocol specification in KB. The selected test cases can be executed by the FPCP prototype to document the test inputs and expected outputs.

An interactive test case generator can be developed to follow the captured execution paths (as defined in the KB) and present associated test cases to the user. Particular test cases can then be selected (interactively) or by means of an automated selection strategy [Short] [Ural87a]. The test case selection strategy is not discussed in this Thesis. Regardless of the selection strategy, the selected test cases can then be executed by using the existing FPCP Prototype Execution feature. Furthermore a separate module could be integrated into FPCP to format the test case inputs and outputs in an international standard test specification method such as TTCN.

As shown in figure 5.2.1 an automated process to generate specification-based test cases is integrated with the FPCP system. The generated test cases are presented to user for selection (or rejection). The selection process shown in the figure is manual. However, a selection strategy can be automated in the "Select Test Cases" process. The selected test cases input sequences are executed by the FPCP protocol prototype to automatically obtain the "expected outputs". The test cases (input sequences) and the test outputs (output sequences) are formatted in an international standard format (such as TTCN) by the "Format Test Output" process to document the conformance test plan.

5.2.4 Conversion Of KB into Estelle Specification

The protocol specification in KB (which were originally derived from protocol specification in Estelle format) can be re-converted to Estelle format. This may be required in the proposed "explanation" facility for explaining the protocol behavior (execution traces) in terms of source Estelle specification.

This facility will also be useful if the FPCP design is changed to obtain protocol specification from sources other than Estelle, such as from an on-line editor of the KB, or from example-driven specification techniques.

5.2.5 Verification Of TTCN Test Suites Against KB

The current version of the FPCP system includes a tool to execute the prototype to check the test sequences stored in a file in TTCN format. In this process the input sequences are obtained from the file and the prototype output sequences are compared with the expected outputs on the test file. During this process the system performs static checks of the test input sequence format against the specification in the KB.

This feature of FPCP system can be enhanced to not only validate the format of the test sequences but also to automate the analysis of differences between the actual and expected output sequences. This analysis could include explanations of the actual outputs in terms of the formal protocol specification. Refer to Figure 5.2.2 for a design proposal for this feature in the FPCP environment.

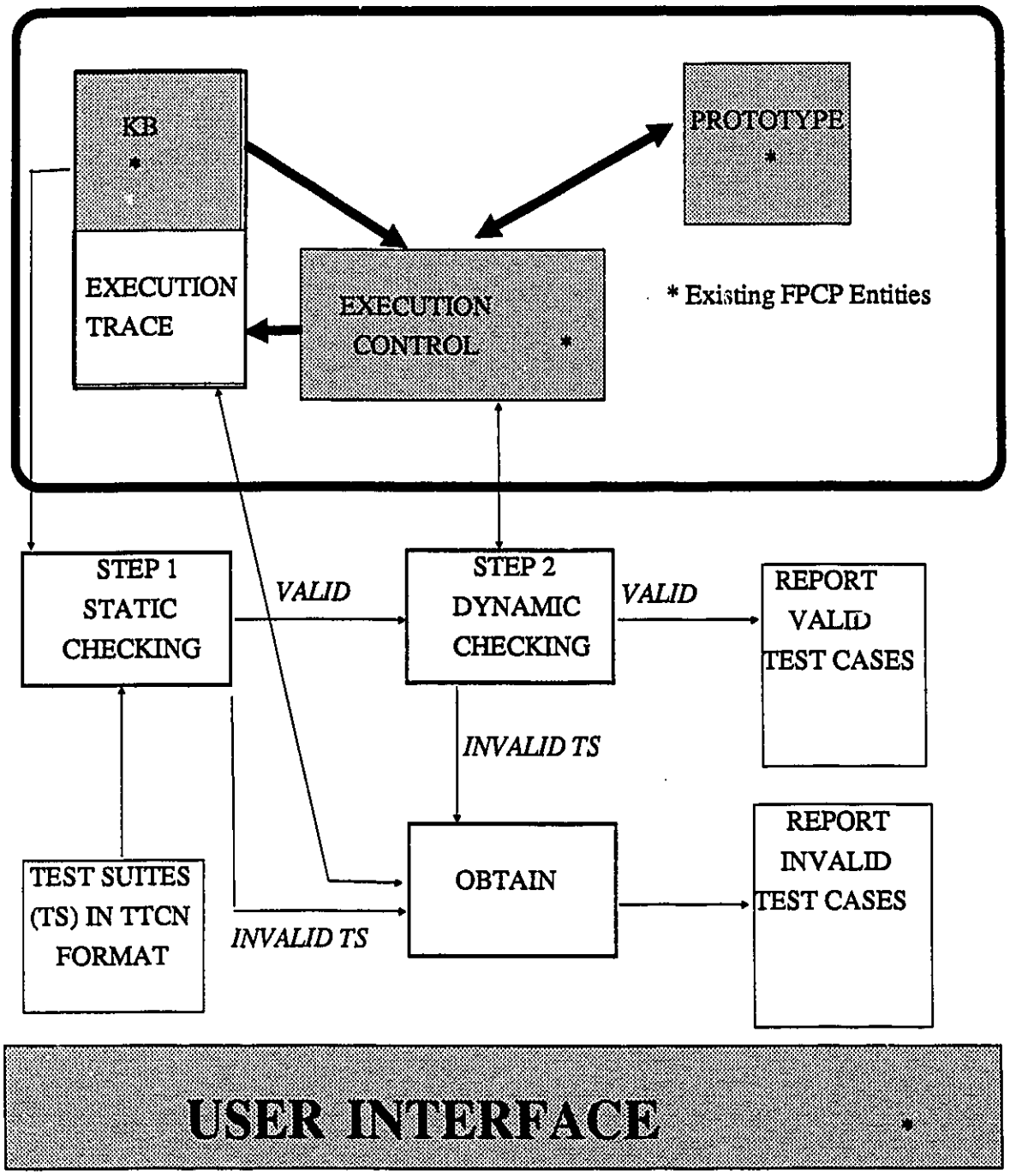


Figure: 5.2.2 Verification Of TTCN Test Suites Against KB Page: 5-7

As shown in figure 5.2.2 test suites documented in TTCN format are validated in a two step process. As the first step the test sequences are checked against the protocol specification in KB by the "static Checking" process. During this process the test inputs (and expected outputs) interaction format is checked against the format specified in Estelle (KB).

In the second step, syntactically valid test suites are executed by the "Dynamic Checking" process. During this process the actual output obtained from the execution is compared with the expected output. Where the actual output differed from the expected output, the "Obtain Explanation" process would automatically report the test execution traces to explain the actual output.

5.2.6 Summary Of Recommended Enhancements

The conversion of KB into Estelle specification (section 5.2.4) and the proposed enhancements to maintain the prototype execution traces in KB (section 5.2.1) are pre-requisite to the other modifications proposed in this chapter.

The feature to convert KB into Estelle would require minimum effort to implement. The enhancements to maintain execution traces in KB would require detailed analysis to design the internal representation of the execution traces as "facts" in KB. These two enhancements, once implemented would provide an important step in developing the other proposed enhancements.

After the two pre-requisite modifications are made, we recommend the development of KB interrogation feature (section 5.2.2) as the next step. The capability to interrogate the formal protocol specification and and the execution traces would provide developers with a powerful tool for protocol specification debugging and protocol testing.

5.3 Comparison Of FPCP to Related Approaches

The automated encoding of formal protocol specifications has been proposed in various studies over the last few years. The approaches presented in these studies included the encoding of the formal specification into programming languages such as Pascal [Hansson], C [Vuong] [Amalu], or in machine language [Saqui].

The encoding of formal specification into high-level programming languages provides a fast executable prototype. However our approach of encoding the specification into a KB (which is used at run-time) not only provides a fast executable prototype it also provides a means to interrogate the formal specifications.

Furthermore, our scheme allows us to build user-friendly interfaces with the generated prototypes. This is possible because we maintain the dynamic part of the specification on the KB, and use the KB at run-time. In some ways our approach is similar to the one developed at the University Of British Columbia [Vuong]. Under their approach the generated C program (like our KB) contained only the implementation depended details, which are linked and compiled with a set of pre-defined run-time support C routines.

Similar approach is being used in the Estelle Simulator Prototype (ESTIM) discussed in section 1.4 [Saqui]. Under their approach, the protocol implementation dependent specifications are translated into machine language, and a separate user interface and interpreter is developed to simulate the protocol.

At the University of Ottawa, in a similar study a LOTOS interpreter (ISLA) is developed and implemented [Logrippo]. ISLA translates the formal specification into a Prolog list. In this project the researchers demonstrated the usefulness and the

feasibility of maintaining the prototype execution traces to provide details of prototype behavior.

The maintenance of execution traces is an integral part of our approach. As noted earlier, our process of encoding the specification into KB is reversible. Therefore with the FPCP approach we can provide the explanation of the prototype behavior in terms of the original Estelle specifications. We keep the KB internal representation transparent to users. The formal specification remains the single authoritative reference during all phases of protocol implementation.

In this thesis we discussed the development of conformance test plan in the FPCP environment. Although we did not present any methodology for generating the test cases, we presented the design for integrating a specification-based test generation approach into FPCP. Several methods, automated and semi automated, have been proposed in recent years [Boyce] [Probert88] [Logrippo] [Linn87] [Saqui]. These specification-based test generation techniques, if incorporated in FPCP will greatly increase the usefulness of the FPCP environment for protocol implementation.

Finally, with the trial implementation of the FPCP system, we presented an integrated approach for the implementation of communication protocols. We tried to address the need for automated tools in all phases of the protocol implementation. This included tools for debugging formal specifications, prototyping, testing, verification, and user training.

In the next section we present the design of some useful enhancements to FPCP system. The requirements for these enhancements have been discussed earlier in chapter 2.

5.4 Conclusions

In this thesis we have presented a Prototyping-based approach to the implementation of communication protocols. In our proposed approach a KnowledgeBase is developed from formal (Estelle) specification of the protocol. The KB is invertible (convertible into formal Estelle specification). The KB representation remains transparent to users. The user interactions with the FPCP system are in Estelle format. Thus we maintained the Estelle protocol specification as the single authoritative reference during all phases of protocol development. The KB provided the means of interrogating the specification and of generating an executable prototype of the protocol.
The KB is then integrated with the generated executable prototype to provide such facilities as:

- Debugging and refinements of specification;
- Development of Conformance Test Suites;
- Validation and analysis of IUT test results;
- User training.

We have designed and implemented an interactive system "Fast Prototyping Of Communication Protocols" (FPCP) to demonstrate the feasibility of our proposed approach and methodology. In the FPCP system we developed and implemented an automated process to encode formal protocol specification into machine processable KnowledgeBase (KB). The KB is transparent to the users. The formal protocol specification remain as the single authoritative reference for the protocol development life cycle.

The FPCP user interface with the specification in KB increases the usefulness of the Estelle formal description, which without the use of automated tools are restricted and difficult to comprehend.

The design of FPCP prototype environment is flexible enough to integrate protocol development tools such as Interrogation of specification, Maintenance of execution traces to provide explanation of prototype behavior, Development of conformance test suites, and Verification of protocol conformance test suites against the specification.

The FPCP protocol implementation environment allowed us to develop user friendly interfaces with the prototype. Unlike automatically generated prototypes which are generated by converting the specification into high-level programming languages like PASCAL, C or Prolog, the FPCP prototype environment contains a generic user friendly interface which hides its use of the KB at run-time.

For reasons of limited availability of other resources the FPCP system was developed on a DEC Rainbow micro computer. Although not all features of the proposed system are implemented in the current version of FPCP, initial system test results were extremely encouraging.

As an evaluation, the FPCP system was used to automatically convert the Transport(0) protocol specification from Estelle into KB. During this process, specification syntax errors were reported by FPCP. Thus this process, in addition to developing the KB, can be used as specification checker for developers.

We conducted various tests by executing the generated Transport(0) prototype. This included protocol testing in Local and Remote modes. Throughout these tests the KB remained transparent. Our approach did not require a new syntax or format for protocol implementation. The formal Estelle specification remained as the single authoritative reference for protocol specification and testing.

Our design of the generated prototype, in Pascal using the KB at run-time proved very effective. It led naturally to development of user friendly interfaces with the prototype. In the current version of the FPCP system, we developed user interfaces to execute the generated prototype in Local Mode and Remote Mode interactively or on a file containing test suites in TTCN format. In section 5.1 we proposed additional potentially useful interfaces as enhancements to the FPCP system.

In developing the FPCP system there were some technical problems. The system was developed on the Rainbow micro computer operating under CP/M with only 64K main memory. This resulted in various limitations in the FPCP system such as the size of the KB and the number of features which could be implemented. However the system was developed in Turbo Pascal which is widely used and available on other micro computer environments. The feasibility of the FPCP implementation under these severe constraints guaranties feasibility on most common systems.

