A GRAPHICAL REPRESENTATION AND PROTOTYPE EDITOR FOR THE FORMAL DESCRIPTION TECHNIQUE ESTELLE *

Justin Templemore-Finlayson
Jean-luc Raffy, Pieter Kritzinger and Stan Budkowski

Data Network Architectures Laboratory
University of Cape Town
South Africa
Software-Networks Department
Institut National des Télécommunications
France

jtemplem@cs.uct.ac.za

Abstract:

This paper presents a graphical representation for the Formal Description Technique Estelle, which has up until now been restricted to a textual syntax based on the general programming language Pascal. The graphical syntax and semantics of the representation are fully descriptive of a general Estelle system and complies with the ISO Estelle standard [ISO97].

This representation aims to become a standard technique for formally specifying and simultaneously documenting concurrent, communicating systems. Conventions introduced in existing graphical Estelle tools are re-used, as are graphical concepts which have become familiar through widespread use of the similar Specification and Description Language (SDL).

In addition, we present a syntax-directed editor based on the graphical syntax. The editor enables practical application of the graphical technique, by translating a graphical description to the equivalent textual form which can be input into existing Estelle tools. The reverse function makes it possible to view and navigate existing textual Estelle specifications more effectively.

*This project is partially funded by CNet France, the French Government and the South African Foundation for Research and Development (FRD).

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35394-4_29

S. Budkowski et al. (eds.), *Formal Description Techniques and Protocol Specification, Testing and Verification*

1.1 INTRODUCTION

Estelle is a well-defined Formal Description Technique (FDT) used for defining and analyzing concurrent, communicating systems. It shares this function with a number of other techniques, including the Specification and Description Language [ITU89], LOTOS [BB87] and Petri-nets [Pet62].

Estelle was specifically developed by the International Standards Organisation (ISO) for the purpose of describing the protocols and services of the Open Systems Intercommunication Model [ISO94]. It is also an implementation-oriented language based on Pascal [ISO83] which can be used as a basis for reliable protocol and service implementation. Estelle is standardized by the ISO and the latest standard ([ISO97]) was produced last year.

Estelle is however considered to be lagging behind other Formal Description Techniques such as the SDL, due to its lack of a standard graphical representation. SDL which is very similar to Estelle experiences much greater popularity, and one reason for this is considered to be the existence of a standard graphical notation and support tools for SDL.

A graphical technique is usually preferable to a textual one due to its intuitive and pictorial nature. A graphical representation is able to better structure conceptual ideas than pure text, and is closer to the abstract meaning of that which is being described.

In fact, graphical techniques have been in use since long before the development of early digital computers, whose limitations of graphical hardware and software forced the development of textual techniques. Modern graphical interfaces have removed this limitation, and graphical techniques have once again become popular.

A graphical technique can be used in two ways in the context of concurrent, communicating systems:

- to statically *define* the unique structure specifying the functional behaviour of a system, or to

- dynamically *describe* (as opposed to "define") the non-deterministic behaviour of the system.

Techniques which fall into the first class have a formal syntax and semantics, and can be used for formally defining systems. This is not true of methods in the second class, which can only describe behaviour instances of a system, i.e. the results of a static definition.

In this paper we introduce a standard, *definitive* graphical technique for Estelle which is currently being developed. We refer to this notation as the **Estelle/GR** (graphical representation), as opposed to the **Estelle/PR** (phrasal representation as defined in the ISO Estelle standard. The Estelle/GR has a formally-defined syntax and semantics, and is compliant with the ISO Estelle/PR definition.

In addition, the second part of this paper presents a prototype editor based on the Estelle/GR.

The editor makes use of the features of modern graphical interfaces, such as multiple, linked windows and virtual scrolling canvases to provide powerful manipulation of large graphical documents.

The editor combines the utility of the graphical technique with the power of existing Estelle/PR software tools, by providing translation of the graphical document into equivalent Estelle/PR text. The reverse function furthermore allows the graphical technique to be applied to existing Estelle/PR specification documents.

1.1.1 Existing Estelle visualisations

Only two tools exist which attempt to implement graphical techniques for Estelle. Neither of these tools attempts to produce a standardized notation, and neither provides the ability to graphically define a general Estelle specification.

P. Amer *et al* present **GROPE** in [NP93, NP90]. GROPE simulates the execution of a general Estelle specification graphically. It which uses a *descriptive* notation to present the dynamic state of an Estelle system and animates changes in this state. It visualizes the structure of the specification using the common SDL-like nested box notation, and describes each internal module behavior separately in the form of a simple finite state automaton. We re-use some of the abstraction concepts of the GROPE automaton description in the Estelle editor. GROPE is not publicly available.

ASA+ is maintained by Verilog, and is an editor for a non-standard variation of Estelle called LSA+. It uses a partially graphical notation which defines the internal structure of an individual module using a nested box notation, and specifies hierarchical module relationships in a tree form, but specifies internal module behavior using a modified Estelle/PR syntax. The usefulness of ASA+ is limited by the its non-standard extensions and restrictions, and it is not widely used.