The system is portable to any micro computer with 64K main memory and 128K of virtual disk operating under CP/M. In developing the FPCP system the CP/M specific features were avoided where possible. The exceptions to this are the use of Virtual Disk (MDRIVE feature) and the Compiler options. The conversion of FPCP onto other micro computer environments with Turbo Pascal is expected to be simple.

The implementation of FPCP onto a more current hardware/software environment would be essential to incorporate the proposed enhancements. This may also be required to eliminate or reduce some of the limitations in the system such as:

- limitation on the maximum number of variables;
- limited size of variable name;
- manual interface to modify Estelle file when Estelle error is detected;

- termination of encoding after the first error is detected;
- User supplied file names instead of the hard-codes names;
- No parsing of Estelle procedure statements.

In developing algorithms for encoding the Estelle specification into KB a simplified, but adequate format of Estelle is assumed. This was done to keep the scope of the FPCP system to a reasonable size. Furthermore the exceptions processing modules and the queuing discipline (Queued or Not Queued) are not implemented in the system. In the FPCP system all interactions are queued. The Estelle delay clause is also not included in the current version of the system. Minor modifications to the FPCP encoding process and the generic "prototype execution eontrol" process can be made to implement this clause.

Despite the limitations mentioned above our thesis objectives were realized by designing and developing an experimental FPCP system. This system has been demonstrated to be useful for implementation of part of a communication protocol. The system limitations mentioned in earlier sections were mostly due to the Hardware and Software limitations of the micro computer on which the system was developed. These limitations can be eliminated easily by converting the system onto any common micro computer with adequate hardware and software capabilities.

Finally, the potential extensions to FPCP system which are described earlier in this chapter if implemented would significantly improve the usefulness of the FPCP system and that of our approach to Fast Prototyping Of Communication Protocols. It is highly recommended that these enhancements be implemented before further studies of operational feasibility are carried out.

6. REFERENCES

[Amalu] C. Amalu "A tool For Automated Prototyping Of Communication Protocols", M. Sc. Thesis, University Of Ottawa, Ottawa, Ontario, 1986,pp 1-136.

[Barbeau] M. Barbeau "FTAM Test Design Using an Automated Test Tool" IEEE 1989, pp 251-259.

[Bochmann] G. Bochmann "Usage of Protocol Development Tools: The Results of a Survey" Protocol Specifications Testing and Verifications Vol 7, 1987 pp 139-153.

[Boyce] T. Boyce, T. Grenier, R.l. Probert, H. Ural "Formalization Of ISDN LAPD For Conformance Testing" IEEE 1989, pp 234-246.

[Clocksin] Clocksin and Mellish "Programming in Prolog" Springer-Verlag Berlin Heidelberg NewYork 1981.

[Cohen] D. Cohen "A Forward Inference Engine to Aid in Understanding Specifications" , University Of Southern California, ISI/RS-84-135, August 1984, pp 1-5.

[Dahl] Dahl "Logic Programming as Representation Of Knowledge" COMPUTER, Oct 1983, pp 106-111.

[Fleischmann] A. Fleischmann "PASS - A Technique For Specifying Communication Protocols" Protocol Specifications Testing and Verifications Vol 7, 1987, pp 61-76.

[Hayes85] Hayes-Roth "Rule Based Systems" Comm. Of The ACM, Sept 1985, Vol 28, No: 9, pp 921-941.

[Hayes84] Hayes-Roth "The Knowledge Based Expert System: A Tutorial" COMPUTER, Sept 1984, pp 11-28.

[Hammond] P. Hammond "Micro Prolog For Expert Systems" Micro-Prolog: Programming in Logic, pp 294-319.

[Hammond84] P. Hammond, Sergot "APES: Augmented Prolog For Expert Systems", Logic Based Systems Ltd, England, July 1984, pp 1.1-7.6.

[Hansson] Hansson "Automatic implementation of formal descriptions of communication protocols", Protocol Specifications Testing and Verification. IFIP 1986, pp259-267.

[Isreal] Isreal, Beranek, Newman "The Role Of Logic in Knowledge Representation" COMPUTER, Oct 1983, pp 37-41.

[ISO86a] Information Processing Systems - Open Systems Interconnection - "ESTELLE - A Formal Description Technique Based on an Extended State Transition Model", 1986, pp 1-107.

[OSI86b] "OSI Conformance Testing Methodology and Framework. Part 2: Annex D: The Tree and Tabular Combined Notation", September 1986, pp 1-27.

[ISO87a] "BNF Definition of Syntax of Tree and Tabular combined Notation for the test suite specification", January 1987, pp 1-6.

[ISO87b] "US comments on OST Conformance Testing Methodology and Framework - Part 5: Test Execution", January 1987, pp 1-33.

[ISO87c] "X.25 DTE Packet Level Conformance Test Suite, a working paper for the proposed DP 8882 part 3", March 1987, pp 1-11.

[ISO87d] "Draft answers to TTCN study items from SC21 WG 1", held in Nice, France, April 1987, pp 1-8.

[ISO87e] "Issues in TTCN, study items for SC21 group on development of test notation", April 1987, pp 1-8.

[Jakobson] Jackobson, Shakid, Rowley, & Crystal "Dialog Design Technology For Cooperative Man-Machine Communication Systems", IEEE 1983, pp 255-260.

[Johnston] S. Johnston "SPIDER- Service and Protocol Interactive Development Environment" FORTE88, 1st International Conference On Formal Description Techniques, September 1988, pp 67-71.

[Kingston] Kingston "Standards for Conformance Testing", Network Technology Division, Bell Northern Research. January 1987.

[Linn] Linn Jr. "Features and Facilities Of Estelle" National Bureau of Standards, Institute Of Computer Science and Technology, Systems and Network Architecture Division, Garthersburg, USA, 1986, pp 271-296.

[Linn86] Richard Linn Jr "Testing To Assure Interworking Of Implementation of ISO/OSI Protocols", Computer Networks and ISDN Systems 11 (1986), pp 277-286.

[Linn87] R. Linn Jr. and J-P Favreau "Automatic Generation Of Test Scenario Skeletons From Protocol Specifications Written in Estelle", Protocol Specifications Testing and Verification. IFIP 1987, pp 191-202.

[Logrippo] L. Logrippo, M.Haj-Hussein and R. Guillemot "Executing Large LOTOS Specifications", Protocol Specifications, Testing and Verification Vol 8, 1988, pp 399-440.

[Mackert] L. Mackert and I. Neumeier-Mackert "Communicating Rule Systems" Protocol Specifications Testing and Verification Vol 7, 1987, pp 77-88.

[Mizuno] T. Mizuno, J.Munemori, F. Sato, T. Nakakawaji and K. Katsuyama "COTTAGE: Systematic Method For The Development Of Communication Software: Protocol Specifications Testing and Verification Vol 8, 1988, pp 269-280.

[Pappalardo] G. Pappalardo "Experiences With a Verification and Simulation Tool For Behavioral Languages" Protocol Specification Testing and Verification Vol 7, 1987, pp 251-253.

[Probert83] R. Probert, Skuce & Ural "Specification Of Representative Test Cases", Proceedings of the Sixteenth Annual Hawaii International Conference on System Sciences, 1983, pp 190-196.

[Probert86] R. Probert "Test Specification Languages For Open Systems Interconnection". Presentation to IFIP Workshop 1986 on Protocol Specification, Testing and Verification, June 11, 1986.

[Probert83] R. Probert "REFSPEC Based Software Development Methodology and Environment". Working Papers from June 1983 to September 1983, Data Network Division, Bell-Northern Research, Ottawa, Ontario.

[Probert88] R. Probert, H. Ural and M. Hornbeck "An Integrated Software Environment For Developing and Validating Standardized Conformance Tests" Protocol Specifications Testing and Verification Vol 8, 1988, pp 87-98.

[Saqui] P. de Saqui-Sannes and J-P. Courtait "ESTIM: The Estelle Simulator Prototype Of The ESPRIT-SEDOS Project"

FORTE88, 1st International Conference On Formal Description Techniques, September 1988, pp 15-27.

[Sarikaya] B. Sarikaya and G. Bochmann "Obtaining normal form specifications for protocols" Proc. of COMNET 85, Budapest Hungary, Oct 1985, pp 6.133-6.149.

[Shiratori] N. Shiratori, K.Takahashi and S. Noguchi "An Intelligent User-Friendly Support System For Protocol and Communication Software Development" Protocol Specification Testing and Verification Vol 8, 1988, pp 257-268.

[Short] R. Short "An Interactive Test Sequence Generator", M. Sc. Thesis, University Of Ottawa, Ottawa, Ontario, 1986, pp 1-63.

[Tenney] R. Tenney "A Tutorial Introduction To Estelle" FORTE88, 1st International Conference On Formal Description Techniques, September 1988, pp 1-37.

[Ural85] H. Ural and R. Probert "Step-Wise Validation Of Communications Protocols and Services". University Of Ottawa, Ottawa, Ontario April 1985, pp 1-26.

[Ural87] H. Ural "A Test Derivation Method For Protocol Conformance Testing", University Of Ottawa, Ottawa, Ontario, TR-87-04, January 1987.

[Ural87a] H. Ural and R. Short "An Interactive Test Sequence Generator", University Of Ottawa, Ottawa, Ontario, 1987, pp 1-19.

[Ural84] H. Ural and R. Probert "Testing Specifications and Designs Of Communication Protocols and Services", University Of Ottawa, Ottawa, TR-84-18, 1984, Ontario, pp 1-36.

[Vuong] S.Vuong, R.Chan and W. Chan "An Estelle-C Compiler For Automatic Protocol Implementation", Protocol Specification, Testing and Verification, Vol 8, 1988, pp 387-394.

[Wiles] Wiles "ITEX - An Interactive TTCN Editor and Executor", Swedish Institute Of Computer Science, 1986, pp 1-13.

[Winston] Winston, Prendergast "The AI Business, The Commercial Uses Of Artificial Intelligence", The MIT Press, Cambridge, Massachusetts, pp 1-13.

APPENDIX A - Sample Of Transport(0) Protocol Specifications In ESTELLE

The following pages contain a partial listing of transport(0) protocol Normal Form Estelle Specifications.

FILE : ESTL.PAS

NORMAL FORM ESTELLE SPECIFICATION OF TRANSPORT(0) PROTOCOL EXAMPLE IN CHAPTER 4.

```
const
     null     ='';
     zero      =0;
     ok_qts   =5;
     ok       ='ok';
     ok_option='normal';
     tpdusize =10;
     qtsest   =5;
(* type declarations*)
type
     int    = integer;
     cmt    = string[12];
     dta    = string[40];
     adr    = integer;
     rsn    = string[12];


(*************************************************)
(*       channel declaration                     *)
(*************************************************)

channel u_access_pt (user,provider);
      by user:
               tcreq     (to_t_adr     : adr ;
                          from_t_adr   : adr;
                          qts_req      : int);

               tcres     (qts_req      : int  );

               tdreq     (ts_usr_reson: rsn  );

               tdatr     (tsdu_dta     : dta);


      by provider:

               tcind     (to_t_adr     :adr;
                          from_t_adr   :adr;
                          qts_pro      :int);

               tdind     (ts_ds_reson : rsn;
                          ts_usr_reson: rsn);

               tdati     (tsdu_dta     :dta);

               tccon     (qts_res      :int );


channel n_access_pt (user , provider);
      by user:
               cr         (source_ref   : int;
```

```
                        option       : cmt;
                        clg_t_adr    : adr;
                        cld_t_adr    : adr;
                        max_tpdu_sz  : int);

              dr        (dest_ref    : adr;
                        disc_reson   : rsn;
                        ad_clr_reson: rsn);

              dt        (usr_dta     : dta);

              cc        (
                        dest_ref     :adr;
                        source_ref   :adr;
                        clg_t_adr    :adr;
                        cld_t_adr    :adr;
                        max_tpdu_sz  :int);

              ndreq     (disc_reson  : rsn);


      by provider:

              nrind     (disc_reson  : rsn);

              ndind     (source_ref  : adr);

module trans_class0
                  (u : u_access_pt (provider) queued;
                   n : n_access_pt (user)     not queued);


state    idle  ,    wftr  ,     wfcc  ,    data          ;

stateset waiting = (  wftr  ,   wfcc    );


var

     local_ref      :adr;
     tpdu_sz        :int;
     clg_t_adr      :adr;
     cld_t_adr      :adr;
     source_ref     :adr;
     option         :cmt;
     remote_ref     :adr;
     qts_est        :int;
     buffer         :dta;
```

```
trans t1 from idle to wfcc when n.tcreq  provided tcreq.qts_req = ok_qts

        begin lccal_ref:= 100;
              tpdu_sz:= 40;
              clg_t_adr:=tcreq.from_t_adr;
              cld_t_adr:=tcreq.to_t_adr;

              cr.source_ref:=local_ref;
              cr.option:='normal';
              cr.clg_t_adr:=clg_t_adr;
              cr.cld_t_adr:=cld_t_adr;
              cr.max_tpdu_sz:=tpdu_sz;

              out( cr );
         end;

trans t2 from idle to idle when n.tcreq  provided tcreq.qts_req<>ok_qts

        begin tdind.ts_ds_reson:= 'not ok?';
              tdind.ts_usr_reson:='req.qts?';

              out (  tdind   );
        end;

trans t3 from idle to wftr when t.cr
                    provided (cr.max_tpdu_sz=zero) and (cr.option=ok_option)
        begin remote_ref:=cr.source_ref;
              tpdu_sz:=cr.max_tpdu_sz;
              clg_t_adr:=cr.clg_t_adr;
              cld_t_adr:=cr.cld_t_adr;
              qts_est:=qtsest;
              tcind.to_t_adr:=cld_t_adr;
              tcind.from_t_adr:=clg_t_adr;
              tcind.qts_pro:=qts_est;

              out(tcind);
         end;

trans t4 from idle to wftr when t.cr
         provided (cr.max_tpdu_sz<>zero) and (cr.option=ok_option)
         begin remote_ref:=cr.source_ref;
               tpdu_sz:=tpdusize;
               tcind.to_t_adr:=cr.cld_t_adr;
               tcind.from_t_adr:=cr.clg_t_adr;
               tcind.qts_pro:=qtsest;

               out(tcind);
         end;


trans t5 from idle to same when t.cr provided cr.option<>ok_option
```

```
          begin dr.dest_ref:=cr.source_ref;
                dr.disc_reson:='cant do';
                dr.ad_clr_reson:='      ';

                out(dr);
          end;

trans t6 from wfcc to data when t.cc provided cc.max_tpdu_sz<>zero
          begin tccon.qts_res:=qtsest;

                out(tccon);
          end;

trans t8 from wfcc to idle when t.dr provided dr.disc_reson='userinit'
          begin ndreq.disc_reson:=dr.disc_reson;
                tdind.ts_ds_reson:=dr.disc_reson;
                tdind.ts_usr_reson:=dr.ad_clr_reson;

                out(ndreq);

                out(tdind);
           end;

trans t9 from wfcc to idle when t.dr provided dr.disc_reson<>'userinit'
          begin ndreq.disc_reson:=dr.disc_reson;
                tdind.ts_ds_reson:=dr.disc_reson;

                out(ndreq);

                out(tdind);
           end;

trans t10 from wftr to data when n.tcres provided tcres.qts_req<=qtsest
           begin local_ref:=1;
                 cc.dest_ref:=remote_ref;
                 cc.source_ref:=local_ref;
                 cc.clg_t_adr:=clg_t_adr;
                 cc.cld_t_adr:=cld_t_adr;
                 cc.max_tpdu_sz:=tpdu_sz;

                 out(cc);
            end;

trans t11 from wftr to idle when n.tcres provided tcres.qts_req>qts_est
          begin dr.dest_ref:=remote_ref;
                dr.disc_reson:='not aval';
                dr.ad_clr_reson:='clear';
                tdind.ts_ds_reson:='????';

                out(dr);

                out(tdind);
```

```
        end;

rans t12 from wftr to idle when n.tdreq
        begin dr.disc_reson:='abcd';
              dr.ad_clr_reson:=tdreq.ts_usr_reson;
              dr.dest_ref:=remote_ref;

              out(dr);
        end;

rans t14 from data to data when n.tdatr
        begin dt.usr_dta:=tdatr.tsdu_dta;

              out(dt);
        end;

rans t16 from data to data when t.dt
        begin tdati.tsdu_dta:=dt.usr_dta;

              out(tdati);
        end;

rans t17 from data to idle when n.tdreq
        begin ndreq.disc_reson:=tdreq.ts_usr_reson;

              out(ndreq);
        end;

rans t18 from data to idle when t.ndind
        begin tdind.ts_ds_reson:='urgent';

              out(tdind);
        end;

      trans t19 from data to idle when t.nrind
                begin tdind.ts_ds_reson:='v.urgent';
                      out(tdind);
                end;


      end.
      end.
```

Appendix B - Sample Of Transport(0) Protocol Test Cases In TTCN Format

FILE : TTCN.PAS

```
testid  t1 test connection request;
u! tcreq d1 pass
   l? cr d2 pass
      l! cc d3
         u? tccon d4 pass
            l!  dt d5
              u? tdati d5 pass
                 u! tdatr d6
                    l?  dt d6 pass
                      u! tdreq d7
                         l? ndreq d7
                         l? nrind d7
   l? ndreq d7

testid t5 test data transfer
u! tcreq d1
   l? cr d2 pass

data
d1=10;20;5;
d2=100;normal;20;10;40;
d3=100;1;0;0;10;
d4=5;
d5=hi mujeeb;
d6=i got your message;
d7=going home;
end;
```

APPENDIX - C : FPCP SYSTEM USER GUIDE

C-1 SYSTEM DESIGN

C-2 HOW TO COMPILE FPCP SYSTEM

C-3 HOW TO START FPCP SYSTEM

C-4 SYSTEM FEATURES

C-5 SAMPLE SESSION

GENERAL

This appendix contains the description of the system design and features of The Fast Prototyping Of Communication Protocols system (FPCP). It also includes the procedures for compiling and executing the FPCP system. In section C-5 a sample session is included. Some sections of chapter 3 and chapter 4 are duplicated in this appendix for completeness.

C-1: SYSTEM DESIGN

The FPCP system (prototype version) was developed on a DEC RAINBOW microcomputer under the CP/M-86/80 Version 2.00 (1.1) operating system. The entire system was developed in TURBO PASCAL Version 3.00A. The DEC system had only 64K main memory, two 360K five and a quarter inch floppy disk drives, and 2 blocks of virtual disk memory, each 64K.

The run-time FPCP system organization is illustrated in Figure C-1. Disk drive A contained a working copy of the operating system, and the TURBO PASCAL Compiler. The second drive (drive B) was used for FPCP system program and data files. The two blocks of virtual memory were used to maintain the KnowledgeBase in memory. This reduced the number of I/Os and improved the performance of the system significantly.

The FPCP system file size exceeded the limits of the Pascal Editor. Therefore the program file had to be divided into a series of smaller files. Each program file contained a subsystem or series of FPCP procedures. The main program file contained the necessary **include** instructions to include all program files when compiling the system.

The FPCP system design is divided into seven processing subsystems as shown in Figure C-1. Five of these subsystems (dark rectangles) are implemented in the current version of FPCP.

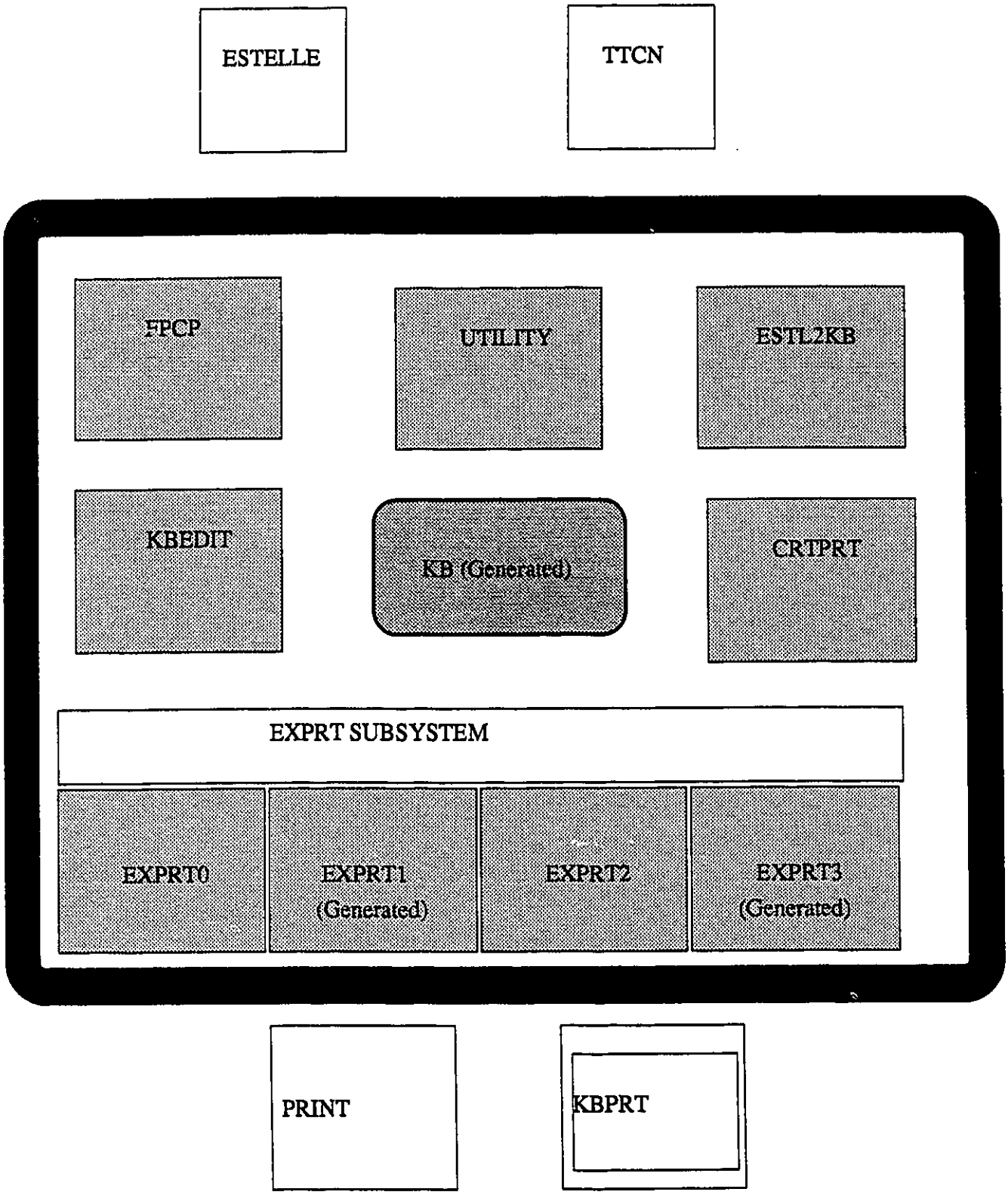


Fig: C-1 FPCP SYSTEM COMPONENTS

There are two Pascal program files which are generated by the FPCP system itself. More details on this is provided in the following sections.

The remainder of this section contains the list and description of the system files.

PROGRAM FILES	D E S C R I P T I O N
FPCP	This is the main program file. It contains the include instructions for Pascal Compiler. The file also contains the necessary code to present and process the system MAIN MENU options.
UTILITY	This file contains variable declarations and several utility procedures.
ESTL2KB	This file contains the subsystem which encodes an ESTELLE file into KB.
KBEDIT	This file contains KB display/Edit Subsystem programs.
CRTPRT	This file contains the subsystem to create the prototype of a protocol
EXPRT0 EXPRT1 EXPRT2 EXPRT3	This file and the following three files: (EXPRT1,EXPRT2,EXPRT3) contain the programs to execute the generated prototype. The files EXPRT0 and EXPRT2 are hard coded. The other two files

	(EXPRT1 & EXPRT3) are generated by FPCP system to prototype a specific protocol
PRINT	This is a standalone print program developed to print the system program and data files.

DATA FILES	D E S C R I P T I O N
ESTL	This file contains ESTELLE specification of a given protocol.
TTCN	This file contains test suites in TTCN format.
KB	This file contains the contents of KB on diskette.
KBPRT	This file contains the contents of KB in print format. The file is used for FPCP system debugging purposes only.

Note: The data file names also contain the file extension of '.PAS'. They are not Pascal programs. This convention is used only to facilitate the creation and editing of the data files through the Pascal Editor.

DETAILED DESCRIPTION OF THE SYSTEM COMPONENTS

File : FPCP.PAS

General	This file contains procedures declarations and the include instructions to include the other files.
Main Program	The main program displays the system main menu and calls the appropriate subsystem to process user selected option

File : UTILITY.PAS

General	This file contains the declarations and the utility procedures The utility procedures are described below.

Procedures

nv	normal video mode
lv	low video mode
hlin	horizontal line
vlin	vertical line
disp-id	display an identifier

erase-id	erases an identifier from screen
erase-ln	erases a line from screen
disp	displays a string of characters on screen
disp-ch	displays a character on screen
disp-sp	displays a space on screen
initialize	initializes system variables and virtual disk
nextrc	reads next record from KB
setnxt	sets pointer to next KB record
addrec	adds a record to KB
cghrec	changes a record in KB
delrec	deletes a record from KB
getrec	gets a record from KB
getkey	gets the key pressed on the keyboard
printkb	creates a print file of KB
finrec	finds a record (of identifier) from KB
switch	switches the KB record buffer with the alternate buffer
savekb	saves the KB on diskette in drive B
loadkb	loads the KB from diskette in drive B
getbuf	gets the data in the channel buffer
addbuf	adds the data in the channel buffer
rembuf	removes the data from the channel buffer
nextline	gets the next line of Estelle specification
error	gets Estelle specs line in error
emsg	prints error message on screen
getnext	get next literal from Estelle specification
getid	get next Estelle item as identifier

geteq	get next Estelle item as '='
getcm	get next Estelle item as ','
getsc	get next Estelle item as ';'
getcl	get next Estelle item as ':'
getdt	get next Estelle item as '.'
getlp	get next Estelle item as '('
getrp	get next Estelle item as ')'
getlb	get next Estelle item as '['
getrb	get next Estelle item as ']'

File : ESTL2KB.PAS

General	This file contains the subsystem to convert the Estelle specification into KB.