1.1.2 Paper outline

Section 1.2 of this paper is a brief introduction to the Estelle model, which provides a starting point for describing graphical representation and structure of the editor. **Sections 1.3, 1.4 and 1.5** introduce the main points Estelle/GR syntax using the Trivial File Transfer Protocol to provide examples. **Section 1.6** discusses the main functions of the prototype editor, and includes some screenshots showing it in use. **Section 1.7** concludes.

Space considerations unfortunately prevent the formal definition of the full Estelle/GR syntax and semantics. Interested readers however are welcome to contact the primary author for further information.

1.2 THE ESTELLE MODEL

An Estelle specification describes a hierarchically structured system of non-deterministic sequential components exchanging messages through bidirectional links between their ports.

Each component is an *instance* of a *generic* module definition. A generic module statically defines a *header* and several associated *body* parts and an instance is dynamically created from a given header-body pair of a generic module definition.

The header part defines the external communication interfaces of the module and specifies the parallel or sequential non-deterministic execution of its child modules; the body part defines its internal behaviour and recursively defines generic children modules.

Estelle uses an extended finite state machine (EFSM) paradigm to describe module behaviour. Each module body defines an EFSM which receives inputs and sends outputs via the communication interfaces of the header. Each transition has a set of conditions which must be satisfied before the transition may fire, and a set of actions which are executed atomically on firing.

An Estelle system has a *dynamic structure* which sets it apart from other FDTs such as SDL, which have a *static structure*. An instance can dynamically create and terminate or release children instances and change their inter-communication structure. As an instance can be instantiated with alternative bodies, it is possible to change the entire sub-hierarchy of instances.

The global behaviour of the system is characterized by the interaction of the executing modules, subject to the concurrency imposed on sibling modules by their parent module, and a priority principle that grants parent instances execution priority over its children.

1.3 THE ESTELLE/GR

The Estelle/GR separates the definition of system structure and behavior, as these two parts are syntactically unrelated. Three notations are used, including two alternative representations for a system structure:

1. A **structure tree diagram** defines a static generic module hierarchy in a tree structure. It can define the dynamic structure of any general Estelle system.

2. An **instance block diagram** defines the initial instance hierarchy and communication structure of an Estelle system which has a statically-determinable initial state. It is used to define Estelle systems limited a static structure, and uses a nested box notation similar to that of ASA+ and SDL.

3. A **transition tree** defines a transition in the internal EFSM of a module body (or instance) of the specification structure. A series of transition trees defines a *forest*, which defines the full EFSM. The transition tree is represented using an SDL-like flow diagram notation.

The editor introduces a further informal **simplified automaton notation** which describes the internal behavior of a module in a simpler automaton form.

A *complete* complete Estelle/GR specification consists of a single **structure tree** or **instance block diagram**, and a **transition tree forest** defining the internal behaviour of each module body or instance occurring in the structure diagram.

In the following sections we define each of these syntaxes using the Trivial File Transfer Protocol to provide illustrative examples. The relative merits of the two structure notations are also discussed. We briefly introduce the TFTP protocol used before continuing.

1.3.1 The Trivial File Transfer Protocol

The Trivial File Transfer Protocol (TFTP) is a simple protocol for reading and writing files across a network. It is defined in [Chi94]. Here we define a version of the TFTP in which the Reader and Writer are implemented as separate 'request handlers' within Initiator and Responder objects, and which are created when the User generates either a read (READ_RQST) or write (WRITE_RQST) request, and destroyed immediately on completion. We only describe and use examples of Writer behavior in this paper.

A User acts as both the Client and a Server. The Client sends requests to the Initiator object, and the Server responds to requests from the Responder object. All network communication takes place between the Initiator and Responder. Our implementation assumes an error-free network link, and connects the Initiator directly to the Responder.

The Client initiates a connection by sending a write request (WRITE_RQST) to the Initiator. If the Initiator is not busy with an existing transfer, it creates a Write Handler, and establishes a link between it and the Responder. The Write handler then forwards the write request to the Responder, which if it is not busy and the Server grants permission, creates a Write Handler to process the request.

The Initiator and Responder Handlers communicate directly to exchange data. Synchronized acknowledgment and timeouts are ensure data integrity. The Initiator handler sends DATA packets, which the Responder handler acknowledges with ACK (acknowledgment) packets. Any errors result in disconnection and the sending of an ERROR packet.

On successful completion of the transfer, or receipt of an error packet, the handlers alert their parent Initiator or Responder objects, which destroy the connections and handlers. The Initiator also receives a result code from the Handler, which it returns to the User in a RESPONSE packet.

1.4 STRUCTURAL REPRESENTATIONS

The idea of representing a structured system of communicating components as a set of of nested boxes connected by vectors representing communication links is a familiar and useful one. Unfortunately, this notation assumes a ***static***

system structure, thus it cannot be used to define a structural dynamic system, such as a general ISO Estelle system.