Procedures

checktypes	checks for types in Estelle specs
checkintype	checks for existing type declarations
estlconst	converts Estelle CONSTANTS declarations
estltype	converts Estelle TYPE declarations
typ-record	converts Estelle RECORD declarations
estlvar	converts Estelle VARIABLES declarations
estlchan	converts Estelle CHANNEL declarations
estltrans	converts Estelle TRANSITION declarations
estlprocode	converts Estelle user defined procedures
filltranstab	creates a transition table used in execution of the prototype
estlmodule	converts MODULE declarations
estlstset	converts Estelle STATESET declarations
estlstate	converts Estelle STATES declarations

File: KBEDIT.PAS

General — This file contain the subsystem to display the KB.

Procedures

disppar	displays parent (backward link) of an entity
dispchild	displays child (forward link) of an entity
getopts	gets user selected option when in KB display screens
nextscreen	displays the next screen of KB
prevscreen	displays the previous screen of KB
editscreen	controls the user input and display of KB screen
constscreen	displays CONSTANTS screens
typescreen	displays TYPE screens
chanscreen	display CHANNEL screens
dispchan	display additional info on channels
modscreen	displays MODULE screens
varscreen	displays VARIABLES screens
transcreen	displays TRANSITION screens
disptrans	displays additional info on transitions
statescreen	displays STATE screens
stsetscreen	displays STATESET screens

File: CRTPRT.PAS

General — This file contains the subsystem to generate the executable prototype.

Procedures

error	displays errors in generating the prototype
nl	writes a line of Pascal code on file

ad	adds a string to Pascal code
cr	adds a character to Pascal code
qt	adds a quote to Pascal code
sc	adds a ';' to Pascal code
bg	adds 'BEGIN' to Pascal code
ed	adds 'END' to Pascal code
sp	adds a space to Pascal code
bl	forces a new line of Pascal code
lp	adds '(' to Pascal code
cm	adds ',' to Pascal code
eq	adds '=' to Pascal code
rp	adds ')' to Pascal code
cl	adds ':' to Pascal code

chnrec	write channel record declarations in Pascal code
mkchrc	makes a channel record declarations
chntyp	adds channel buffer variables type declarations in Pascal code
chnio	generates channel input output code
putspn	put the name of the Service Primitive in the Pascal code

File : EXPRTO.PAS

General	This file contain the main program and various procedures to execute the generated prototype interactively.

Procedures

bell	rings bell
disp-st	displays user state name
spn-in	checks if service primitive is in

chkpro	checks for transition PROVIDED conditions
exectrans	initiates a transition
whichtrans	checks which transition to initiate
upper-layer	process upper layer I/Os
lower-layer	process lower layer I/Os
display-exprt	displays the screen for prototype execution
getinputs	gets user inputs during prototype execution
setpf	sets cursor position according to the PF keys entered by user
exprt-interactive	handles prototype execution in the interactive mode
exprt-ttcn	executes prototype using TTCN
init-ttcn	initializes TTCN file setup
send	if a TTCN send is completed
upper	if the TTCN line was for the upper layer
nextlvl	if the next TTCN line is at different level
stepexecuted	if the TTCN step is executed
stepfailed	if the TTCN step has failed
testfailed	if the TTCN test file
perform-tests	performs TTCN tests
matched	if the results matched with expected TTCN results
comp-ttcn	compiles TTCN file data
getdn	gets a D number in TTCN
gettn	gets a T number in TTCN

File : CRTPRT1.PAS

General

This file is generated by the CRTPRT subsystem
It contains the constants, types , variables declarations for a specific protocol.
The file also contain the following procedure

Procedures

chanelio	performs protocol specific channel inputs/outputs

File: CRTPRT.PAS

General	This file contains the general procedures to perform channel buffer input and output

Procedures

loadstoresp	loads or stores a data on channel buffer
out	outputs a request

File: EXPRT3.PAS

General	This file is generated by the 'Create Prototype ' Subsystem.

Procedures

chkpro	contains the necessary code to check for the PROVIDED conditions
exectran	contains the necessary code to execute a transition procedure.

C-2 :HOW TO COMPILE FPCP SYSTEM

The following steps assume that a working copy of CP/M 86 operating system has been created. This working copy also includes the TURBO PASCAL Compiler.

1. Turn on the computer.

2. Insert the CP/M operating system diskette in drive A.

3. Enter A to start the system from drive A.
   The following message would appear:

   'CP/M-86/80 Version 2.00(1.1)

4. Enter TURBO to load the Pascal compiler.
   The following screen would appear:

```
TURBO pascal system     Version 3.00A
                        CP/M-80, Z80

Copyright (C) 1985      BORLAND INC

Include error messages(Y/N)?
```

5. Enter Y to include error messages. The following screen should appear:

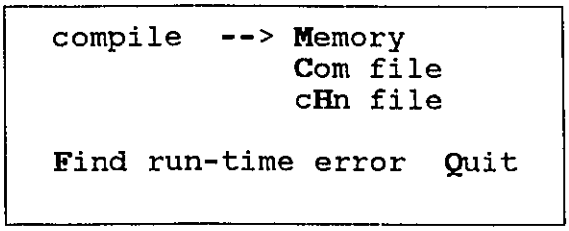
```
Logged drive :A
Work file:
Main file:

Edit Compile Run Save
eXecute Dir Quit compiler Options
```

6. Insert FPCP system diskette in drive B.

7. Enter L to change the logged drive to B.

8. Enter M followed by the program name (FPCP) to load the main program.

9. Enter O to select compiler options.
   The following sub-menu would appear:



10. Enter C to select COM file option.

11. Enter Q to exit the compiler option menu.

12. Enter C on TURBO Pascal main menu to compile.

C-3: HOW TO START THE FPCP SYSTEM

1. Turn on the computer.

2. Insert the CP/M operating system diskette in drive A.

3. Enter A to start the system from drive A.

4. Enter MDRIVE 2 to allocate the memory drive (virtual disk)
   The following message would appear:

   'M: drive installed. Available file space = 126K'

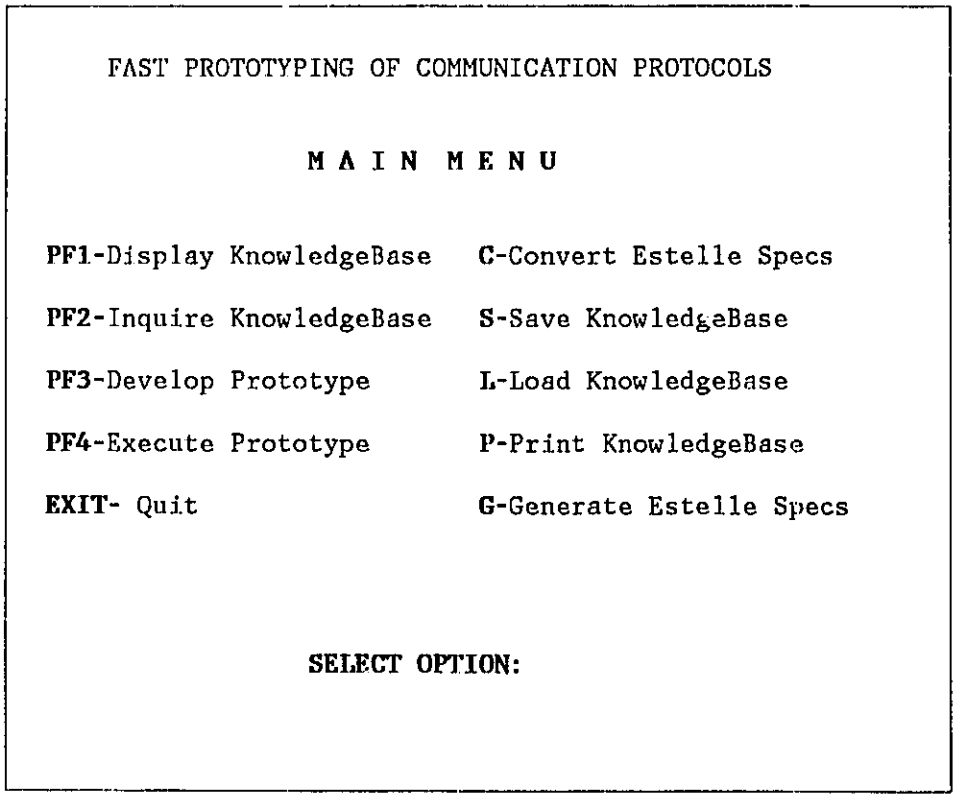
5. Enter B: to select the drive B as default.

6. Enter FPCP to start the system.

C-4 : SYSTEM FEATURES AND LIMITATIONS

THE FPCP SYSTEM MAIN MENU

The system main menu displays the functions available to the users. The FPCP system Main Menu screen layout is shown below.



These functions are described in more detail in the following sections. Please note that a user can return to this Main Menu by pressing the MAIN MENU key from any other screen in the system.

Option C - Convert Estelle Specs

This option allows users to convert the given Estelle specification into a KnowledgeBase (KB). The KB is maintained in main memory as virtual disk records.

A file named 'ESTL.PAS' must be supplied containing the protocol specification in Estelle. (ESTL.PAS is a sequential text file. Any Editor capable of creating/editing sequential text files can be used to maintain this file. In this study the TURBO Pascal Editor was used to maintain this file).

If an Estelle Syntax error is detected the error is displayed and the conversion is terminated. There are some limitations and extensions to Estelle syntax. They are described in Chapter 2. Upon completion of this task the system returns to the Main Menu.

This step is required at least once, and every time the Estelle source specification is changed. Once the specification has been converted into KB , it can be saved in KB form (see option S below), thus avoiding the need to re-convert the same specification every time the system is started.

Option S - Save KnowledgeBase

This option is used to save the contents of KB on diskette. The KB is saved on a file named "KB.PAS'. The FPCP working diskette must be in drive B.

This step is necessary when a new version of the Estelle specification is converted into KB during the current session.

If no conversion was done during this session and the contents of the KB did not change during the current session then this step is not necessary.

A user can rename this file (using the CP/M File Maintenance functions) to maintain different versions of the KB.

Option L - Load KnowledgeBase

This option is used to load an existing KB (File named 'KB.PAS') from diskette in drive B.
The current version of FPCP only loads the KB file named 'KB.PAS'. This function is provided as a main menu item to allow for future enhancements to the system to permit users named files to be loaded as KB. Currently, to load a different version of a KB the required version has to be renamed to KB.PAS prior to starting the FPCP system.

Option P - Print KnowledgeBase

This option allows users to create a print file of the KB in memory. The function is available as FPCP system debugging tool. The print file created is named KBPRT.PAS. This print file can be printed by using the CPM PIP command:

```
PIP lst:=b:kbprt.pas
```

Option G - Generate Estelle Specification

This option is not available in the current version of the FPCP system. It is intended to convert the KB back into the Estelle specification file.

Option PF1 - Display KnowledgeBase

This option lets the user browse through the KB. The system displays a series of screens containing the information in the KB. The top two lines of each screen are:

```
OPTION: 1=Const 2=Type 3=Channel 4=Module 5=State 6=Stateset
        7=Variables 8=Transitions 9=Procedures
```

Each option can be selected for display by pressing the 'ADDITIONAL OPTIONS' key followed by the option number (1 to 9).
The screen containing the information on the module constants declarations is presented first. Then a user can select any one of the nine screens as mentioned above.

From each screen a user can select next or previous screen by pressing 'NEXT SCREEN' or 'PREV SCREEN' keys respectively. Where applicable, the PgUp and the PgDn keys can be used to browse through the previous and next page of a screen. The Up and Down arrow keys are used to move the cursor up and down to select an item on the screen.

The format of each screen and the use of some function keys in each screen is different from screen to screen. A detailed description of each screen and the applicable function keys is given on the following pages.

The layout of each screen, its description, and the use of applicable keys are given on the following pages. The following screen samples contain examples derived from the Estelle specification of transport(0) protocol as listed in Appendix A.

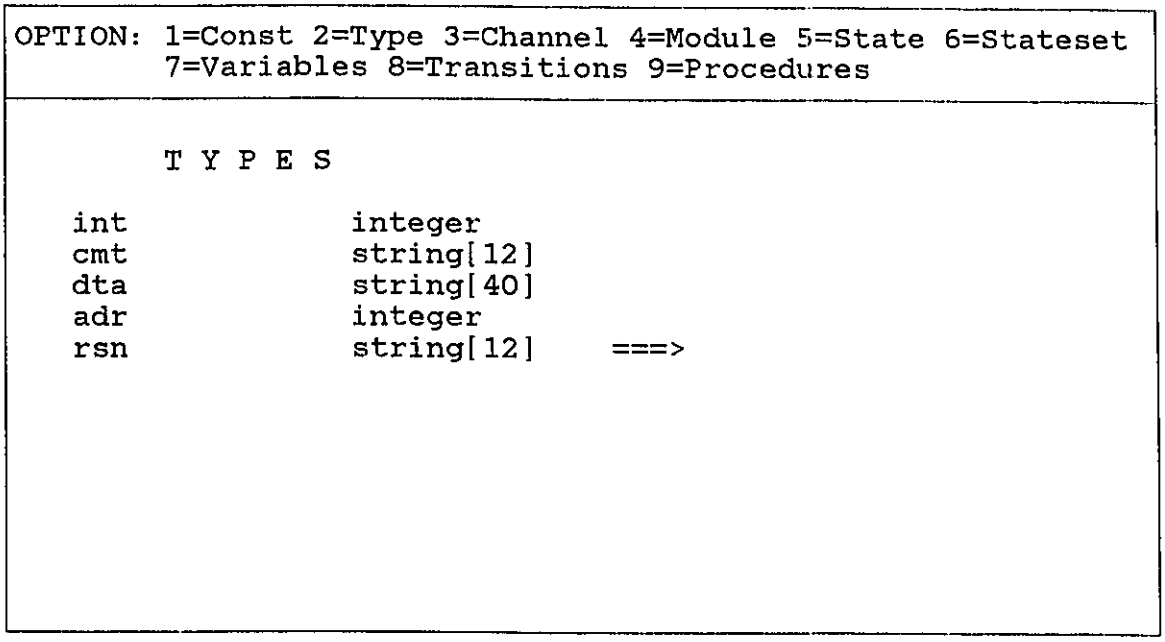
SCREEN NAME: CONSTANTS

```
OPTION: 1=Const 2=Type 3=Channel 4=Module 5=State 6=Stateset
        7=Variables 8=Transitions 9=Procedures

        C O N S T A N T S

   null              ''
   zero              0
   ok-qts            5      ===>
   tpdusize          10
   ok                'ok'
   ok-option         'normal'
   qtsest            5
```

Description: This screen displays the protocol entity constants and their values. This is the first screen displayed when the 'Display KnowledgeBase' function is selected from the system Main Menu. In addition to the ADDITIONAL OPTIONS key the following keys are active:

PREV SCREEN - return to system Main Menu
NEXT SCREEN - display TYPE screen
PgUp - display previous page of Constants
PgDn - display next page of Constants

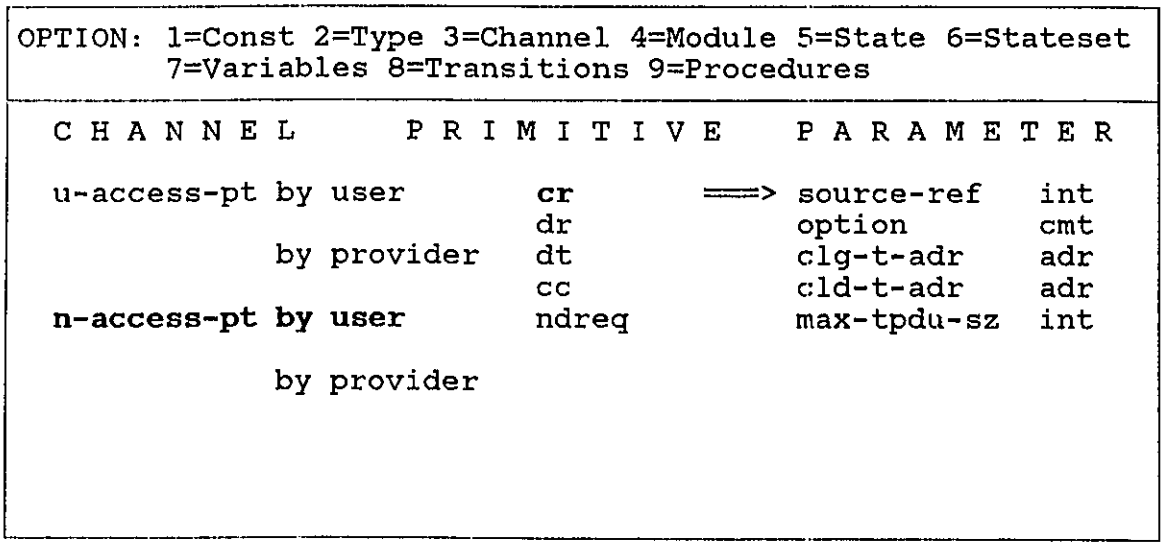
SCREEN NAME: TYPE



Description: This screen displays the module entity TYPEs. The following keys can be used on this screen:

PREV SCREEN - display CONSTANT screen
NEXT SCREEN - display CHANNEL screen
PgUP - displays previous page of Variables
PgDn - displays next page of Variables
-> - (if the cursor is highlighted) displays additional information on the selected variable such as Range, Record, Array details.

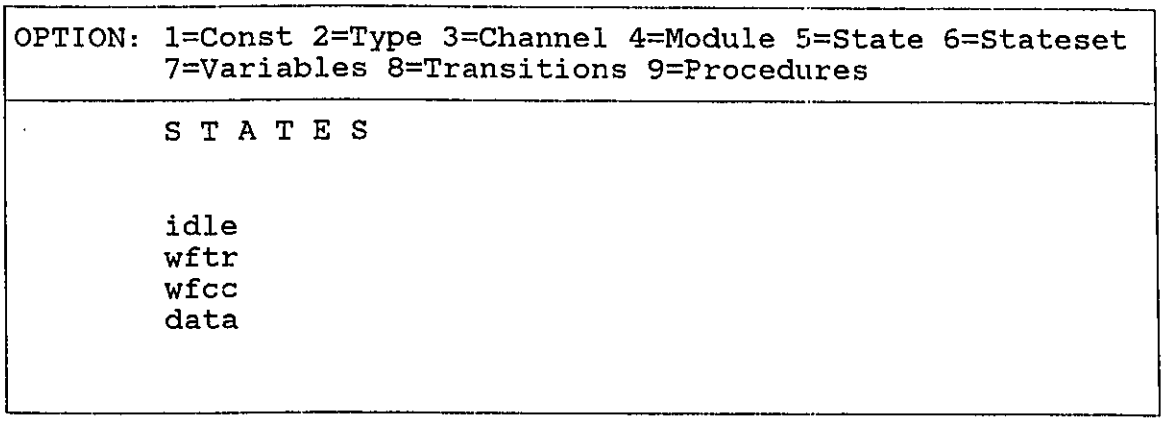
SCREEN NAME: CHANNELS



Description: This screen displays the protocol channel information. Three columns of information is displayed. Initially, only the column 1 is displayed listing the protocol interaction points. At this stage the user can move the cursor up and down by the Up and Down arrow keys. The interaction point next to the cursor is highlighted. By pressing the SELECT (or ->) key a user can obtain a list of Service Primitives (in column 2) applicable to the highlighted interaction point. Again as column 2 is displayed the user can move the cursor up and down by using the Up and Down keys to select a Primitive. By pressing SELECT at this stage system displays (in column 3) the parameters and their types applicable to the selected Service Primitive.

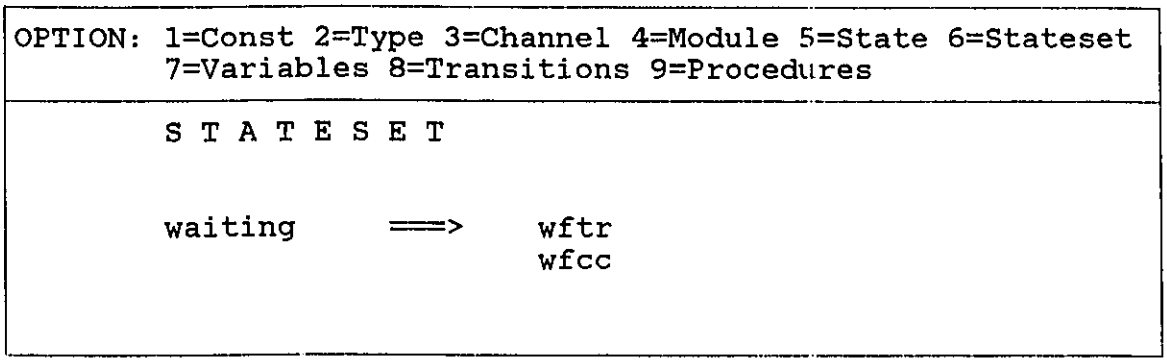
In the above example the parameters listed are that of the Service Primitive cr (Connect Request), and the list of primitives displayed in column are associated with the highlighted interaction point (n-access-pt by user).

SCREEN NAME: STATES



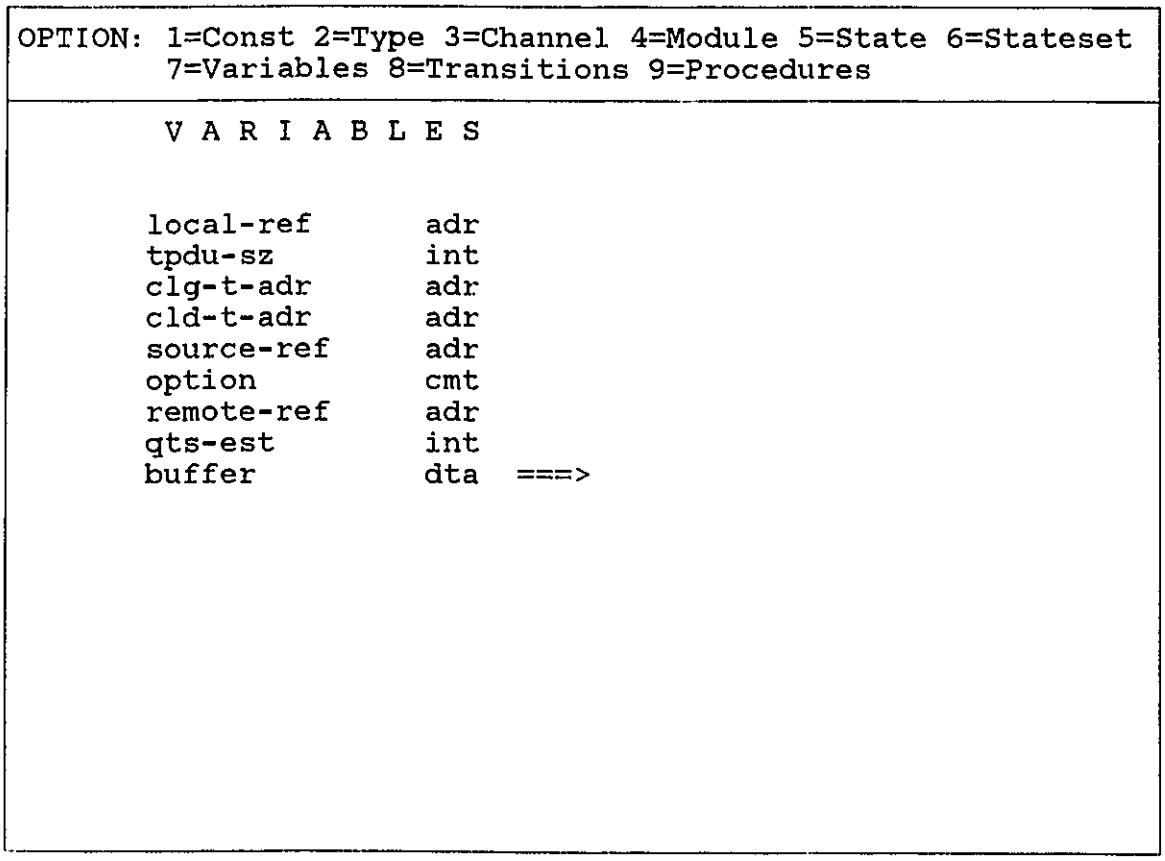
Description: This screen displays the list of module states.