In order to define a structurally dynamic system, we introduce a notation which we call the *structure tree*. This notation is not as informative as the nested box notation as it cannot make any assumptions about the dynamic instance and communication structure. However, as it is based on generic module definition, can fully define all structural aspects of an Estelle system.

Which notation should be used depends on whether the system structure is *dynamic* or *static*. The nested box notation is more expressive than a structure tree, therefore it is recommended where the system is structurally static. At its most general, it can also usefully describe the initial structure of a dynamic system where this initial structure is statically determinable.

1.4.1 Nested boxes notation

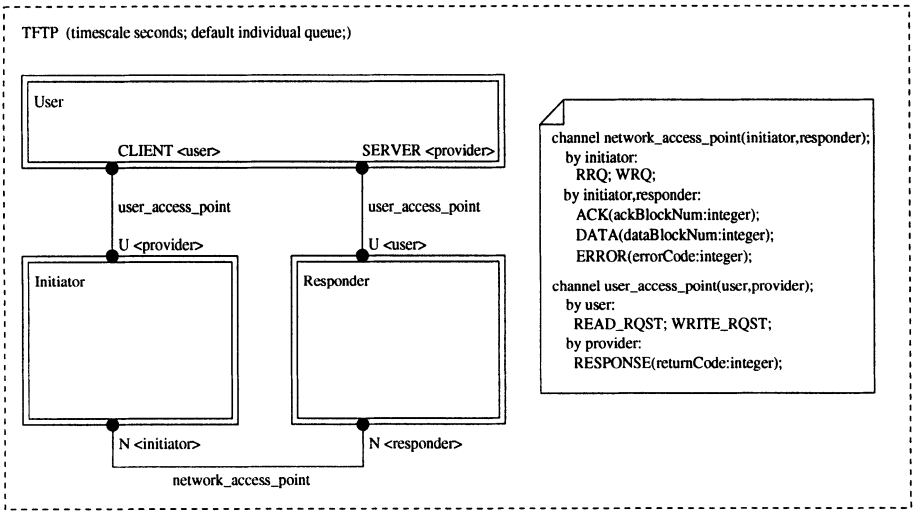


Figure 1.1 The TFTP nested box diagram (showing the initial structure of the dynamic system)

The nested box notation defines system *instances* which effectively have single, statically-determined module header and body parts. For this reason, it cannot define a dynamic system where headers have multiple associated bodies.

The nested box notation thus defines not only the system structure, but also the module body initialization parts required to initialize that structure.

Figure 1.1 shows the nested box representation of the TFTP structure. The nested box can describe the initial structure of this system, as it is statically-determinable. However, it is not *definitive*, as the full definition of the structure tree in Figure 1.3 reveals the definition of descendent modules for both the `Initiator` and `Responder` modules.

In Figure 1.1, each box defines a module instance and the graphical inclusion of one box within another indicates hierarchical substructuring. A box's shape is used to define the mode of concurrency between the included child instances. The diagram shows three systemactivity modules executing asynchronously in parallel, as their parent instance (the specification) is unattributed. The Estelle/GR uses the box shapes shown in Figure 1.2 to define concurrency between children instances.

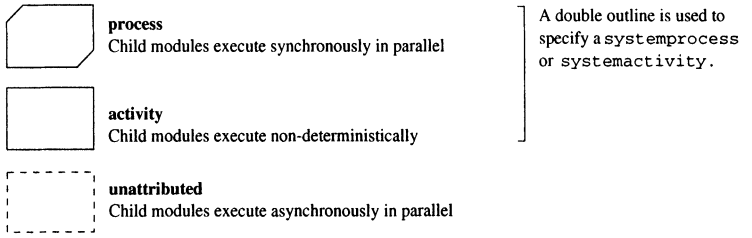
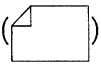
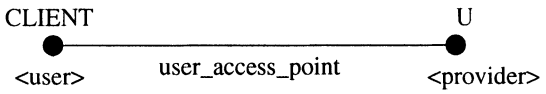


Figure 1.2 Specifying concurrency between children modules in the Estelle/GR

Child instance boxes should not be confused with channel declaration boxes, shown as a rectangle with a 'folded' corner (). These define channels for the containing instance.

Communication links. A major feature of the nested box notation is the inclusion of communication link information. Channels and interaction point definitions are semantically grouped by a vector terminating at two "dots". The extracted link attached between the `User` and `Responder` modules in Figure 1.1 has the following characteristics:



The solid '  ' defines an interaction point with an individual queue (a common queue is shown by a '  '). The vector label defines the channel shared by the interaction points, and the identifier in angled brackets defines the role of each interaction point. From these labels it is possible to look up what interactions each interaction point may send in the channel declaration. For instance, interaction point `CLIENT` may send interactions `READ_RQST` and `WRITE_RQST`.

1.4.2 Structure tree notation

A structure tree is a tree of generic modules. A generic module is itself visualized as a two level tree with the module header as root and associated bodies as children. The separation of module headers and bodies makes it possible specify multiple instance sub-hierarchies which the nested box notation cannot.