SCREEN NAME: STATESET



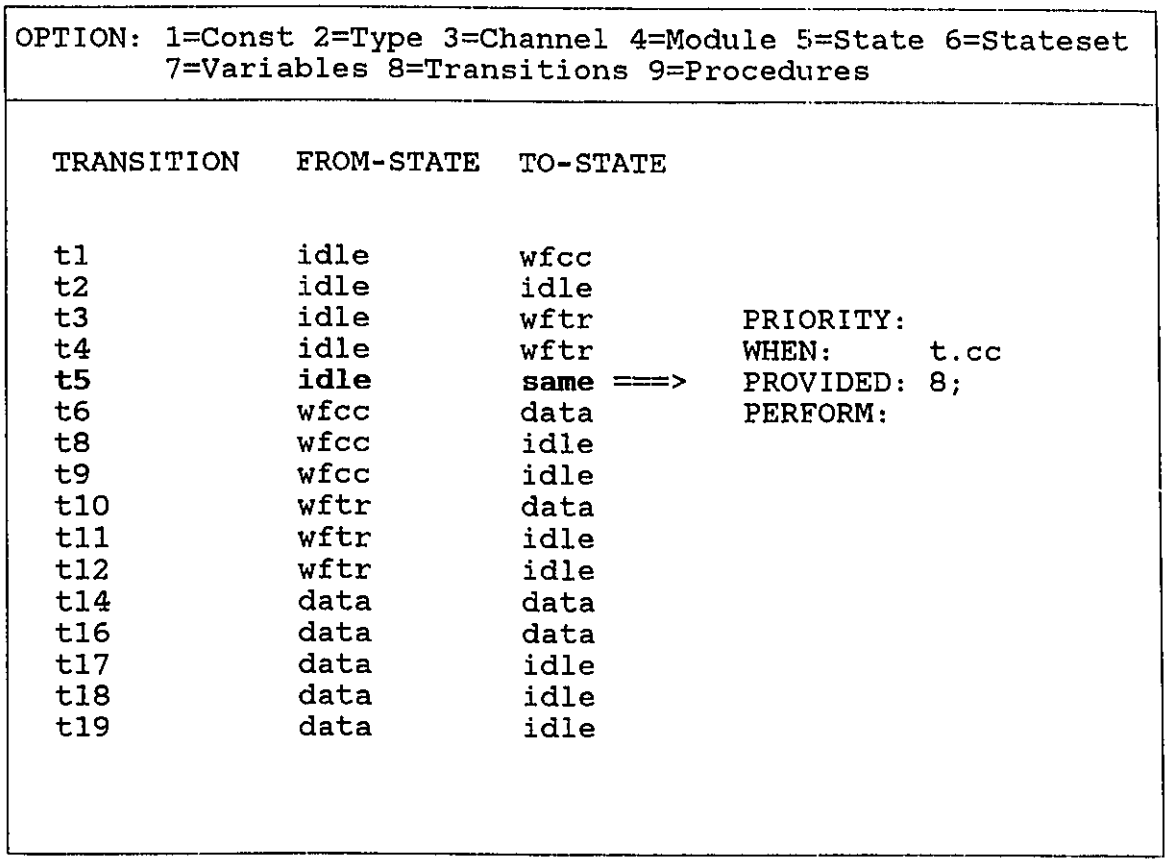
Description: This screen displays the STATESETs defined for a protocol in Estelle specification. Initially, this screen displays the list of Statesets in column 1. A user can select any one of the Statesets by moving the cursor up or down with the Up and Down keys. By pressing SELECT key, a user can obtain the list of states in the selected Stateset.

SCREEN NAME: VARIABLES



Description: This screen displays the module variables information. The initial display lists the Variables and their type. A user can select a variable by moving the cursor up or down with the Up and Down arrow keys. By pressing SELECT key any additional information on the selected Variable (if there is any) is displayed.

SCREEN NAME: TRANSITIONS



Description: This screen displays information on the Transitions of a module. The initial display contains three columns listing the transition id, From-State, and the To-State. A user can press PgUp and PgDn keys to display the complete list of transitions. The cursor can be moved up and down by pressing the Up and Down arrow keys. The SELECT key displays the additional information on the transition such as the transition Priority, WHEN condition, and references to PROVIDED Boolean function and PERFORM procedure.

SCREEN NAME: MODULE

```
OPTION: 1=Const 2=Type 3=Channel 4=Module 5=State 6=Stateset
        7=Variables 8=Transitions 9=Procedures
-------------------------------------------------------------
        M O D U L E

        Module trans-class0

               u : u-access-pt  provider queued

               n : n-access-pt  user not queued
```

Description: This screen displays the protocol Module Header information.

SCREEN NAME: PROCEDURES

```
OPTION: 1=Const 2=Type 3=Channel 4=Module 5=State 6=Stateset
        7=Variables 8=Transitions 9=Procedures
-------------------------------------------------------------
        P R O C E D U R E S
```

Description: The PROCEDURE screen was designed to display the FPCP generated Boolean procedures to handle the PROVIDED conditions, and also to display the user coded procedures to be performed when a transition is selected. However, in the current version of this system this feature has not been implemented.

Option PF2 - Inquire On KnowledgeBase

This option is not available in the current version of the FPCP system. This option combined with a natural language interface is expected to allow users the capability to interrogate the KB. Furthermore when this function is integrated with the executable prototype it could provide better explanations of protocol behavior (from the prototype execution trails).

Option PF3 - Develop Prototype

This option allows user to develop an executable prototype of a protocol from KB. This function when executed generates two Pascal program files EXPRT1.PAS and EXPRT3.PAS containing protocol module constants, variable declarations, channel input output code tailored for the protocol, and the necessary CALLs to user defined procedures.
The user interface to execute the prototype in the interactive mode is part of the common generic functions in the FPCP system.

This option which generates the Pascal code is the first of two steps in generating the executable prototype. The second step is to re-compile the FPCP system. The FPCP main program file contains the necessary include instructions to include the two generated files.
Like all other functions on this menu, this function and the re-compilation can be performed as many times as needed. A user can create different versions of prototypes from different versions of KB.

Option PF4 - Execute Prototype

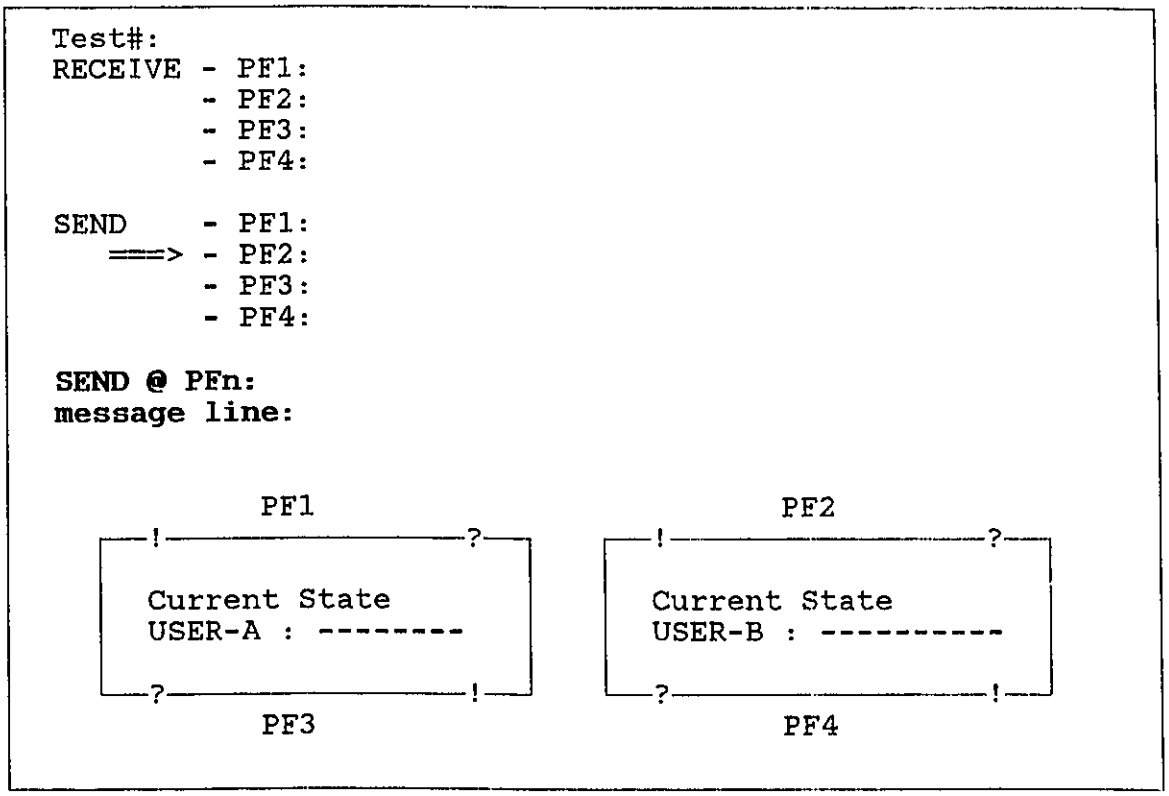
This option is available once a prototype of the protocol has been developed (using the option PF3 as stated above), and the FPCP system re-compiled. This option allows the user to execute the protocol interactively. The user is presented with a choice of three different execution modes(see the screen sample below).

PROTOTYPE EXECUTION - OPTIONS MENU

```
        EXECUTE PROTOTYPE

  I - Interactive Execution
  R - Remote Test Method - TTCN File
  L - Local Test Method  - TTCN File
      Enter Option:
```

For each one of the three prototype execution modes the system displays a screen as shown on the following page. The screen lets users interact with the protocol and displays the protocol behavior.

PROTOTYPE EXECUTION SCREEN



This screen displays the status of two communicating protocol entities. Each box represents a protocol entity. In the center of each box, the current STATE of protocol entity is displayed.

The two entities interaction points and the data channel buffers are annotated by PFi, '!', and '?' indicators.

PF1 is the protocol entity A interaction point with the higher layer. PF3 is the entity A interaction point with the lower layer. Similarly PF2 and PF4 are the protocol entity B interaction points. The '!' and the '?' indicators indicate the protocol entity interaction point channel input and output buffers respectively.

During the execution of the prototype , the current STATE of each entity is displayed in the appropriate box. The names of Service Primitives sent or received are are displayed along with the '!' and '?'. The top portion of the screen shows the actual messages which are sent and received in each buffer.
The 'SEND @ PFn' line is used in the 'INTERACTIVE' mode only to select an interaction point for sending messages. To select an interaction point one of the PF keys 1 to 4 is pressed. To send messages after selecting the interaction point press SEND. A new line will appear for the user to enter the desired message.

A description of each one of the three execution modes is given in the following paragraphs.

Option I - Interactive Execution

Upon selection of this option the Prototype Execution Screen is presented. The initial state of the two entities is displayed in the two entities boxes. The user can select any interaction point by pressing the PF keys (PF1 to PF4). As described above each one of this PF key is associated with one of the four interaction points.
Once an interaction point is selected the user can send a message (Service Primitive) by pressing the 'SEND' key followed by the message on the 'message line' area.

The user input interactions are verified against the service primitives specification in the KB. If the input sequence is incorrect an error message is displayed. These errors are checked by the User Interface.

During the validation of user input on this screen the FPCP system may display one of the following error messages:

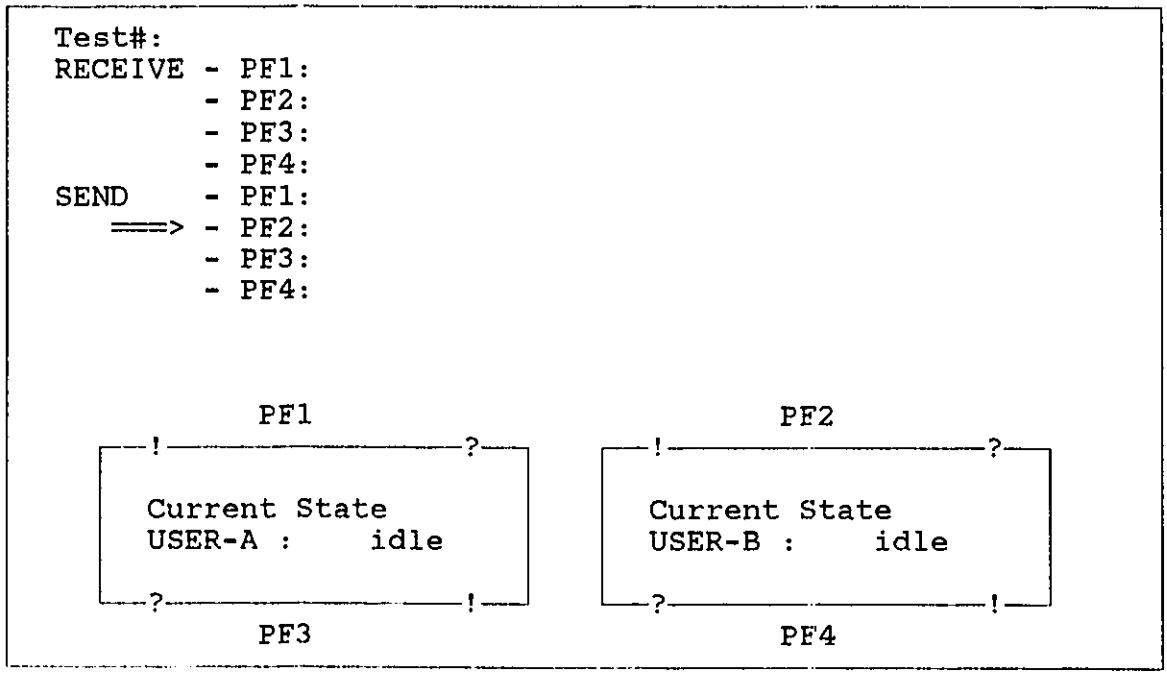
- SP NOT FOUND (Invalid service primitive name);
- SP INVALID (Invalid service primitive at the selected interaction point);
- INVALID @ CURRENT STATE (Invalid service primitive at current module state);
- TOO FEW PARAMETERS;
- TOO MANY PARAMETERS;
- PARAMETER SYNTAX (The syntax of the parameter is not according to the specification in KB.