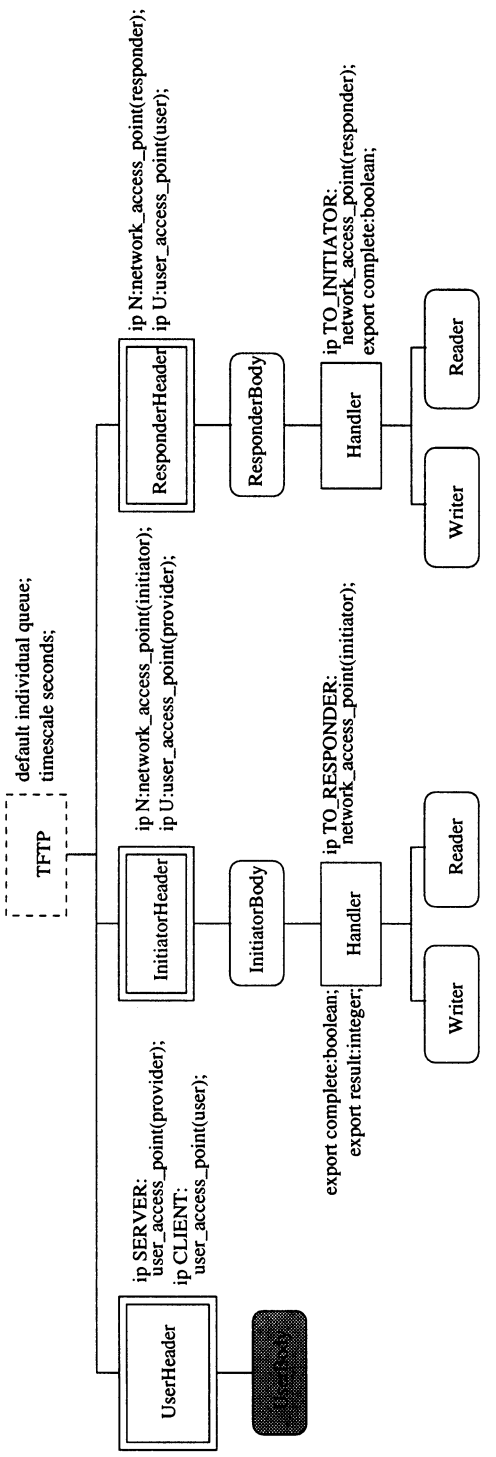
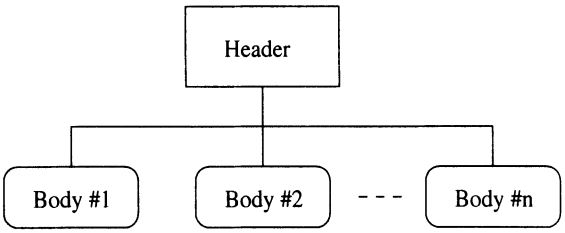


Figure 1.3 The TFTP structure tree

The tree is started using a special specification node and built downwards by adding generic modules as children of the module body in which they would be textually defined. As an example, the TFTP structure tree is illustrated in Figure 1.3.

Generic module representation. A generic module is graphically defined as follows:



The header definition part is named `Header`, and has bodies `Body #1`, `Body #2`, ..., `Body #n` associated to it. A module instance can be created from the pairing of `Header` with any of the bodies during execution.

A module body is always represented by a rounded rectangle, but a module header may be any of the nested box shapes listed in Figure 1.2, which determine the mode of concurrency between child modules.

Communication links. The structure tree cannot represent communication links as it defines a structurally *dynamic* Estelle system, in which communication links are only created during execution

The structure tree is limited to defining the static *communication interfaces* of module headers. These consist of `interaction points`, `exported variables` and `parameters`, and are shown as textual annotations of a module header node.

This can be awkward in even a simple protocol such as TFTP, as is obvious in Figure 1.3. This is one reason the module instance diagram is recommended to be used whenever possible.

1.5 BEHAVIORAL REPRESENTATION

The Estelle/GR uses a flow diagram notation to define individual transitions in the extended finite state machine making up the internal behavior of a module instance. This notation has been widely popularised by SDL, is easy to understand, and describes all details of a transition definition.

A flow diagram defining a transition in the Estelle/GR is called a *transition tree*, and the set of transition trees making up an EFSM definition is called a *forest*. Figure 1.4 shows a small forest of transition trees.

The basic structure of a transition tree is shown in Figure 1.5. The origin and destination states, the conditions necessary to fire the transition and the

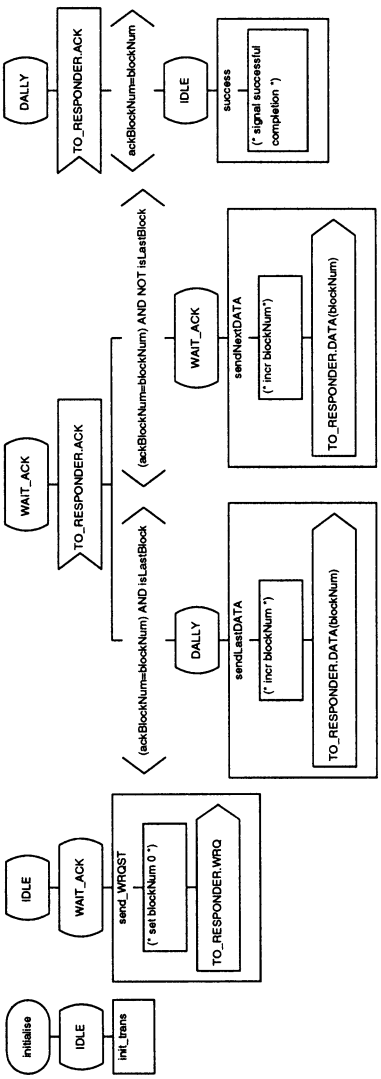


Figure 1.4 A subset of the forest of transition trees which define the `Writer` Handler of the Initiator (Figure 1.12 shows the simplified automaton of this forest)

actions taken on firing are explicitly shown as individual symbols in a flow diagram. The different parts of a transition tree are described below.

1.5.1 FROM and TO symbols

The FROM and TO symbols may be included as condition clauses in the condition part of the transition definition to allow nesting on common origin and destination states (see the section on nested transitions below). All the tran-

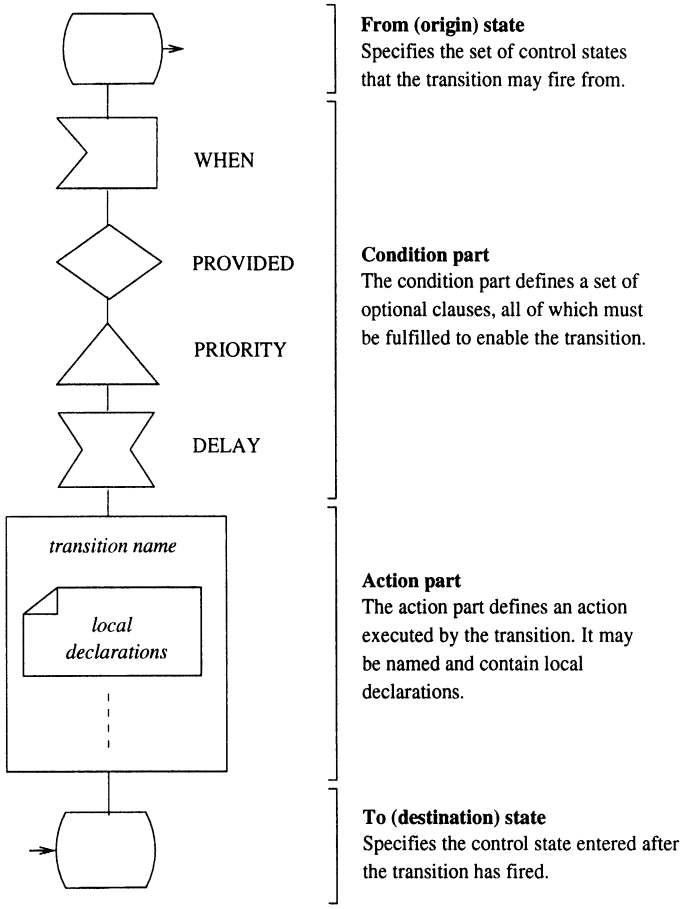


Figure 1.5 The structure of an Estelle/GR transition tree

sition examples in this section define the TO state as part of the condition part.

1.5.2 Condition part

The condition part is made up of optional condition symbols (listed in Figure 1.6) each of which defines a condition which must be satisfied before the transition may fire.

These conditions may occur in any order, but only once each. By treating the `FROM` symbol as a condition, it is possible to start the transition with any of the conditions listed in Figure 1.6. Combined with branched transition trees, this allows a number of transitions which respond to a common event to be semantically defined together (see the section on nested transitions below).

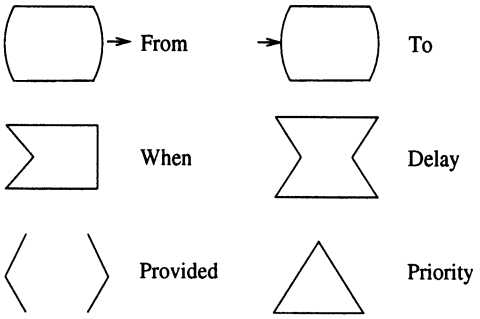
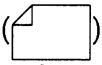


Figure 1.6 Estelle/GR condition symbols

1.5.3 Action part

The action part is separated from the condition part by a solid box, which also indicates that the enclosed actions are atomically executed. Transition naming is handled by the labeling of this box. The action part also has a scope in which declarations can be made. This is done textually within a graphical declaration symbol (), which is included in the action part box.