This feature is used to debug and refine the protocol specification. The input sequences are entered on this screen. The system displays the module entities output sequences and the contents of each channel. The outputs are validated against the specification by the user.
This screen can also be used to validate IUT test results. The tests conducted on IUT can be repeated here and the outputs of the prototype compared to that of the IUT results.

An Example:

This example shows the FPCP Interactive Prototype Execution using the the transport(0) FPCP Prototype. Please refer to the protocol specification in Appendix A.

The example contains three input sequences and the prototype outputs after each input sequence. Upon selection of option "I - Interactive Execution" the Prototype Execution Screen" is displayed. The current state of each entity is displayed as "idle".

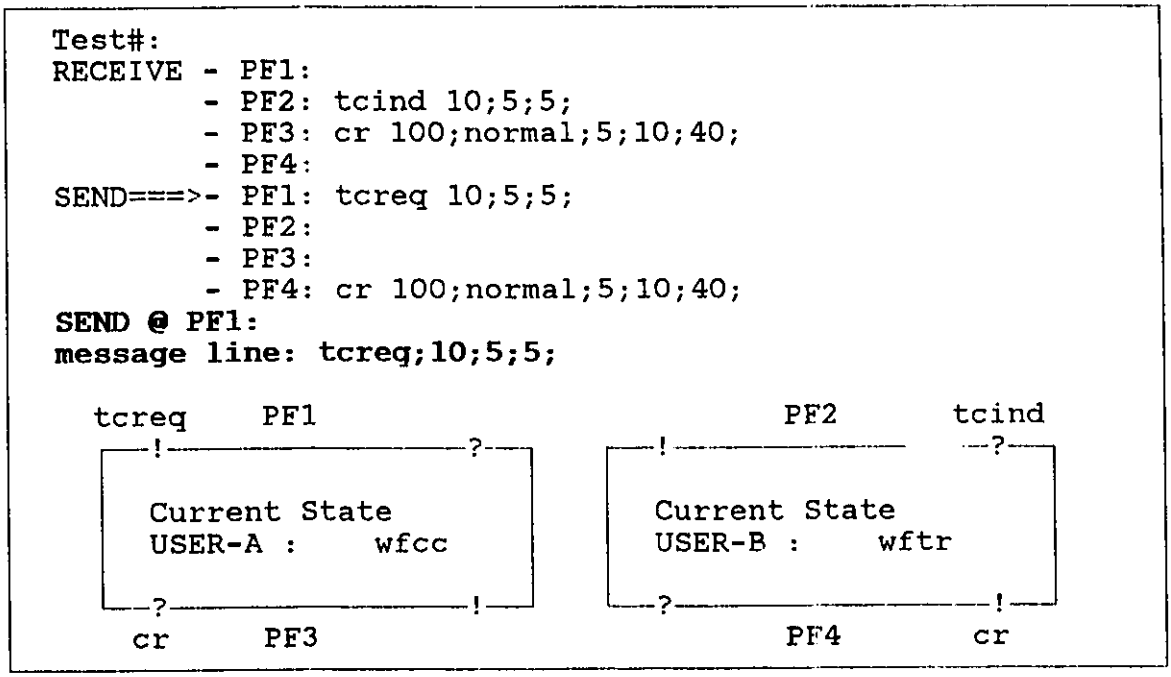


Prototype Display at Initial Stage

The first input sequence is the connection request from user A(site address 10) to user B (site address 5) with quality of service 5. To send this request from user A to user B the following steps are performed:

1) Enter "PF1" to select the user A interaction point;
   The cursor (===>) moves to SEND PF1 line.
2) Enter "SELECT" to display the message line;
   A message line appears for user to enter the input sequence.
3) Enter the input sequence. (tcreq;10;5;5;).
   The input sequence consists of the service primitive name followed by the parameters. Each item in the input sequence is separated by a semicolon (;).

Upon receiving the input sequence the prototype will execute the sequence and display the outputs as shown below.



Prototype Display after Sending tcreq at User A Upper Level

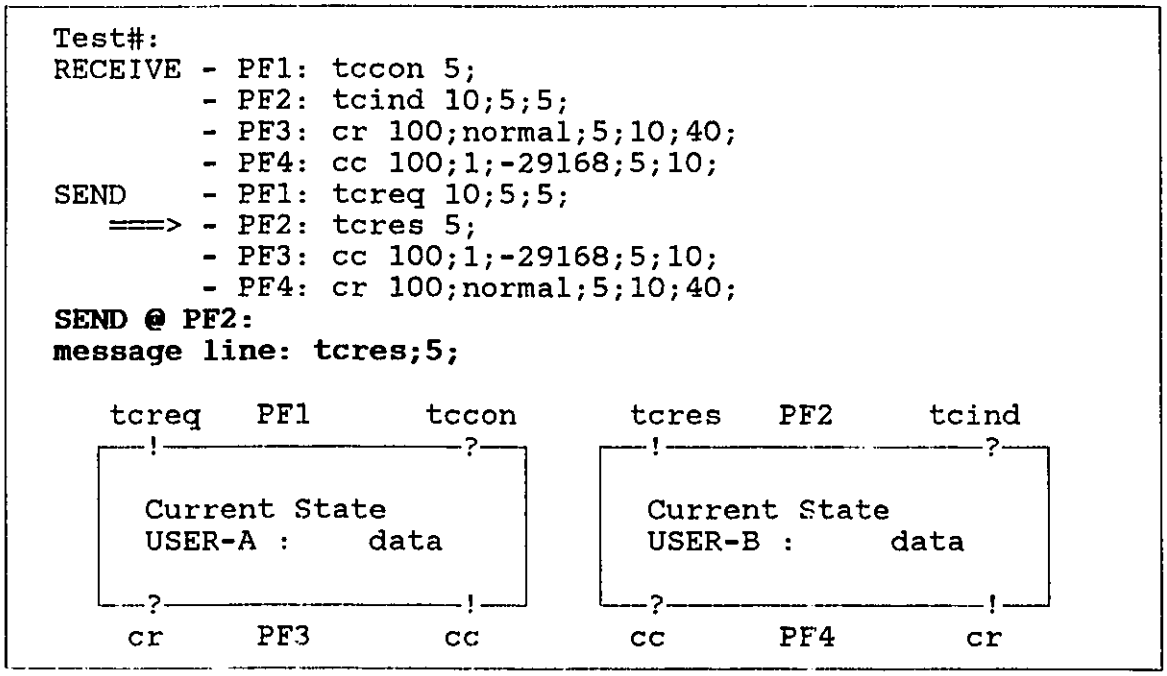
The second input sequence is the connection response from user B indicating the quality of service 5. The following steps are performed to send this sequence:

1) Enter"PF2" to select user B interaction point.

2) Enter "SELECT" to display the message line.

3) Enter the input sequence (tcres;5;).

In response to this sequence the display changes as follows:



Prototype Display after Sending tcres at User B Upper Level

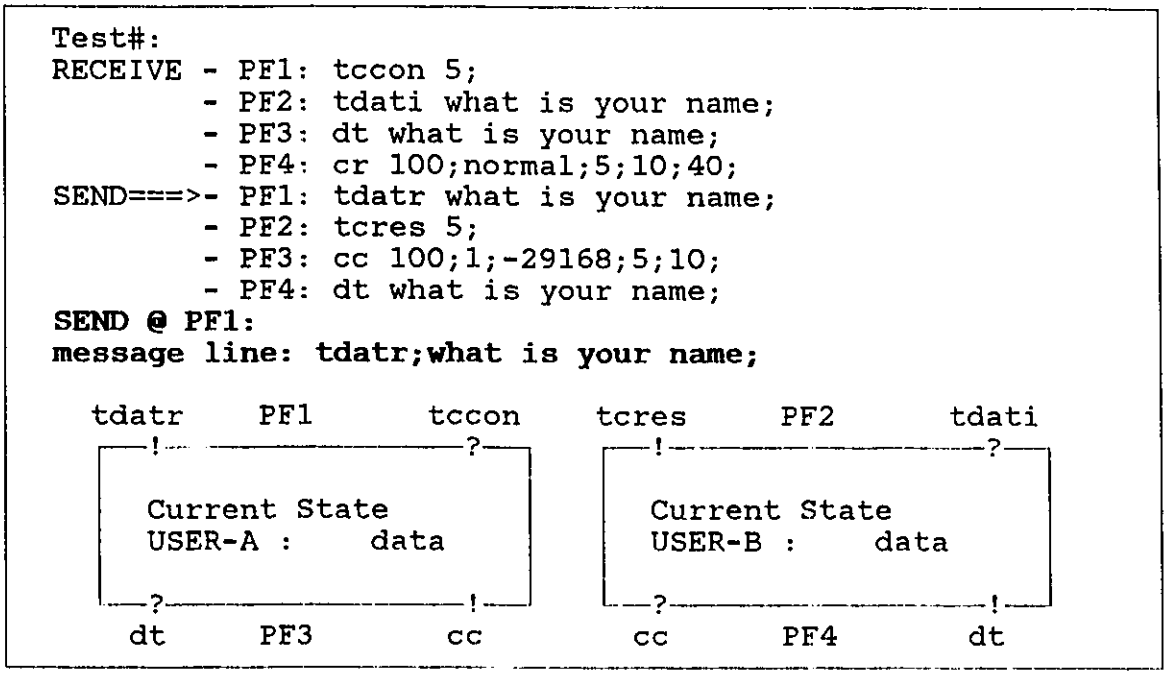
The third sequence is the data ("what is your name" exchange from user A to B. The following steps are performed to process this sequence:

1) Enter "PF1" to select user A interaction point.

2) Enter "SELECT" to display the message line.

3) enter the input sequence (tdatr;what is your name;).

This input sequence results in the changes to display as follows:



Prototype Display after Sending tdatr at User A Upper Level

Option R - Remote Test Method - TTCN File

The Remote Test method allows the user to conduct testing of the prototype using protocol test suites coded in TTCN format. This option expects a disk file in drive B named 'TTCN.PAS' containing the test suite.

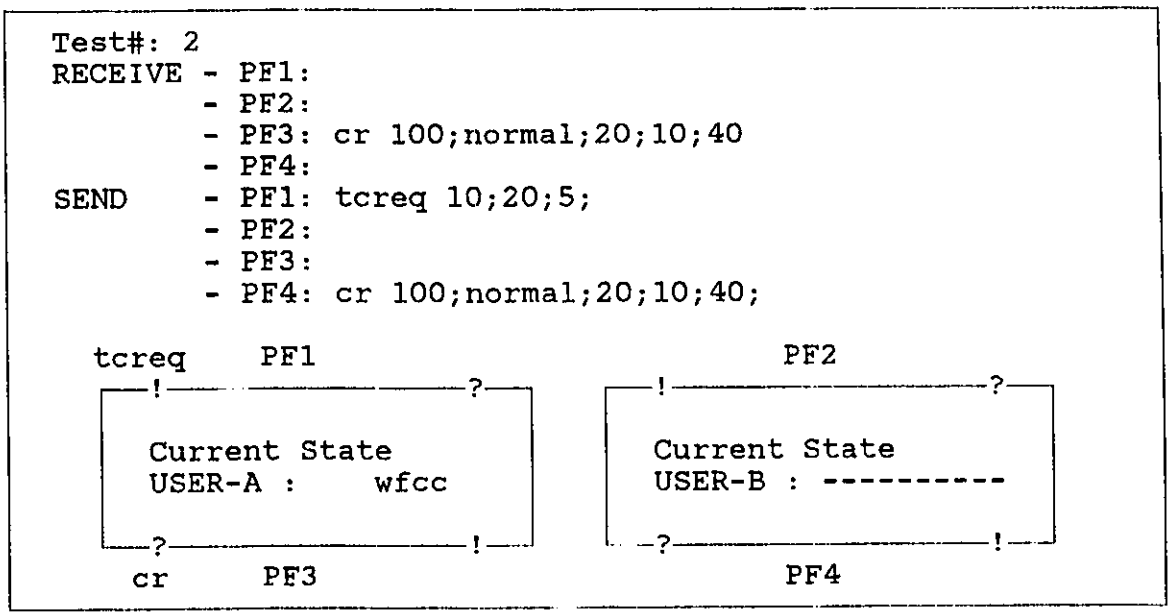
Under this mode the prototype gets its inputs from the TTCN file.
At each step of testing, a complete picture of the protocol entities states and contents of the buffers are flashed on the screen. The prototype output is compared with the expected results specified in TTCN file. The results of this comparison is displayed as each test sequence is executed. At the end of each test the system pauses for the user to press any key to continue testing until all TTCN file test inputs are processed. Please refer to the example below.