Unlike the condition part symbols, action symbols (listed in Figure 1.7), may be repeated. In the Estelle/GR we only define action symbols for Estelle-specific extensions to Pascal, and native Pascal statements are included in the graphical text symbol,  . There are two exceptions to this rule:

- The Estelle `exist-one` expression has no graphical representation, as it is used as an operand in boolean expressions;
- A Pascal `procedure call` is shown explicitly, due to its value in abstracting detail.

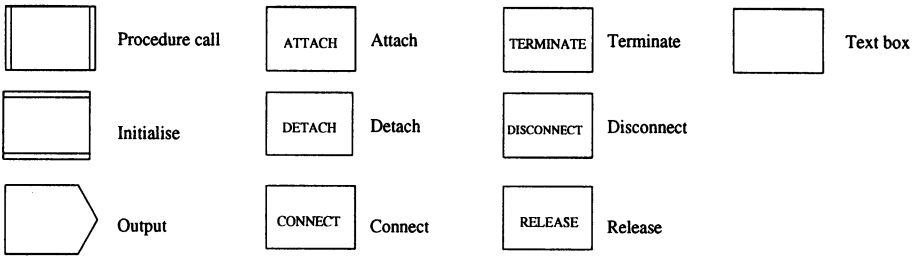


Figure 1.7 Estelle/GR action symbols

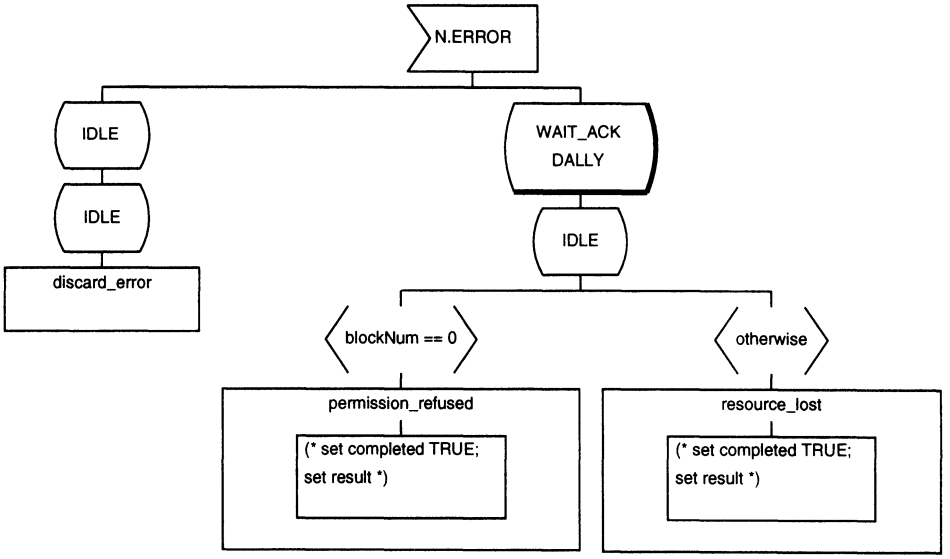


Figure 1.8 A transition nested on a `WHEN` condition, showing the specification of error recovery in the `Writer` handler of the `Initiator` module of the TFTP. The enboldened FROM state indicates that the transition may fire from more than one state in the EFSM

1.5.4 Nested transition trees

The term *transition tree* is used as an Estelle transition may be split into branches within the condition part. A branched transition tree defines a *nested* transition, as informally defined in the Estelle ISO standard. Within a branched transition tree, the conditions above a split apply to each branch created by the split.

The advantage of introducing branching into a transition tree is the ability to semantically group together transitions which result from a common event. For example, in the TFTP, error packets may be received at any time, in any state. Rooting a transition tree at a 'WHEN ERROR' condition, and branching the transition tree with different actions dependent on the current state, is a sensible way of specifying error-recovery behaviour. Figure 1.8 shows error recovery in the TFTP `Writer Handler` of the `Initiator`. The single transition defines three error-handling procedures depending on the state the machine is in, in a single transition tree definition. This is semantically more meaningful than defining three procedures in separate, not obviously related transitions.

1.5.5 The initialize transition

The initialize part of a module behaviour part is represented as a transition tree starting by a special initialize symbol, (initialise). The initialize transition for the TFTP specification module is shown in Figure 1.9. This initializes the

User, Initiator and Responder instances at the system level, and establishes the link between the Initiator and Responder.

Note that this initialization *is* statically determinable (there is only one configuration possible from the initialize transition), which is why the initial structure of the TFTP protocol can be described by an instance block diagram: the instance block diagram effectively interprets this initialization part However, when used to define a system, the instance block diagram implicitly defines this initialization part.

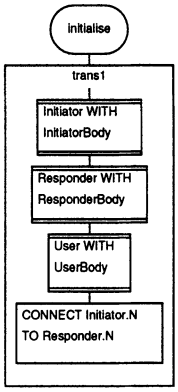


Figure 1.9 Initialize transition for the TFTP specification module

1.5.6 Special considerations

Estelle FORONE and ALL statements are represented as complex statements of other Estelle/GR symbols. Figure 1.10 show the templates for these constructs.

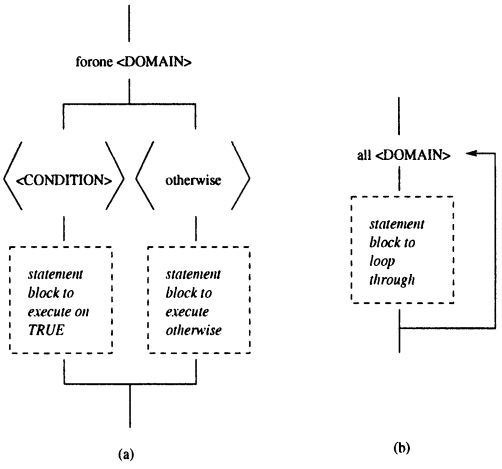


Figure 1.10 Estelle/GR complex statements: (a)forone and (b)all

The ANY macro is represented in a transition tree using a special symbol,  . It defines identifiers which are available to all symbols following it in the transition tree.

1.6 THE PROTOTYPE EDITOR

The prototype editor has two main functions:

- It is a syntax-directed editor for the definition of a specification using the Estelle/GR syntax.
- It translates a graphical Estelle/GR document into Estelle/PR code.

These two functions combined enable the editor to turn the Estelle/GR into a powerful graphical technique that can make use of existing tools supporting the Estelle/PR technique.

The editor also provides a navigational aid for complex module body definitions in the form of a *simplified automaton* view.

1.6.1 Editor structure

The editor provides separate interfaces for the definition of system structure and module behaviors.

Launching the editor opens the structure editor. The structure editor opens a single specification at a time, and controls the usual functions of file handling and printing, as well as the import and export of Estelle/PR specification documents. The structure editor is used to define a specification structure in the general *structure tree* syntax of the Estelle/GR. The *instance block* notation is not supported.

For each module body of the structure tree defined, a behavior editor can be opened in which the behavior of that module body is defined using the Estelle/GR *transition tree* syntax. Multiple behavior windows can be open simultaneously, making concurrent development of bodies possible, and enabling 'cut, copy and paste' editing between two bodies. The structure tree in the permanently open structure editor is a useful navigational aid for moving between a number of module body definitions.

The structure and behavior editors are very similar in presentation, and a screenshot of the behavior editor is presented in Figure 1.11.

1.6.2 Simplified automaton view

The behavior editor can display a simplified automaton view of its behavior definition in a separate window. Figure 1.12 presents the *simplified automaton* view of the set of transition trees defined in Figure 1.4.

The automaton is *simplified* as it does not display any transition information, and aggregates all transitions between two states into a single *macro-transition*, as defined in the work of GROPE [NP93, NP90]. The simplified EFSM therefore

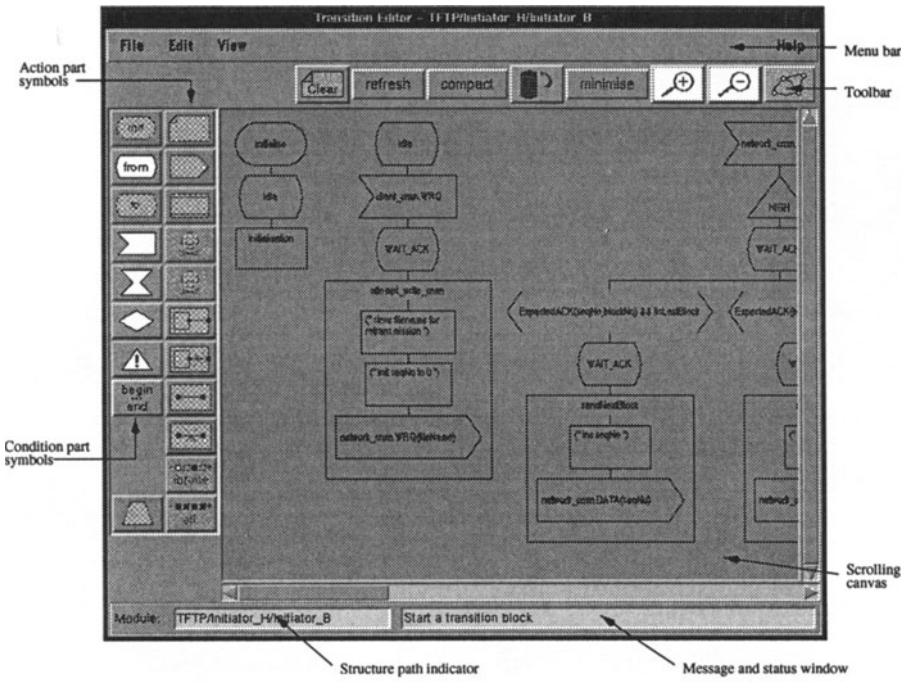


Figure 1.11 Screenshot of the behavior editor window.

consists of all states, but with at most one arc in each direction between each state pair, labeled with the names of the individual transitions aggregated on that arc.