An Example: Remote Test Method
This example shows prototype testing with test suite coded in TTCN format. In this example the sample of transport(0) test suite (on file TTCN.PAS) in Appendix D is used. The FPCP prototype outputs for the following four steps is illustrated.

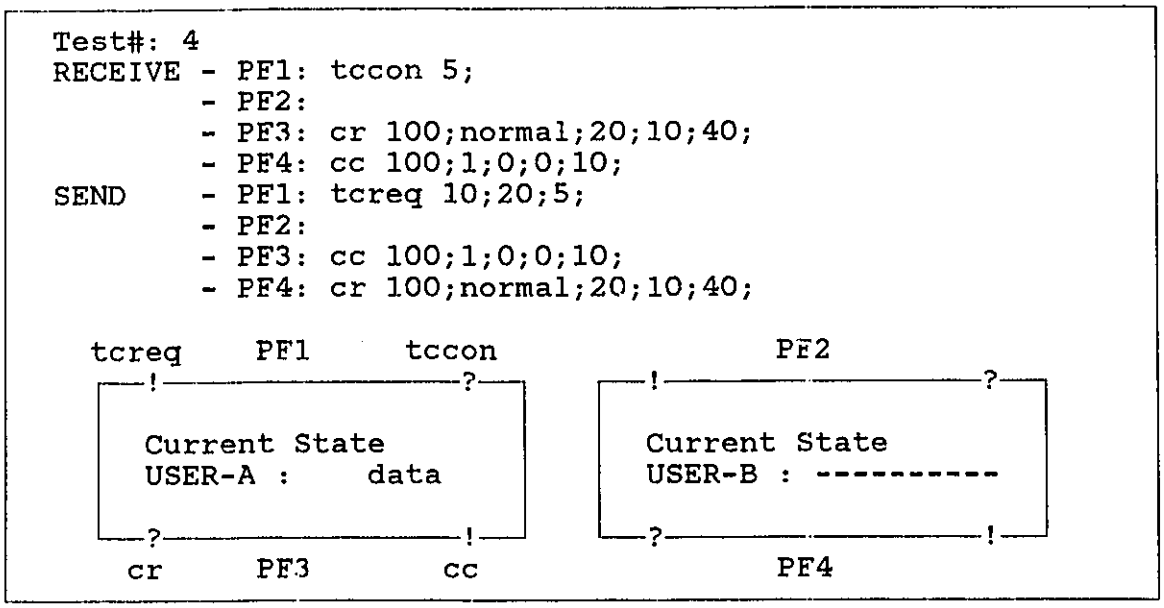
```
Step 1: U! tcreq d1 pass  (d1=10;20;5;).
        (Send tcreq at Upper Level with data d1).

Step 2: L? cr d2 pass (d2=100;normal;20;10;40;).
        (Receive cr at Lower Level with data d2).

Step 3: L! cc d3 (d3=100;1;0;0;10;).
        (Send cc at Lower Level with data d3).

Step 4: U? tccon d4 pass   (d4=5;).
        (Receive tccon at Upper Level with data d4).
```

For each test step which requires a "SEND" the FPCP User Interface prepares a tpdu (transport protocol data unit) from the given test sequence. The Interface also performs the necessary validation of the input sequences against the specification in the KB. The User Interface then sends this tpdu at an appropriate interaction point as specified by the test case. The Protocol Execution Control displays the protocol execution as shown with this example.

For each test step which requires a "RECEIVE" the FPCP User Interface compares the contents of the specified interaction point with the expected output data specified in the test case. This step does not result in any changes to the Prototype Display.



Prototype Display after Remote Test TTCN Steps 1 and 2



Prototype Display after Remote Test TTCN Steps 3 and 4

Option L - Local Test Method - TTCN File

This mode is similar to the "Remote Test Method" discussed above. Under the "Local Test Method" the prototype is executed as single (local) entity as oppose to two remote peer entities as in the Remote Test Method.

C-5 : SAMPLE FPCP SESSION

This section presents examples of FPCP session with transport(0) protocol Normal Form Estelle specification.

PREPARING A COPY OF FPCP DISKETTE

Make a separate copy of FPCP system diskette for each protocol. This facilitates maintenance of protocol specification file, test files, and the executable prototype files all on one diskette. Follow the steps below to make a copy of FPCP system diskette.

1. Insert the CPM Operating diskette in drive A.
2. Start the system from drive A.
3. Prepare a blank formatted diskette (Refer to CPM User Manual for procedures to format diskettes).
4. Enter DISKCOPY to start the CPM copy program. Follow the system prompts to make the copy.

Keep the original at safe place. Use the copy as FPCP system diskette for all examples below. Label the diskette as FPCP Transport(0).

TRANSPORT(0) NORMAL FORM ESTELLE SPECIFICATION

Develop the protocol specification in Estelle. Make sure the specification are in Normal Form (Single Module). If necessary perform the necessary transformations to convert the specification into Normal Form.

When developing the Estelle specification make sure the identifiers (variable names, interaction points , etc) are not larger than 12 characters.

In Estelle transition part include a transition identifier after the keyword "trans". In the transition processing code make sure the coding is valid Pascal statements. Use the build-in procedure OUT(spname) to output a service primitive.

Please refer to example of transport(0) Estelle specification in Appendix A.

CREATING ESTELLE SPECIFICATION FILE

Create a sequential text file containing the protocol Estelle specification. Make sure to name the file "estl.pas". The file should be created (or copied onto) the protocol FPCP diskette.

Note: Any text Editor which can create sequential text files can be used to create this file. The file in appendix A was created by using the Turbo Pascal Program File Editor.

The current version of FPCP system works with only one specification file (estl.pas). You may create different versions of the protocol specification using different file names. For processing the desired version of the specification make sure to name the file "estl.pas".

STARTING THE FPCP SYSTEM

Start the FPCP system with the protocol FPCP diskette. Follow the procedures in Appendix C-3. The system main menu with all the system options would appear.

When starting the system for the first time (or before creating your own protocol specific KnowledgeBase) the only option which is applicable is "Option C - Convert Estelle Specs".
The other options if selected may process the KB which was copied from original FPCP diskette.

CONVERTING ESTELLE SPECIFICATION ONTO KB

Make sure the Estelle Specification file is named "estl.pas" and is on the protocol FPCP diskette. Select Option C to start the conversion of Estelle into KB. The system displays Estelle specification as they are converted. On any conversion error the error message is displayed and conversion process terminates. If a conversion error occurs correct the error on

the specification file and restart the system and try the conversion again. Repeat these steps until all specification syntax errors are eliminated. This conversion step is required every time the Estelle specification are changed.

Once an error free conversion is completed then use the other Main Menu options as required. You can use the Option "Save KnowledgeBase" to save the KB on disk. This avoids the need to convert the Estelle file every time the FPCP system is started.

SAVING KNOWLEDGEBASE(KB) ON DISK

Use the option "S - Save KnowledgeBase" to save the KB on disk. The KB is saved on disk in drive B as a file named "kb.pas". A KB once saved can be reloaded in subsequent FPCP sessions by the main menu option "Load KnowledgeBase".

The current version of the system works with only one version of the KB file (kb.pas). Like Estelle specification you can maintain different versions of KB by copying the KB file (kb.pas) with different file names. When required to process a desired version of the KB rename the desired version as "kb.pas".

LOADING KNOWLEDGEBASE (KB) FROM DISK

Use to main menu option "L - Load KnowledgeBase" to load a KB from disk into FPCP system.

PRINTING CONTENTS OF KNOWLEDGEBASE (KB)

Use the main menu option "P - Print KnowledgeBase" to create a print file. The print file is named "kbprt.pas". Use the CP/M PIP commands to print this print file on printer.

BROWSING THROUGH THE KB

Use the main menu option "PF1 - Display KnowledgeBase" to browse through the contents of the KB. A complete description of all inquiry screens is given in Appendix C-4

DEVELOPING EXECUTABLE PROTOTYPES

Use the main menu option "PF3 - Create Prototype" as the first step in creating an executable prototype from the KB. This step creates two files ("exprt1" and "exprt3") on disk in drive B. Make sure your protocol FPCP disk is in drive B. These two files contain protocol specific processing code. Once the processing of this option is completed the system returns to the main menu. To perform the second step in creating the executable prototype you must exit from the FPCP system. Enter EXIT key to exit from FPCP.

The second step requires the recompilation of the FPCP system with the two generated files included. The steps required for this task are documented in Appendix C-2. In summary it requires: - starting the Turbo Pascal from drive A;

- loading the FPCP main program (fpcp.pas);
- compiling with compiler option for a .COM file.

The compiler instructions to include the two generated files are in the FPCP main program file (fpcp.pas).

The FPCP system with the generated code included is expected to compile error free. However the Estelle "transition" processing code part of the specification are copied as is on the generated portion. This may result in some compilation errors. To correct this situation do not change the generated code, make the necessary changes in the Estelle specification file. Reprocess the updated Estelle file through the FPCP system. Perform these iterations until the FPCP system compiles error free.

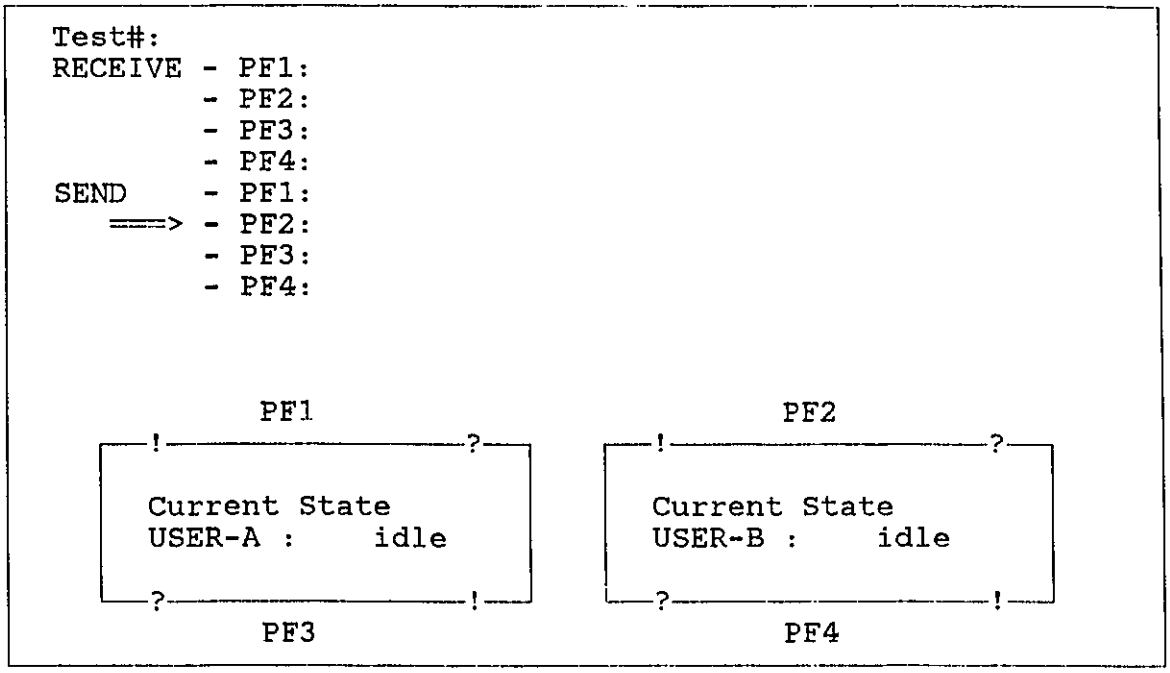
EXECUTING FPCP PROTOTYPES - GENERAL

Use the main menu option "PF4 - Execute Prototype" to execute the prototype (of the transport(0) protocol). Upon selection of this option the system displays the Prototype Execution Options menu. This options menu allows for selection of one of the three different execution modes.

PROTOTYPE EXECUTION - OPTIONS MENU

```
        EXECUTE PROTOTYPE

   I - Interactive Execution
   R - Remote Test Method - TTCN File
   L - Local Test Method  - TTCN File

        Enter Option:
```

For each one of the three prototype execution modes the system displays a screen as shown below. The screen lets users interact with the protocol and displays the protocol behavior.



Section C-4 contains detailed examples of FPCP generated Transport(0) prototype. In Appendix D , along with the FPCP program listing, we included the listing of the two generated files (EXPRT1 and EXPRT3).