The utility of the simplified automaton view is derived from the linking of each transition names in this view with its full transition tree definition in the behavior editor: When the transition name is selected in the automaton view, the transition definition is centered and highlighted in the behavior editor.

A module behavior definition may stretch over several virtual screens, whereas the simplified automaton will usually fit on one. This makes the latter a useful navigational aid when defining or exploring a module body behavior. In its simplified form it provides an overview of the transition definitions, and by way of the dynamic links, functions as an 'index' into the module body definition.

1.6.3 Ongoing work

The editor currently cannot import Estelle/PR specification documents. The instance block notation for specifying structurally static systems is not supported. These features are under development.

The editor will ultimately be integrated into the XEdt toolkit, making seamless use of the graphical technique and its set of Estelle/PR-based tools possible.

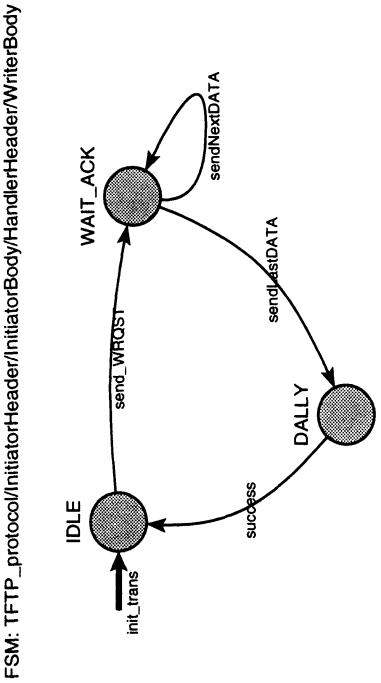


Figure 1.12 The simplified automaton view of the transition tree definitions in Figure 1.4 (Displayed sideways for easier comparison)

1.7 CONCLUSION

The recently completed MIL-STD-188-220A [FAS+97] Estelle specification is one of first Estelle descriptions to be included as an official part of a commercial formal specification. To encourage further acceptance of the Estelle technique, we believe that it is essential to make it accessible to a wider range of non-technical users. We believe that a graphical syntax and support tool is an important contribution to this process.

In the process of developing a graphical representation for Estelle, we found that it is more complex than similar techniques like SDL, as it permits a system to dynamically change its structure. This structural dynamicity means that the common "nested box" structural representation is not expressive enough to define an Estelle system, and a less expressive but more general *structure tree* notation had to be introduced. We retain the "nested box" notation in the

Estelle/GR though, for use the large subset of Estelle specifications which do not have a dynamic structure and for which it is definitive.

We believe that we have created a graphical syntax which will be immediately familiar to existing Estelle users, being built on the knowledge of existing Estelle and other FDT notations.

The functionality of the editor, and integration with existing Estelle/PR should also make the graphical technique a powerful and practically useful tool.

Acknowledgments

The authors wish to thank the anonymous reviewers for their constructive criticisms.

References

[BB87] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1), 1987.

[Chi94] Noel Chiappa. The TFTP protocol (revision 2). July 22, 1994.

[FAS+97] M Fecko, P Amer, A Sethi, M Uyar, T Dzik, R Menell, and M Mcmahon. Formal design and testing of mil-std 188-220 based on estelle. In *Proceedings of MILCOM '97*, Monterey, California, November 1997.

[ISO83] *ISO/IEC 7185, Information Technology - Programming Languages - PASCAL.* International Standards Organisation, 1983.

[ISO94] *ISO/IEC 7498-1, Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model.* International Standards Organisation, 1994.

[ISO97] *ISO/IEC 9074(E), Estelle: A Formal Description Technique based on a finite state transition model.* International Standards Organisation, 1997.

[ITU89] *SDL, Specification and Description Language, Recommendation Z100.* International Telecommunications Union, Geneva, 1989.

[NP90] D. New and Amer P. Adding graphics and animation to Estelle. *Information and Software Technology*, 32(2):149–161, 1990.

[NP93] D. New and Amer P. Protocol visualisation in Estelle. *Computer Networks and ISDN Systems*, 25:741–760, 1993.

[Pet62] C.A. Petri. *Kommunikation mit Automaten.* PhD thesis, Universität Bonn, 1962.

Justin Templemore-Finlayson graduated in the degree of Bachelor of Business Science and Computer Science from the University of Cape Town in 1996. On graduation he won a scholarship to follow an internship at the Institut National des Telecommunications in Evry, where he met Stan Budkowski and Jean-luc Raffy and who introduced him to the world of Estelle. This relationship developed into a Master of Science degree to develop a formal graphical representation for Estelle.

This research is supervised by Prof PS Kritzinger of the Data Network Architectures Laboratory in South Africa, and will be completed by the end of 1998.